Cooperative Autonomous Resilient Defense Platform for Cyber-Physical Systems

Mohamed Mahmoud Mahmoud Azab

Dissertation submitted to the Faculty of the Virginia Polytechnic Institute and State University in partial fulfillment of the requirements for the degree of

Doctor of Philosophy
In Computer Engineering

Hou, Yiwei Thomas,
Eltoweissy, Mohamed Youssef
Rizk, Mohamed Rizk Mohamed
Riad, Sedki Mohamed
Chen, Ing Ray
Yang, Yaling

Jan 25th, 2013

Blacksburg, Virginia

Keywords: Cyber Physical Systems, Security, Resilience, Cloud Computing, Autonomic Management

Cooperative Autonomous Resilient Defense Platform for Cyber-Physical Systems

Mohamed Mahmoud Mahmoud Azab

Abstract

Cyber-Physical Systems (CPS) entail the tight integration of and coordination between computational and physical resources. These systems are increasingly becoming vital to modernizing the national critical infrastructure systems ranging from healthcare, to transportation and energy, to homeland security and national defense. Advances in CPS technology are needed to help improve their current capabilities as well as their adaptability, autonomicity, efficiency, reliability, safety and usability. Due to the proliferation of increasingly sophisticated cyber threats with exponentially destructive effects, CPS defense systems must systematically evolve their detection, understanding, attribution, and mitigation capabilities. Unfortunately most of the current CPS defense systems fall short to adequately provision defense services while maintaining operational continuity and stability of the targeted CPS applications in presence of advanced persistent attacks. Most of these defense systems use un-coordinated combinations of disparate tools to provision defense services for the cyber and physical components. Such isolation and lack of awareness of and cooperation between defense tools may lead to massive resource waste due to unnecessary redundancy, and potential conflicts that can be utilized by a resourceful attacker to penetrate the system.

Recent research argued against the suitability of the current security solutions to CPS environments. ***We assert the*** **need for new defense platforms that effectively and efficiently manage dynamic defense missions and toolsets in real-time with the following goals**:

- Achieve asymmetric advantage to CPS defenders, prohibitively increasing the cost for attackers;
- Ensure resilient operations in presence of persistent and evolving attacks and failures; and
- Facilitate defense alliances, effectively and efficiently diffusing defense intelligence and operations transcending organizational boundaries.

Our proposed solution comprehensively addresses the aforementioned goals offering an evolutionary CPS defense system. The presented CPS defense platform, termed CyPhyCARD (Cooperative Autonomous Resilient Defenses for Cyber-Physical systems) presents a unified defense platform to monitor, manage, and control the heterogeneous composition of CPS components. CyPhyCARD relies on three interrelated pillars to construct its defense platform. CyPhyCARD comprehensively integrates these pillars, therefore building a large scale, intrinsically resilient, self- and situation- aware, cooperative, and autonomous defense cloud-like platform that provisions adequate, prompt, and pervasive defense services for large-scale, heterogeneously-composed CPS. The CyPhyCARD pillars are:

- Autonomous management platform (CyberX) for CyPhyCARD's foundation. CyberX enables application elasticity and autonomic adaptation to changes by runtime diversity employment, enhances the application resilience against attacks and failures by multimodal recovery mechanism, and enables unified application execution on heterogeneously composed platforms by a smart employment of a fine-grained environment-virtualization technology.
- Diversity management system (ChameleonSoft) built on CyberX. ChameleonSoft encrypts software execution behavior by smart employment of runtime diversity across multiple dimensions to include time, space, and platform heterogeneity inducing a trace-resistant

moving-target defense that works on securing CyPhyCARD platform against software attacks.

- Evolutionary Sensory system (EvoSense) built on CyberX. EvoSense realizes pervasive, intrinsically-resilient, situation-aware sense and response system to seamlessly effect biological-immune-system like defense. EvoSense acts as a middle layer between the defense service provider(s) and the Target of Defense (ToD) creating a uniform defense interface that hides ToD's scale and heterogeneity concerns from defense-provisioning management.

CyPhyCARD is evaluated both qualitatively and quantitatively. The efficacy of the presented approach is assessed **qualitatively**, through a complex synthetic CPS attack scenario. In addition to the presented scenario, we devised multiple prototype packages for the presented pillars to assess their applicability in real execution environment and applications. Further, the efficacy and the efficiency of the presented approach is comprehensively assessed **quantitatively** by a set of custom-made simulation packages simulating each CyPhyCARD pillar for ***performance and security evaluation***. The evaluation illustrated the success of CyPhyCARD and its constructing pillars to efficiently and effectively achieve its design objective with reasonable overhead.

Dedication

To my great precious Mom and Dad, for their unwavering support, encouragement, wisdom and guidance.

To my wonderful wife, for her patience and dedication.

To Youssef and Fattima,

For lighting up my life with their adorable smiles.

To my country that supporting me each step of the way.

Acknowledgments

Though the following dissertation and research work is an individual work, I could never have succeeded without the sincere help, support, guidance, and efforts of several individuals who in one way or another contributed in the realization of this work. First and foremost, I would like to express my deepest gratitude to my advisor Dr Mohamed Eltoweissy, for teaching me how to be a good scientist. Dr Mohamed has been my inspiration as I bypass all the obstacles in the way of completing this research work. His guidance helped me in all the time of research and writing of this dissertation. I am so proud that I had the chance to be one of his students, and I hope that one day I can be as a good advisor to my student as he was to me.

I would like to extend my sincere appreciation to Dr, Tom Hou, for his guidance, insight, and support throughout this research endeavor. His diligent efforts have created a unique working environment that made this work possible.

I would like to thank my committee members, Dr. Yaling Yang, Dr. Sedki M. Riad, Dr. Ing-Ray Chen, and Dr. Mohamed Rizk for reviewing this manuscript, and for providing me with their valuable comments and feedback that guided me to improve the quality of this dissertation.

I always dreamed to get my PhD from a prestigious university like Virginia Tech. This dream could have never come true without the efforts of Dr. Sedki M. Riad in establishing the VT-MENA program, and the VT-MENA Blacksburg-community. I would like to thank him and all the members of the VT-MENA program for giving me the chance of enjoying Virginia Tech and the lovely life here in Blacksburg.

Finally, I would like to thank my friends, and colleges for the encouragements, and support. I would like also to express my deep gratitude and appreciation to my family. Special thanks to my parents for their encouragement, wisdom and guidance, and for my wonderful wife for her support and patience. She devoted her talent in devising unique, impressive, and illustrative drawings that easily delivered the enclosed message without further description.

Table of Contents

Table of Figures

List of Tables

Chapter 1

Introduction

> "Research is to see what everybody else has seen, and to think what nobody else has thought." Albert Szent-Gyorgi

1.1 **Motivation and Problem Statement**

Cyber Physical Systems (CPS) are increasingly becoming indispensable to our critical infrastructure and defense domains, ranging from smart grids and smart healthcare to smart cities and smart warfare. CPS usually come with large-scale heterogeneous compositions of interacting cyber and physical components with differing capabilities and requirements. Securing these large-scale, distributed, heterogeneous compositions remains a challenge especially with the significant increase in cyber-physical attacker/attack sophistication.

CPS attacks usually target valuable infrastructure assets taking advantage of potential weaknesses in their defense systems. These weaknesses might arise from:

- **Large-scale heterogeneous compositions** of interacting cyber and physical components with varying capabilities and requirements
- **Increased automation** resulting in significant increase in volume of data flowing between cyber and physical processes exceeding the analysis and investigation capabilities of current defense solutions
- **Patching can't be fully automated in large-scale operational CPS** as operation and interaction occur at multiple temporal and spatial scales

- **Legacy compatibility** limits security system capabilities to deeply analyze and correlate network behavior at runtime
- **Isolated situation-oblivious defense service provisioning**
  - Cyber and physical security isolation could increase conflicts
  - Possible privacy policy violation limits sharing of information
- **Adversary asymmetric advantage**
  - Low cost of entry
  - Widely available resources
  - COTS security products makes it easy for attackers to discover possible security system flaws
  - Software monoculture facilitates attack re-application/diffusion

1.2 The BlackWidow attack scenario

To motivate our research, throughout the remainder of this document we will be referring to the following working scenario depicting a hypothetical CPS attack named the BlackWidow attack. The name came from the similarity between the operational characteristics and the destructive effect of the attack and the deadly BlackWidow spider.

Definition: The **BlackWidow malware** (BlackWidow for short) is our synthetic experimental attack that is designed to split into a set of code parts and spread in different directions and locations to decrease the probability of detection. The distribution of parts and the interconnection between the parts in different hosts weave a large web. This web is bi-

directionally traversed to send any harvested data from the attacked target and to update the malware with new tools and missions. The BW is designed to be as generic as possible; it is not oriented to any specific application. BW exploits system weak points “Ex, zero day exploits” to penetrate the system to spread its initial web seeds that will help in constructing the whole web. By constructing the BW web the attacker can starts to direct the BW towards its designated mission based on the attacker target. These directions might be remotely assigned through the internet or preprogrammed in internet inaccessible locations.

Using BlackWidow to facilitate border penetration

Attacker possible goals

- Espionage “Stealing secrets as a first wave to be used to construct the second wave”
- Take control of organization’s property for own gain
- Physical property manipulation

Attacker tools and capabilities

- Zero-day system exploits
- Social engineering methods to recruit insider agents via social networks
- Well trained and funded attackers
- Stolen certificates and digital keys
- Small lab to mimic the attacked system and its defense system

Design aspects

The attack is designed to be stealthy by hiding from the defense system sensors searching for attack signatures. The attack will target an intermediate host machine that will contain the worm

and command and control channel communications.

In order to do so, the worm is designed to not harm the host or change any of its settings that might raise the Anti-Malware (AM) alerts. The malware will use minimal resources and will work in a very slow fashion not to alert the network defense systems by its existence.

The only way to detect this malware is through deep analysis of the logs of all the communicating nodes, which is computationally very costly to the current systems that share the same host machines. Further, in order to deeply analyze and correlate strange communications patterns spreading all over the network, a global view for all the communicating entities within the network will be needed.

The malware is equipped with a self-destruct timer that automatically resets upon successful communication with the attacker. The self-destructive code adds to the sophistication of the attack that removes any traces of the worm and attacker actions, while inflicting damage to the target resources as a last resort.

The worm is later updated to use stolen digital certificates to authenticate its existence in the host machine in the form of drivers.

The malware is intended to be targeted, but due to the intentionally random deployment method, the code works in two modes as follows: (1) Benign mode where the malware infects other machines that do not belong to the target space. The machines might be used later in case of target change, or as a base for future attacks; and (2) Malicious mode, where the worm works only on the target host systems. The attacker feedback can determine the mode. The default will be benign unless the attacker changes that or predetermined targets have been programmed.

Attacker assumptions:

- The defense system shares the same network or host with the target of attack/defense system.[Note: defense system might be exposed to attack by compromising the ToD.]
- The attack target defense system, or major parts of it, uses COTS security products.[Note: A majority of defense systems are signature based, so that is probably easily to bypass with custom code.]
- The system is not capable of being fully situation aware of all its components in a massive-scale network in real time.
    - Building a very slow motion worm will increase the log file sample size needed to detect it.
    - The attack will spread in small parts in the target network hosted by geographically remote locations. This will make it more difficult to detect attacker activity unless a deep nearly network-scale analysis can be conducted to correlate all disparate logs.
- The defense system management workstations (that the administrators use) share the same network with the target of defense. [Note: Stolen passwords can simply be used to modify rules of IDS, routers, switches, firewalls, proxies, etc.]
- Attack hosts will not be manipulated in a detectable way so as not to alert the host AM. These hosts will be used only to launch attack on the primary target. [Note: this might be possible by using zero day exploits and malware code never seen before.]
- Host-based defense systems usually use malware signatures as an indication for infection from various forms of malware.
- It is not feasible to monitor all the host behavior patterns while sharing the same

workstation that is performing user tasks.

- Defense systems are not resilient against attacks, and have weak recovery mechanisms. [Note: most of them assume that they will not be the target of an attack as long as they were able to secure their ToD. Additionally, usually they have no intrinsic failure recovery.]

- Cyber security is oblivious of and is not coordinated with physical security to protect the target cyber-physical system. Human intervention is need to facilitate such coordination.[Note: the attack can make them conflict with each other to bypass both of them.]

1.2.1 *Homeland Security example*

Players:

The attacker is a sophisticated group working for XYZ's intelligence to facilitate *border penetration as a part of a military operation aiming to release captured prisoners by ABC military. These prisoners are held at a maximum-security facility close to the borders. The facility will be wiped out if necessary after releasing all the prisoners.*

ABC is a country with sprawling borders. ABC uses automated Unmanned Arial Vehicles (UAV) to monitor the country borders against possible penetrations. These border patrol UAVs are equipped with multiple sensors searching for possible indications of border penetration. Infrared cameras provide visual and thermal imaging represents the most important source of information that the UAVs provides to the border management department. The image is beamed via microwave link to a ground control stations; where a group of trained agents watches the surveillance feed to assure border safety.

ABC uses UAVs in multiple applications besides border patrol. Agriculture, rescue effort, aerial securities are examples of such applications. All these UAVs share the same control platform manufactured by EFG a worldwide company specialized in UAV design and manufacturing. EFG provides all the tools that will be used to design the UAV missions to its customers. Unfortunately, such tools share a common base as they are targeting the same controller provided by EFG.

The (synthetic) **BlackWidow malware** (BlackWidow for short) is an attack that is designed to split into a set of parts and spread in different directions and locations to decrease the probability of detection. The distribution of parts and the interconnection between parts in different hosts weave a large web. This web is bi-directionally traversed to send any harvested data from the attacked target to the attacker, and to update the malware with new tools and missions.

Border patrol operations using EFG's UAV (Normal condition)

Given ABC's vast sprawling borders, ABC's DHS has decided to buy a group of EFG's UAVs to be used primarily in border patrol. Due to the limitations of the radio communication range of this UAV, the DHS built a distributed set of ground stations to control such UAVs. These remote ground stations where operated by trained agents to control and program the UAV missions and visually inspect the UAV video feeds. In case of a positive detection of a border penetration attempt, the agents will mount appropriate respond to halt such attempts. Many of the ground stations were in remote areas. Communications with headquarters were limited to long-range radio. No direct communications between the ground stations computers and any other DHS department were possible. For that reason agents were allowed to use their personal computers or PDAs in break hours as means of entertainment.

Normally, agents are working in shifts, each shift is composed of 4 to 6 agents each one has

full control of a group of UAVs. The agent is responsible in programming and loading missions and monitor video feeds to/from the UAVs. At the end of the day agents hock their personal computers to the station's network and starts enjoying their break hours by constructing a tournament in one of the recent video games. The winner shall be eligible for a long vacation next week after the last game in the tournament.

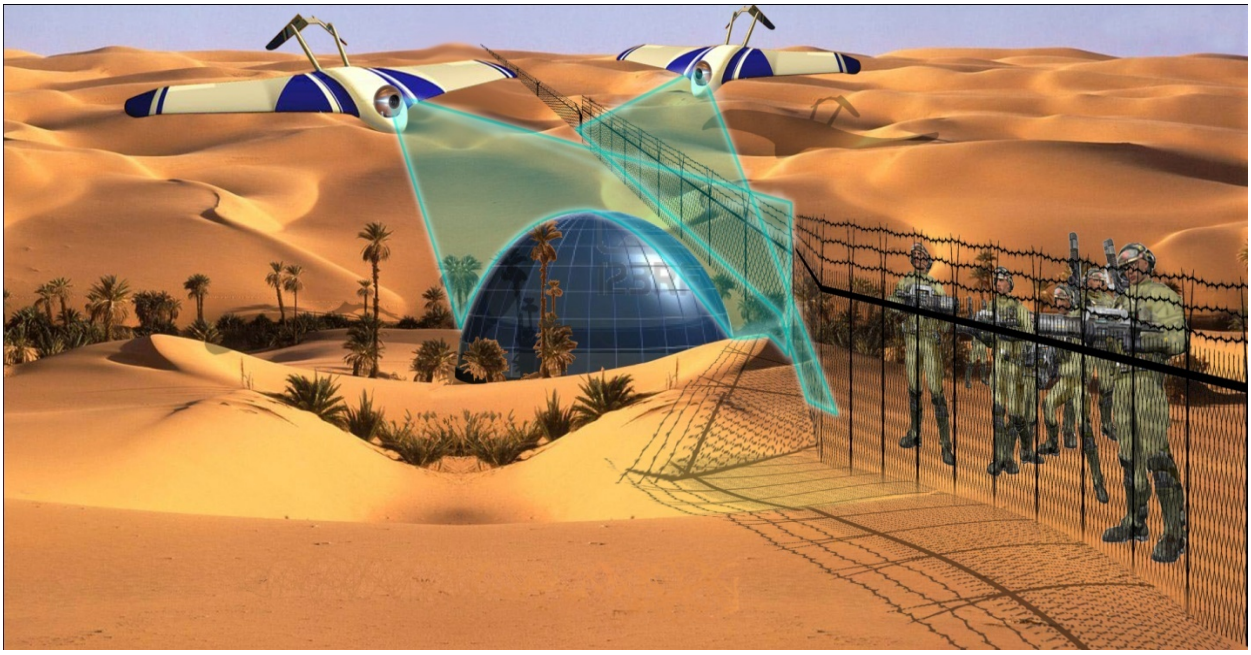


Figure 1.1 Border patrol attack scenario

Attack procedure

This attack will be executed as follows.

Phase 1

- Attack on the UAV manufacturer EFG to steal the design files and access codes.

Phase 2

- Use the design files and the access codes to generate a patch for ABC's UAVs mission files to manipulate the mission to follow a predetermined plan.

- Construct a test lab using one or two UAVs to test the attack
- Hack into the control stations that control the UAVs and patch the mission files.

Details

Phase 1

The attacker uses phishing attack that targets users' emails and social network personal pages. The attacker uses social networks as a source of information to generate more convincing phishing emails. These emails will be directed from one of the closely related contacts to the victim.

The attacker selects a group of employees working in different branches of EFG. These branches are distributed in various geographical locations, and the victims that will be the malware couriers have no direct relation with each other. This will increase the chance of the attack's success in case that the same phishing technique is used with different targets. The BW is programmed to search the user network for connected computers then it starts using one of the zero days exploits to clone itself into these computers.

The attack victims will receive parts of the malware. Each of these parts will contain a fraction of the designated mission and a simple communication module. The communications module will be used to open a direct channel with the attacker and to search and establish communication with other parts. Directions to other parts' locations might be sent by the attacker to minimize the search time.

The attacker uses malware fractions to construct logical executable entities in the form of mobile software agents targeting different objectives. The first objective will be to search and infiltrate the network for data stores.

The malware will sniff network traffic searching for predetermined signatures for such locations. The second objective will be to attack such data stores using the zero day exploits and the stolen certificates to locate targeted industrial secrets (the UAV design files, mission file design, and any available access codes). The malware will frequently update the attacker of its findings based on a predetermined update methodology.

Phase 2

The attacker will use the data received to generate a patch file that will manipulate the mission files adding predetermined entries representing a set of tasks to be activated exactly on a predetermined hour. These tasks include the following:

- Control the UAV surveillance feed source
- Clone the feed on the UAV internal storage
- Change the feed source from a live source to playback

The attacker uses one or two UAVs to test the patch file before using it on XYZ computers to make sure of its success. This should be easy as all EFG products share the same controllers and mission design file format. XYZ can buy these UAVs claiming that it will be used for any nonmilitary application.

After successful design and testing of the patch file, the BW will carry the patch file to the targeting ABC ground control stations.

The attacker uses the same phishing attack used in “phase 1” targeting agents working on the ABC targeted UAV control stations to facilitate infecting the control stations computers with the BW. A recruited insider might facilitate such infection in case of previous attack failure.

After successful infection, BlackWidow will search for machines holding the mission design files. Upon successful determination of their location, BlackWidow will target these locations. After successfully infecting these computers, the BW will schedule three executions of the patch file with three different predetermined zero hours. The goal behind that is to give the XYZ troop some flexibility to select the most convenient time to penetrate ABC borders.

After successfully patching the mission files and upon the first use of these files to update the UAV missions, the UAV executes the mission tasks as expected. Then 30 mints before the predetermined zero hour the UAV will start recording the video feeds to its internal data store. The recorded video file will be used to replace the life feed for 30 mints at the exact zero hour. Within these 30 mints the XYZ troops will make use on the area not under surveillance to cross the ABC borders to execute the designated rescue mission. After exactly 30 mints the UAV will maintain the life video feed and erase the recorded file.

Figure 1.2 Border patrol successful attack

1.2.2 *Commercial security example*

Attack specific goals

- Operation disruption to cause losses
- Launch same (low cost) attack on competitors to maximize gain

Attack procedure (on Air-gapped Target)

The attacker uses phishing attack that targets users' emails and social network personal pages. The attacker uses social networks as a source of information to generate more convincing phishing emails. These emails will be directed from one of the closely related contacts to the victim.

The attacker selects a group of employees working in different branches of ABC. These branches are distributed in various geographical locations, and the victims that will be the malware couriers have no direct relation with each other. This will increase the chance of the attack's success in case that the same phishing technique is used with different targets. The BW is programmed to search the user network for connected computers then it starts using one of the zero days exploits to clone itself into these computers.

The attack victims will receive parts of the malware. Each of these parts will contain a fraction of the designated mission and a simple communication module. The communications module will be used to open a direct channel with the attacker and to search and establish communication with other parts. Directions to other parts' locations might be sent by the attacker to minimize the search time.

The attacker uses malware fractions to construct logical executable entities in the form of

mobile software agents targeting different objectives. The first objective will be to search and infiltrate the network for data stores.

The malware will sniff network traffic searching for predetermined signatures for such locations. The second objective will be to attack such data stores using the zero day exploits and the stolen certificates to locate targeted industrial secrets, and any available access keys to the protected area behind the air gap. The malware will frequently update the attacker of its findings based on a predetermined update methodology.

After successful reception of this data, the attacker will use it to generate legitimate keys to access the air gap.

The attacker will use the malware to locate the workstations controlling the surveillance cameras. In locations with no surveillance cameras the malware might use any available user connected web cameras. The malware will record periodic video feeds to be sent to the attacker. These videos with the help of the attacker generated access keys will guide a recruited insider into infecting the air gap with a copy of the BlackWidow.

The malware controlling the video cameras will make sure that this process will not be recorded on any of the cameras to protect the recruited insider.

The air gap malware is programmed to increase the operational hours of certain machines that use specific raw materials manufactured by XYZ to increase XYZ profits. The malware can easily identify such machines by searching a predetermined fixed identifier that must be added to all the programming files targeting such machines. Further, the attacker will use the stolen secrets and designs to equip the malware with the needed logic to randomly manipulate the operational motors frequency in the production machines to induce random defects in the output

products to lower its quality. Doing so shall cause multiple financial problems to ABC. XYZ shall benefit from ABC's loss due to its low quality products. Additionally XYZ will maliciously gain both financially and more control over ABC's production lines by, for example, carefully adjusting the amount of consumed and supplied raw materials.

Figure 1.3 Commercial security example

1.3 **Research Approach**

Recent research argued against the suitability of the current security solutions to CPS environments. We assert the ***need for new defense platforms that effectively and efficiently coordinate defense missions and tools in real-time to achieve the following goals:***

- ❖ Achieve asymmetric advantage to CPS defenders, prohibitively increasing the cost for attackers;
- ❖ Ensure resilient operations in presence of persistent and evolving attacks and failures; and
- ❖ Facilitate defense alliances, effectively and efficiently diffusing defense intelligence and operations transcending organizational boundaries.

Our proposed solution aims to comprehensively address these goals in order to present an evolutionary defense platform that would enable self and situation awareness, resilient adaptive defense, and cooperative autonomous control and sharing amongst cooperating organizations without violating their individual privacy policy. Enabling such features makes it possible to successfully provision defense services to mission critical heterogeneously composed systems like CPS, while maintaining the operation timeliness and stability in presence of persistent attacks.

In this work, we present CyPhyCARD (Cooperative Autonomous Resilient Defense platform for Cyber-Physical Systems) - a biologically-inspired distributed dynamically configurable, runtime programmable platform that manages a large number of cyber and physical resources and services upon which evolutionary defenses can be built to protect participant organizations. Figure 1.4 presents an abstract view of CyPhyCARD.

Figure 1.4 Abstract view of CyPhyCARD

CyPhyCARD features a set of platform-managed capabilities and services through a biologically-inspired architecture and methodologies to effect trace-resistant, resilient, and allied defenses. CyPhyCARD provisions its services via an evolutionary sensory system, EvoSense, working through an intrinsically resilient and autonomously-managed adaptable platform, CyberX, and protected by novel moving-target defense mechanism, ChameleonSoft. Figure 1.5 presents CyPhyCARD goals and features which are described as follows:

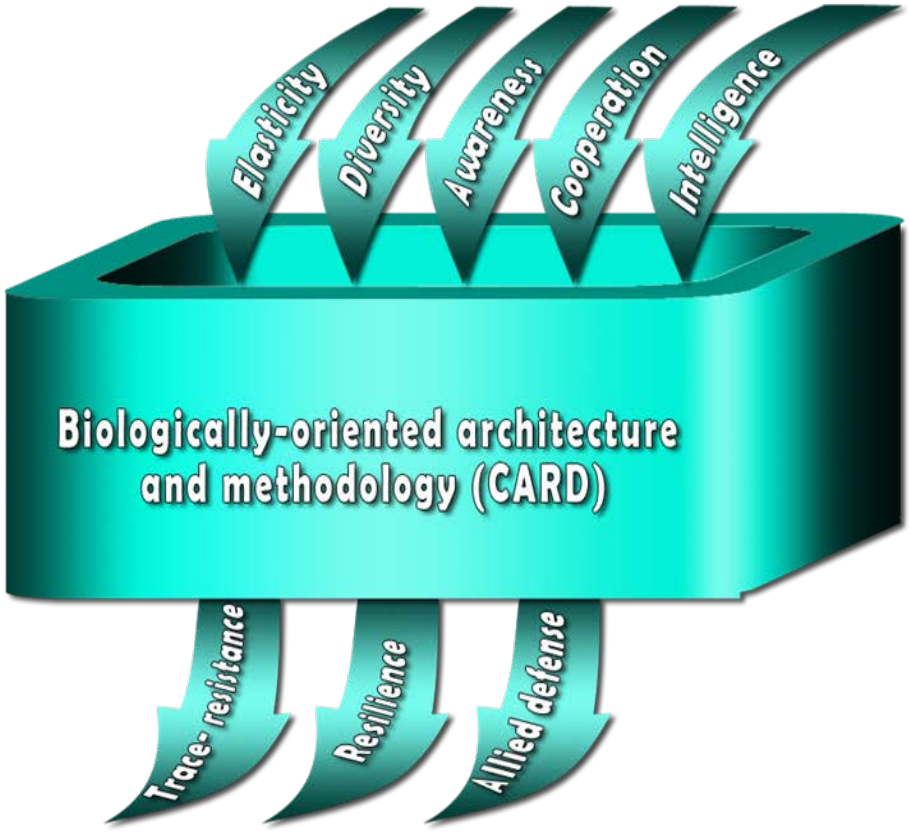


Figure 1.5 CyPhyCARD goals and features

Goals:

- Resilient operations: by managing automatic failure recovery and containment, and adapting structure, function and performance to varying network scales and contexts;
- Trace-resistant moving-target defense: by multidimensional mobilization of the attack target evading attackers; and
- Allied defense: by isolating the defense provisioning design concerns sensing, effecting, control, and physical resources; enabling trustworthy automated defense sharing and cooperation.

Features:

- ❖ Awareness: by providing pervasive monitoring and analytics for self and situation awareness distributed throughout the targeted systems;
- ❖ Elasticity: right-sized resources and services by autonomically marshaling and adaptively provisioning resources (cyber and physical) and services (monitoring, detection and response) to effect appropriate evolutionary immune responses;
- ❖ Intelligence: by using autonomic, independent, self- and situation-aware, smart building blocks to build the entire defense platform;
- ❖ Diversity: to induce software behavior encryption (i.e., inducing adequate confusion and diffusion similar to message encryption); and
- ❖ Cooperative defense: by enabling mixed initiative and fully autonomic cooperative tipping and cueing among participating organizations without violating their individual policies.

Hypotheses:

- ❖ Resilient operation in presence of attacks/failures can be significantly enhanced by:
  - Utilizing intrinsically resilient, online programmable, composable building blocks
  - Mobilizing software for encrypted execution behavior (moving target)
  - Enabling resource- and context-aware automated recovery
  - Enabling trustworthy information sharing
- ❖ Efficiency of defense services can be significantly enhanced by:
  - Providing a generic platform for provisioning defense services
  - Tailoring resource consumption based on the task on hand
  - Enabling online programmability and re-tasking

- Provisioning defense services in isolation from Target of Defense (ToD) operations
- Context-aware automated resource management and control

Underlying assumptions:

- It is feasible to decouple application logic, state, data and physical resources
- The capability exist to produce functionally-equivalent behaviorally-different code modules targeting different quality attribute objectives
- Applications can be defined in terms of interacting entities

Design principles:

- Platform managed
  - Replication and automated recovery
  - On demand resource acquisition, allocation, de-allocation, and sharing
  - Dynamic, situation aware , real-time adaptation to changes
- A smart isolation layer isolating and exclusively interfacing between the defense platform and target of defense
- Self- and situation- aware composable basic building blocks that autonomously manage the underlying logical and physical resources
- An autonomously-managed trace-resistant moving-target defense securing the infrastructure
- Autonomous privacy-preserving information sharing and exchange

1.4 **Evaluation**

For the purpose of evaluating the efficiency and applicability of the presented approach, we constructed a prototype of the defense platform three pillars CyberX, ChameleonSoft, and EvoSense. Our studies also include the design and implementation of different simulation packages simulating each of the presented pillars. These simulators were utilized to quantitatively evaluate the platform's various performance and security aspects.

We devised a digital version of the Cell as a basic building block for CyPhyCARD pillars. We used the synthetic Cell to build a prototype of ChameleonSoft behavior encryption mechanism along with CyberX automated system adaptation and a multimodal recovery system. The prototype illustrated ChameleonSoft capability to encrypt the runtime execution behavior of software. The prototype also illustrated CyberX success to autonomously adapt to frequent changes and to autonomously detect and recover Cell failure. Further, we devised a set of sensors and effectors Cells and a circulation management platform as a part of EvoSense prototype. The Cells are deployed on heterogeneously configured hosts and the feedback was analyzed to detect and resolve a preexistent system malfunction. The prototype illustrated EvoSense success in isolating the defense concerns "sensing, effecting, control, and physical resources", its ability to circulate defense tools on various hosts regardless of its configuration, and its capability to collect privacy friendly feedback regarding specific incident.

Additionally, we designed and implemented three simulation packages using MATLAB to evaluate the performance and security aspects of CyberX, ChameleonSoft, and EvoSense. The results extracted from these packages showed that the three pillars were effectively and efficiently successful in achieving their design objectives. However there were tradeoffs between

increasing the level of provisioned security of the system and maintaining the performance quality at all times. The results also showed that when we utilize the system adaptive and elastic features we can simply tolerate such tradeoffs in a way that satisfies almost all of the targeted quality attribute objectives by the platform at all times.

Finally, we devised a synthetic attack scenario in order to conduct a qualitative study of CyPhyCARD's security effectiveness in provisioning defense services to CPS applications. The study illustrated that CyPhyCARD has the capability and the tools to efficiently mitigate that attack with minimal consequences. We also clarified the role of each of CyPhyCARD's pillar in defeating the attackers list of assumptions and hypotheses supporting their attack. We surmise that by invalidating such assumptions, the attack itself will no longer be feasible.

1.5 **Contributions**

Intellectual Merit: To realize these capabilities, CyPhyCARD construction is based on three main contributions:

- Biologically-inspired Management Platform (***CyberX***)
  - Manages a distributed construction of composable basic building blocks termed "Cells".
    - Enable Cell dynamic runtime configuration
    - Support the Cell self-monitoring
    - Enable the Cells to dynamically adapt to changing internal and external conditions and acquire resources on demand based on the dynamics of the tasks on hand

- A multimode, autonomous situation-aware recovery system for enhanced system resilience

❖ Software Behavior Encryption System (***ChameleonSoft***)

- Employs runtime multidimensional software diversity to induce confusion and diffusion to, in effect, induce spatiotemporal software behavior encryption
- ChameleonSoft mobilize running Cells among heterogeneously configured hosts in a way that makes the attack target in a continuous random motion inducing trace-resistant moving target defense .
- An elastic software platform that dynamically and autonomously change diversity application and recovery policies to match the surroundings frequent changes

❖ Evolutionary Sensory System (***EvoSense***)

- Defense service provisioning by autonomous abstraction and virtualization of heterogeneous compositions of physical resources, conventional defense services, and autonomously customized formations of sensing and effecting tools
- Enable smart pervasive sensor circulation for enhanced detection efficiency and better resource utilization
- Early enable trustworthy cooperative autonomous control and sharing of defense intelligence amongst interconnected CyPhyCARDs and/or Target of Defense (ToD) systems to enhance attack detection and deterrence

Broader Impact: CPS integrate computational and physical processes. Modernizing the critical infrastructure often involves upgrades with CPS to enhance efficiency, safety and reliability. New security and resilience requirements arise given the mission- and time-critical nature of

infrastructure systems, the massive-scale of enterprise and field deployments of, often resource-constraint, CPS devices, and the emergent behavior and interactions between the interconnected CPS components. Further, the expected profound increase in sophisticated persistent attacks targeting high-value infrastructure assets and exploiting the potential vulnerabilities of the new cyber-physical integration poses formidable challenges. The broader impact of our work includes:

- Evolutionary comprehensive defense system capable of providing defense services to mission critical CPS
- Enable continuity of operations as well as moving target defense to prohibitively increase the cost on potential attackers
- Automated trustworthy multi organization information sharing enabling early attack alarm and enhancing decision making accuracy
- Enable building intrinsically resilient, resource efficient, and adaptable software
- CyPhyCARD provides defense, resilience, security as services

Unified platform to monitor, manage, and control heterogeneously composed CPS components expanding their applicability in multiple domains

1.6 **Document Organization**

The remainder of the dissertation is organized as follows: Chapter 2 presents details about CyPhyCARD platform and CyberX, Chapter 3 describes ChameleonSoft behavior encryption and moving target defense for platform security and resilience, Chapter 4 presents EvoSense for defense service delivery and sharing, Chapter 5 presents our evaluation approach and models, simulation framework, and evaluation results, Chapter 6 overviews related work, and finally Chapter 7 concludes the dissertation and highlights future work.

Chapter 2

CyberX: Biologically-inspired CyPhyCARD Management Platform

> "Simple things should be simple and complex things should be possible." Alan Kay

2.1 **Introduction**

Today, cyber systems form the backbone of national critical infrastructures, which means that a major security incident on such systems could have significant disruptive impact on the operation reliability and safety of many of the systems that we rely on to maintain our everyday life. Both researchers and practitioners have been paying considerable attention to the cyber security problems for more than two decades. However, the problems are far from being comprehensively solved. The main challenges facing the current cyber security practice is that the security approach is largely heterogeneous, increasingly complicated, and it is struggling to keep pace with quickly evolving threats. The CARD concept is presented to inherently address such challenges.

To achieve the CARD vision and simultaneously improve the nation's cyber security posture, the CARD should support a portfolio of defense techniques that when homogeneously-composed into one solution (CyPhyCARD) would enable adequate and trustworthy defense provisioning. In our work the CARD encompasses trace-resistant moving-target defense, resilience against failures and attacks, and autonomous trustworthy allied-defense. We surmise that enabling the CARD

would require software development, management, and operation to be based on five main pillars: elasticity, diversity, awareness, cooperation, and intelligence.

Currently software products depend mostly on static or partially dynamic architectures where data, logic, and/or physical resources are primarily tightly coupled. Multiple attempts have been presented in the literature to partially decouple these design concerns [4, 5, 6]. However, up to our knowledge our Cell Oriented architecture (COA) is the only architecture that comprehensively supports intrinsic separation of design concerns needed for runtime re-programmability, intrinsic autonomic online composability, and dynamic software adaptation and elasticity.

In this chapter, we propose CyberX, a situation-aware trustworthy management platform that utilizes the COA features to realize the aforementioned pillars. COA is a biologically-inspired architecture with active components termed Cells that support development, deployment, execution, maintenance, and evolution of software. Cells separate logic, state and physical resource management. Cells are realized in the form of intelligent capsules that encapsulates executable applications defined as code variants. Cells are dynamically composable into organisms that are bound to functional roles at runtime. CyberX manages such construction to enable online re-programmability, hot code-swapping, local/global situation awareness, and automated recovery.

CyberX enables applications to dynamically adapt to serious runtime changes in their execution environment via a runtime diversification of multiple functionally-equivalent, objectively-different "targeting different quality attributes" code variants. Reliability, performance, robustness, reliability, survivability, compatibility, scalability, and mobility are examples of such attributes. Currently we are in the process of using the same technology to

enhance the system resilience against software attacks. Our objective is to employ spatiotemporal diversification of similar-function different-behavior code variants for moving target defense.

CyberX utilize the COA feature of enabling the application to exchange real-time status and recommendation messages with the host Cell for administrative purposes to enhance the Cell local application awareness and to enable application driven adaptation. CyberX use these messages to guide the Cell runtime quality-attribute manipulation towards accurate and prompt adaptation. Further, CyberX collects, analyze and trustworthy-share these messages and status reports constructing a real-time sharable global view of the Cell network.

CyberX enhances the system resilience by multiple recovery modes to cover different application-requirements and host-configurations. CyberX offers a prompt and accurate fine-grained recovery for resourceful hosts executing critical applications, and a more resource efficient course-grained recovery for less critical applications. CyberX uses the COA loosely coupled features to allow applications to seamlessly change their current active recovery modes based on context, environment, or application-objective change.

CyberX contributions presented through this chapter are as follows:

- A biologically inspired architecture with the following capabilities:
  - Intrinsic separation of design concerns (data, logic, and physical resources); and
  - Employing a mission-oriented application design and inline code distribution to enable adaptability, and online dynamic re-tasking;
- Elastic system design and platform-managed control enabling the following:
  - Runtime diversity employment for hot manipulation of quality attributes to effect trace-resistance and moving target defense;

- Multimodal, autonomous situation-aware recovery system for enhanced system resilience; and
- Dynamic and autonomous change of shuffling and recovery policies according to run-time changes in the execution environment.

2.2 **The Cell Oriented architecture**

The COA is an employment of a mission-oriented application design and inline code distribution to enable adaptability, dynamic re-tasking, and re-programmability. The Cell is the basic building block in COA. The COA Cell is inspired from the biological Cell in its independent, generic, composable construction. COA Cell is an abstraction of a mission-oriented autonomous active resource. Generic Cells termed stem-Cells, are seamlessly created by the host-side middleware or the COA Cell DNA (CCDNA). Further, they participate in emerging tasks through a process called specialization. The CCDNA is a middleware program that allows a physical workstation to host Cells and facilitates Cell physical resource allocation and management.

We envision applications built over COA as a group of cooperating roles representing mission objectives. The term organism is used to represent a role player that performs a dedicated mission. An organism might be composed of a single or multiple Cells based on its objectives. Figure 2.1 illustrates the different components of the COA.

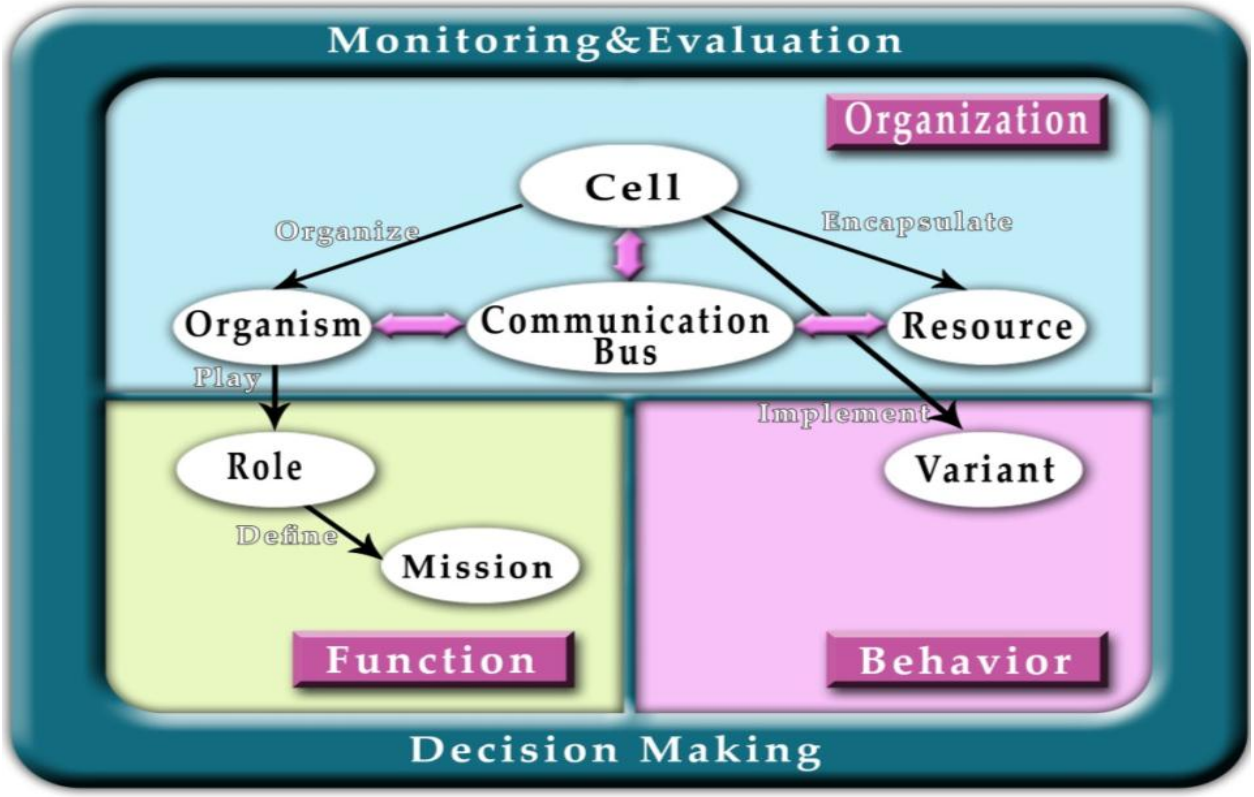


Figure 2.1 Components of our COA

2.2.1 *The Cell*

Conceptually, the Cell is the smallest active resource in a distributed computing platform. Cells are intelligent, and independent, autonomous, single-application capsules “sandbox” that acquires, on the fly, application specific functionality in the form of an executable code variant "The specialization process". Cells act as a simple virtualization environment isolating the executable **Logic** from the underlying **Physical** resources. Figure 2.2 illustrates an abstract view of a COA Cell at runtime. The Cell is dynamically composable into larger structures “organisms” representing complex multi-tasking applications.

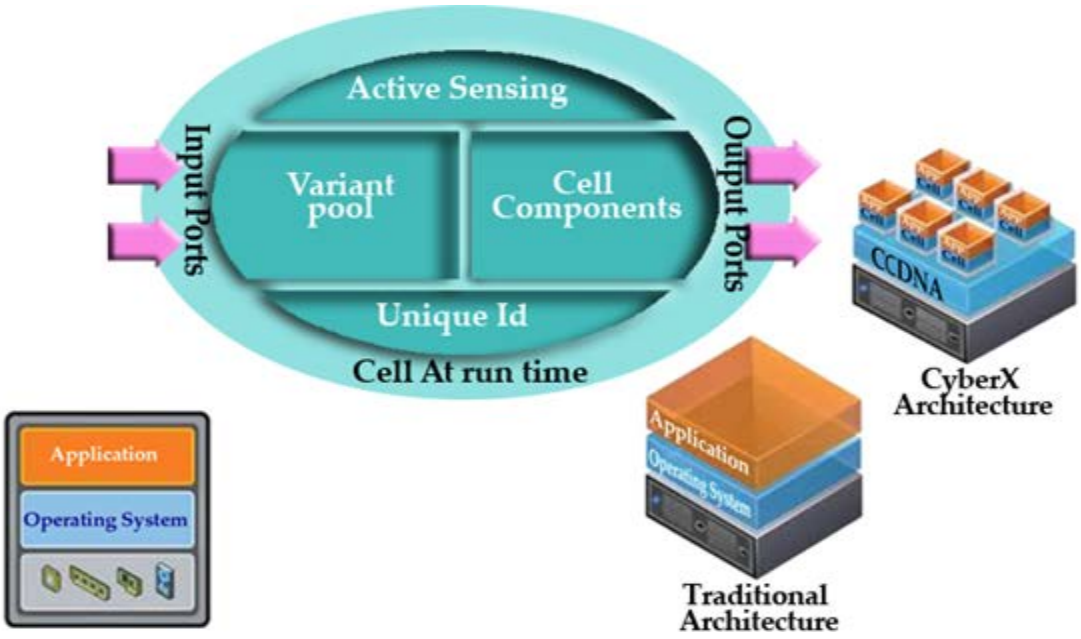


Figure 2.2 COA Cell at runtime

A single workstation can host one or more Cells, providing a flexible way to share the physical resources among multiple applications. Figure 2.3 illustrates the main components of the COA Cell briefly described as follows.

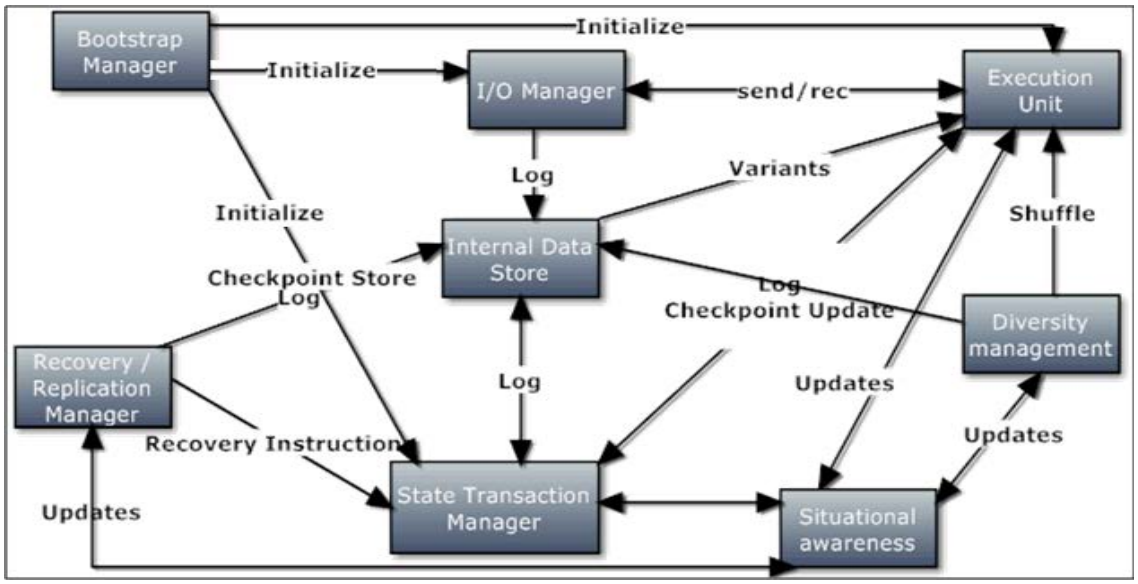


Figure 2.3 The Cell

Cells are instantiated at bootstrapping when the bootstrap manager initializes the Cell components and ports with the appropriate parameters based on the bootstrap context. The communications unit (I/O manager) handles local and remote I/O communication setup, I/O logging, and IP/Port/Virtual naming resolution.

The specialization process occurs when the execution unit receives an executable COA-ready code variant that represents the application specific functionality that the Cell should acquire.

A COA-ready variant is a program that enables check-pointing and frequent reporting through a predetermined channel using predetermined syntax. We isolate the **Data** from the **Logic** by necessitating that all sensitive data is committed to a remote data storage using a dedicated data channel provided by the infrastructure before each checkpoint. The program must ask for, and start execution from an infrastructure provided starting point. This point is zero for fresh Cells. Finally, the programmer has to provide At least two similar-function different-objective variants to enable CyberX quality attribute manipulation.

The execution unit starts by launching the selected variant with the appropriate parameters “Ex., the Cell Id”. The execution unit is also responsible for the termination and replacement of the executing variants based on incoming shuffling commands. All the issues regarding diversity employment-methodology, shuffling policy, “shuffling frequency, commanding, and variant selection” are the responsibility of the diversity-management unit.

The State Transaction Manager (STM) is responsible for monitoring the variant execution progress. It is the only unit with direct access to the executing application through a dedicated communication channel. STM reports checkpoint change and other incoming application requests and status reports to the appropriate units “ex, holding shuffling frequency change, objective change requests, etc”.

The recovery manager is responsible for adjusting the recovery settings, recovery mode change, in addition to restoring and synchronizing checkpoints at the time of failure-recovery with the cooperation of the execution unit. It is also responsible of sending the Cell beacon messages to the tracking servers. These messages include the last checkpoint reported by STM, and other reports regarding Cell state reported by the situational awareness unit; and any other administrative messages needs to be delivered to the Global Management Servers (GMS). The details about CyberX multimodal failure recovery processes are illustrated in section 4.

The situational awareness unit, is responsible for providing the needed situational and context awareness information to the other Cell units to support their decisions. It monitors the internal and the external Cell surroundings and generates guideline reports for all Cell units. It also informs the GMS with awareness reports through attached messages to the Cell frequent beacon messages. GMS use these stored beacons to generate more meaningful status reports. These reports contain information, directions, and commands that CyberX wants to deliver to a certain area in the network. For example, if one of the Cells reported a malicious event that might affect other neighbor Cells, GMS might inform other Cells to change the current variant to more secure variant.

The decision-making tasks are totally distributed in the Cell. Each unit takes its own decisions regarding its specific task autonomously. The global operation of the Cell is handled by the real time cooperation between all these units.

2.2.1.1 The COA Cell VS biological Cell

We mimic biological systems in our work starting from the basic construction building blocks. One of the most successful biological ability to adapt to changes is realized in the Sea

Chameleons. Our investigation illustrated that the key to success for Chameleon diversity starts from its basic building block the biological Cell. In Table 2.1 we clarified by comparison the main similarities and differences between the biological Cell and our digital COA Cell. We also compared between CyberX and the Sea Chameleon identifying points of similarities and differences. Table 2.1 illustrates that we selected most of the useful features from these biological constructions, and enhanced some of these features to better serve our objective.

	Biology	COA Cell
Cell	Composable	Composable
	Polymorphic before specialization	Polymorphic
	Generic	Generic
	Specialize once “disposable”	Multiple specialization “disposable/reusable”
	Autonomous	Autonomous
	Adaptive before specialization	Adaptive
	Resilient	Resilient
Chameleon	Autonomous	Autonomous
	Independent	Independent
	Collaborative	Collaborative
	Adaptive	Adaptive

	Resilient	Resilient
	Multipurpose diversity utilization	Multipurpose diversity utilization
	Distributed diversity management and control	Distributed diversity management and control
	Locally and globally situational aware	Locally and globally situational aware

Table 2.1 Comparison between biology and CyberX "Cell"

2.2.2 *The Organism*

An organism is an autonomous logical execution unit that follows the logic patterns of role providers. A role is an interpretation of a dedicated mission dynamically assigned to organisms. An organism might comprise a number of Cells wired together dynamically (at runtime) to form software structure having an independent execution context, see Figure 2.4. For example, let us consider a distributed application defined as a set of tasks executing independently and communicate via exchanging messages. We see this application as an organism playing a role defined by the application objectives. The organism is composed of a set of Cells. Each Cell encapsulates and executes one of the application tasks defined by a set of code variants.

The simplest organism is composed of only a single Cell. A more complex organism may span any number of Cells that can be distributed among multiple physical computing hosts enabling hosts with limited capabilities to collectively participate in the execution of complex autonomous roles.

2.3 **The CyberX management platform**

CyberX is a situation-aware trustworthy management platform that utilizes the COA features to enable a wide set of features and capabilities. Online re-programmability, hot code-swapping, local/global situation awareness, and automated recovery are examples of such capabilities that participate in the realization of the CARD concept. In the next subsections we describe CyberX architecture, the main components participating in its construction and the functionality of each component. Further, we will discuss the communication aspects and security issues with and within CyberX.

2.3.1 ***CyberX platform architecture***

CyberX is composed of a set of central powerful nodes we will address them as servers. These servers cooperate autonomously to manage the whole network of Cells. This platform is responsible for the organism creation "composition and deployment of Cells", management, the host side API(s) "CCDNA", real-time monitoring and evaluation of the executing Cells, and recovery management. Further, it provides the necessary management tools for system administrators to manage, analyze, and evaluate the working Cells /organisms.

Auditing and Reputation Management Servers (ARMS); its main task is to monitor outgoing or incoming Cell administrative messages for the lifetime of the Cell. This information is used to assist evaluating the trustworthiness of the Cell. These servers cooperate with the recovery tracking servers, routing nodes to frequently evaluate the Cell behavior for any

malicious activities. These servers will hold comprehensive reports about each Cell for the lifetime of the Cell.

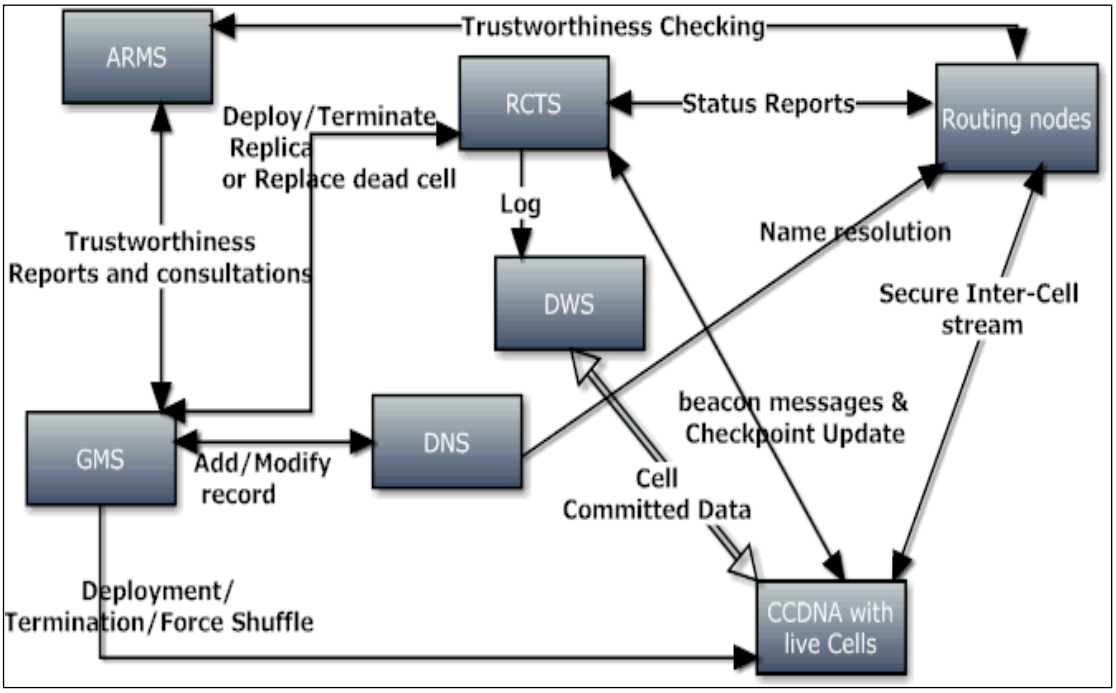


Figure 2.4 The management platform architecture

Recovery & Checkpoint Tracking Servers (RCTS); its main task is to monitor, and store checkpoints changes for all running Cells. Checkpoint updates are always enclosed as a part of the Cell frequent beacon message update. This server is also responsible for reporting failure events by comparing the duration between consecutive beacon messages to a certain threshold matching the reporting frequency settings of each Cell. Failure events are validated by comparing the recently noticed reporting-delay for a particular Cell to the average reporting-delay within its neighbors and other Cells hosted in the same host. A Cell failure notice is reported to the global management servers with the last known failure recovery settings, Checkpoint, and variant settings to start deploying replacement Cells.

Global Management Servers (GMS); its main task is to manage the underlying COA infrastructure. It is responsible of Cell deployment, coordinating between servers, facilitating and providing a platform for administrative control. It is the only server authorized of issuing Cell termination signals. It can also force Cell migration or change the current active recovery policy when needed. It is responsible of assigning the infrastructure global policy, routing protocol, auditing granularity, registering/revoking new hosts, and keeping/adjusting the host-platform configuration file.

The Data-Warehouse Servers (DWS), it is one of the main components of the infrastructure that participate in the separation between the Data, Logic, and Physical-resources. DWS are distributed through the Cell network, they are responsible for holding and maintaining all the data being processed, and any other sensitive data that the management units want to store. All running Cells are not permitted to store sensitive data "data processed and committed prior to a Checkpoint event" on their local memory. All sensitive data has to be remotely stored in a specific DWS serving the Cell area through the dedicated data channel. DWS synchronize their data independently.

Distributed Naming Servers (DNS), is responsible for resolving the real host IP/Port mapping to the virtual Cell Id and organism names. The working Cells use this mapping at runtime to direct incoming and outgoing communications. DNS is major player in the COA's separation of concerns that enables virtually seamless, Cell relocation, and workload transition in case of failure recovery. In case of Cell movement, the DNS will be instructed by the GMS to maintain communication redirection.

2.3.2 *CyberX trustworthy platform-communication*

This section discusses CyberX management of the secrecy, authenticity, and anonymity of the inter-Cell communications. We present a suitable key management scheme for various connection types in the system. Further, we will illustrate our mechanism to detect maliciously behaving and problematic cells in our system. Additionally, we present our secure authentication mechanism securing the inter-Cell communications against identity theft attacks.

In order to maintain the **secrecy of the sensitive information** stored locally or externally, or being exchanged over communication lines; CyberX uses an asymmetric key encryption scheme to encrypt this data. At the deployment time, the GMS assigns a pair of keys to each cell, a public key and a private key. The public key will be used to encrypt all incoming messages to the Cell. The private key will be used to decrypt these incoming messages. The Cell can use the public key to encrypt the sensitive data within the Cell itself, if the situation necessitates that. For example, it can encrypt sensitive data stored in the local hard drive, or within the memory of the host in locations not controlled by the Cell itself. Figure 2.5, illustrates the architecture of CyberX local security mechanism.

CyberX manages the Cell to Server, and Cell to Replica **data authenticity** using a set of encryption/decryption keys. At the deployment time GMS attaches to the Cell deployment package, the Cell inputs, configuration parameters, the Cell public and private keys, and a pool of public keys for other entities that the Cell might communicate with. The public keys pool will include keys for CyberX servers and routers that the Cell might need to be indirect contact with. Additionally, if the Cell had any replicas at the deployment time, the public keys for those replicas are also included.

At runtime, cells can acquire new replicas as a response to a change in the current recovery mechanism. The process will start by a request from this Cell or the RCTS to GMS to deploy new

replicas. GMS will reply with the public key and the unique Cell name of the new replica to the requester in an encrypted message using the requester public key.

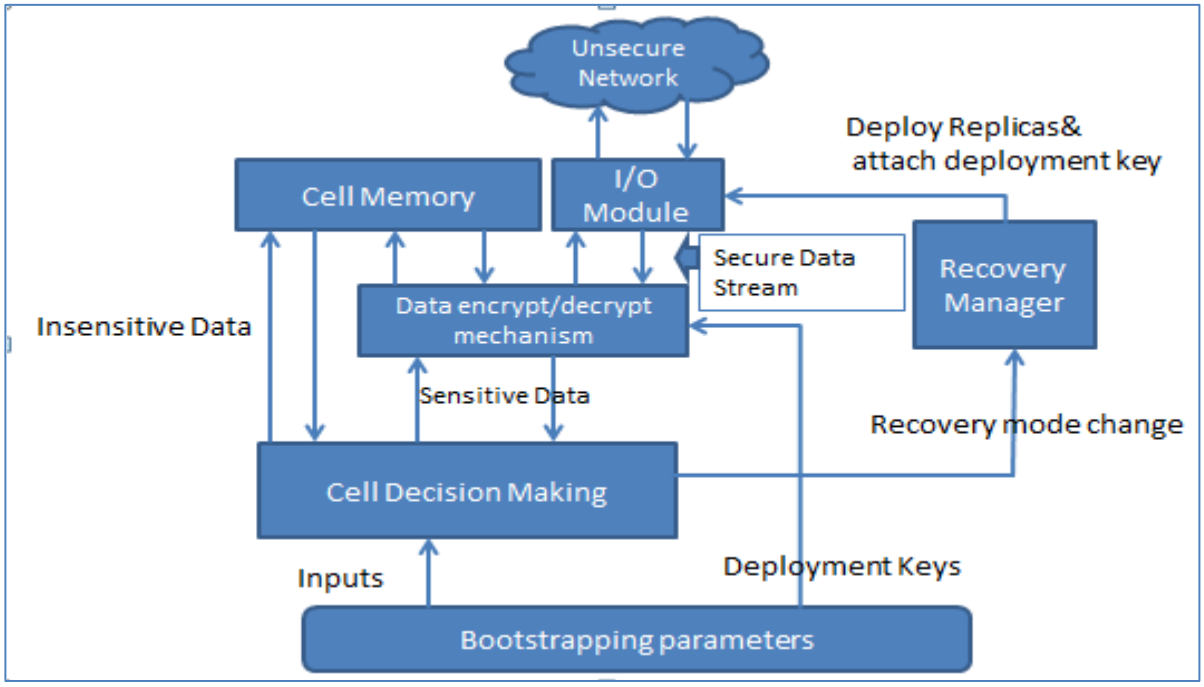


Figure 2.5 CyberX security framework

In order to guaranty the authenticity of all incoming messages, the source id will be enclosed and encrypted with the message. The ARMS will be monitoring inter-Cell behavior with the cooperation of RCTS that keeps track of all the Cells activities. Malicious, or problematic Cells, will be terminated, and their terminated Cell id will be blacklisted and announced to all routing Cells.

In CyberX managed applications**, Inter-Cell communications** can be classified into two main types, administrative related communications, and application related communications. Application related communications are messages being exchanged to serve the application needs and identified by the application designer. The administrative communications are messages like,

recovery beacon messages between Cells and replicas or RCTS; alerts and events between Cells and ARMS; and messages between Cells and routing nodes.

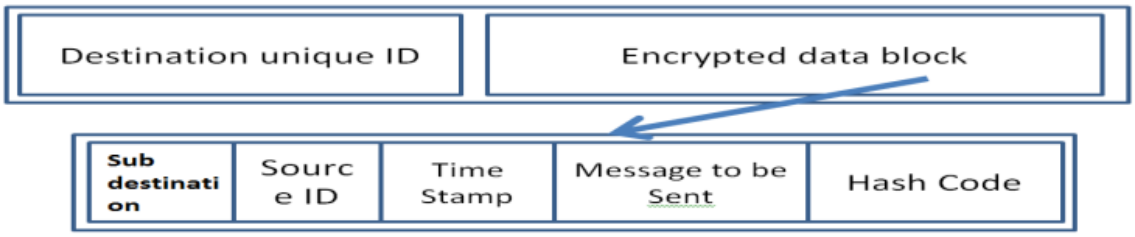


Figure 2.6 The Inter-Cell message format.

Figure 2.6 describes an abstract view for **The Inter-Cell message format.** The message is divided into two main parts, the destination id, encrypted data block.

The encrypted data block is divided into four parts encrypted with different keys, sub destination id, the source id, timestamp, message to be sent, and message integrity assurance data "like hash code".

Inter-Cell communications anonymity, Cells are not allowed to directly exchange messages. The reason behind that is to protect the anonymity of the inter-Cell communications. Cells communicate to intermediate routing nodes to conceal the physical location of the communicating nodes like "replicas, and fractions of the same application", and to control administrative related communications. CyberX uses intelligent routing cells to anonymize the source and destination of any outgoing message. Doing so can block attackers with access to the network from monitoring outgoing messages searching for a certain transmission pattern like "Beacon messages". Identifying these patterns can expose the physical location, and the functionality of the destination cells "replica".

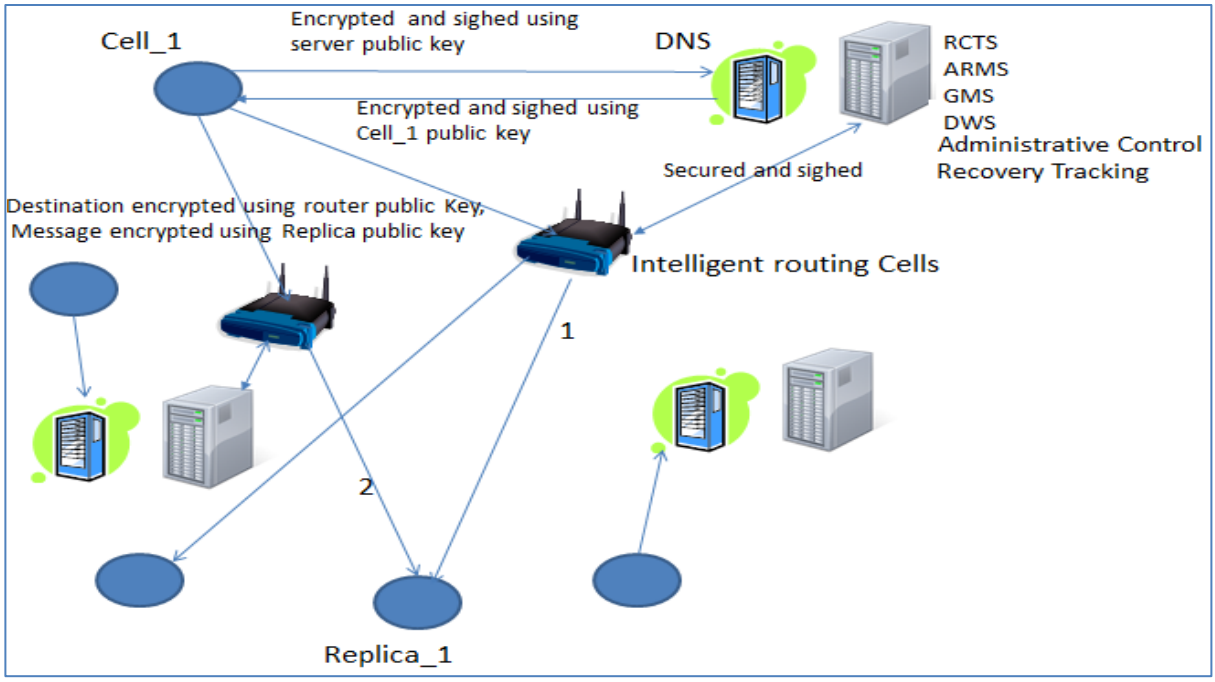


Figure 2.7 CyberX secure messaging system

Figure 2.7 illustrates a communication scenario between different nodes in the system, cells, replicas, servers. Each node uses the destination public key to encrypt and sign all outgoing messages. We use random router selection for each message (EX,1,2)

Cells are only permitted to directly communicate with routing nodes, and servers. Application and administrative related communications involving Cell to Cell messages, has to go through an intermediate routing node. The routing nodes will receive these messages and forward them to their designated destinations in order to hide their physical location. The source Cell will use two different keys to send a message. First, a router public key to encrypt the source ID, and the sub destination ID part of the message. The sub destination ID is the final destination "targeted cell" that the message is indented to be transmitted to. Figure 2.8 is an example of an incoming message to the router from one of the Cells. Second the final destination key, which will be used to encrypt the message and the integrity check fields.

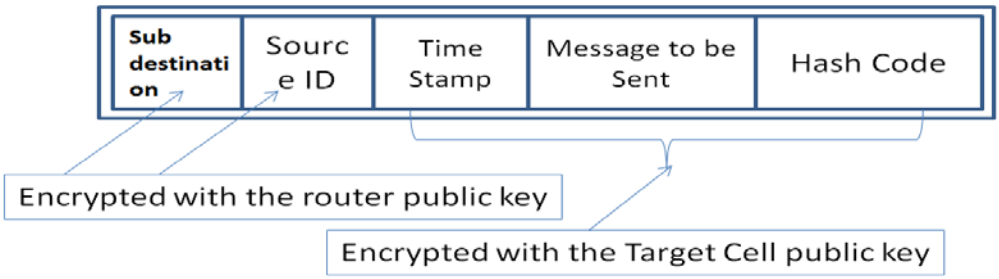


Figure 2.8 Incoming router message.

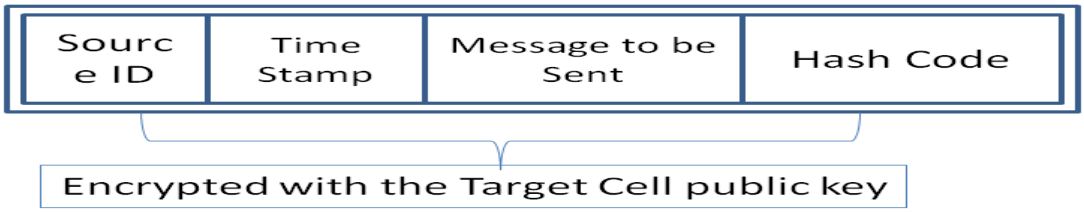


Figure 2.9 Router outgoing message.

The destination ID will be the ID of one of the routers that are close to the Cell. Figure 2.9 is an outgoing message from the router to one of the Cells. The list of close by routers is preloaded to the cell at the deployment time, and updated when needed.

At each routing node the incoming messages will be decrypted using the router private key to extract the source and sub destination information. If the source was blacklisted, the message will be discarded. If the source was not blacklisted, the source ID will be re-encrypted with the destination public key, and attached to the reaming part of the message into a new message to be forwarded to the targeted cell.

We prefer using pre-deployed keys instead of asking for public keys prior communication to block any attempts of a Man in the Middle attack.

2.4 CyberX enabling the CARD concept

As mentioned in Chapter 1, and illustrated in Figure 2.10, CyPhyCARD is designed to realize Trace- resistance, Resilience, and Allied defense needed to support CPS security, by integrating, utilizing, and employing Elasticity, Diversity, Awareness, Cooperation, and Intelligence techniques through a biologically-oriented architecture and methodology. CyberX plays a major role in enabling these five main techniques and in facilitating the employment of such techniques towards the realization of CyPhyCARD design objectives. In this section, we will illustrate how CyberX participates in the realization and employment of such techniques.

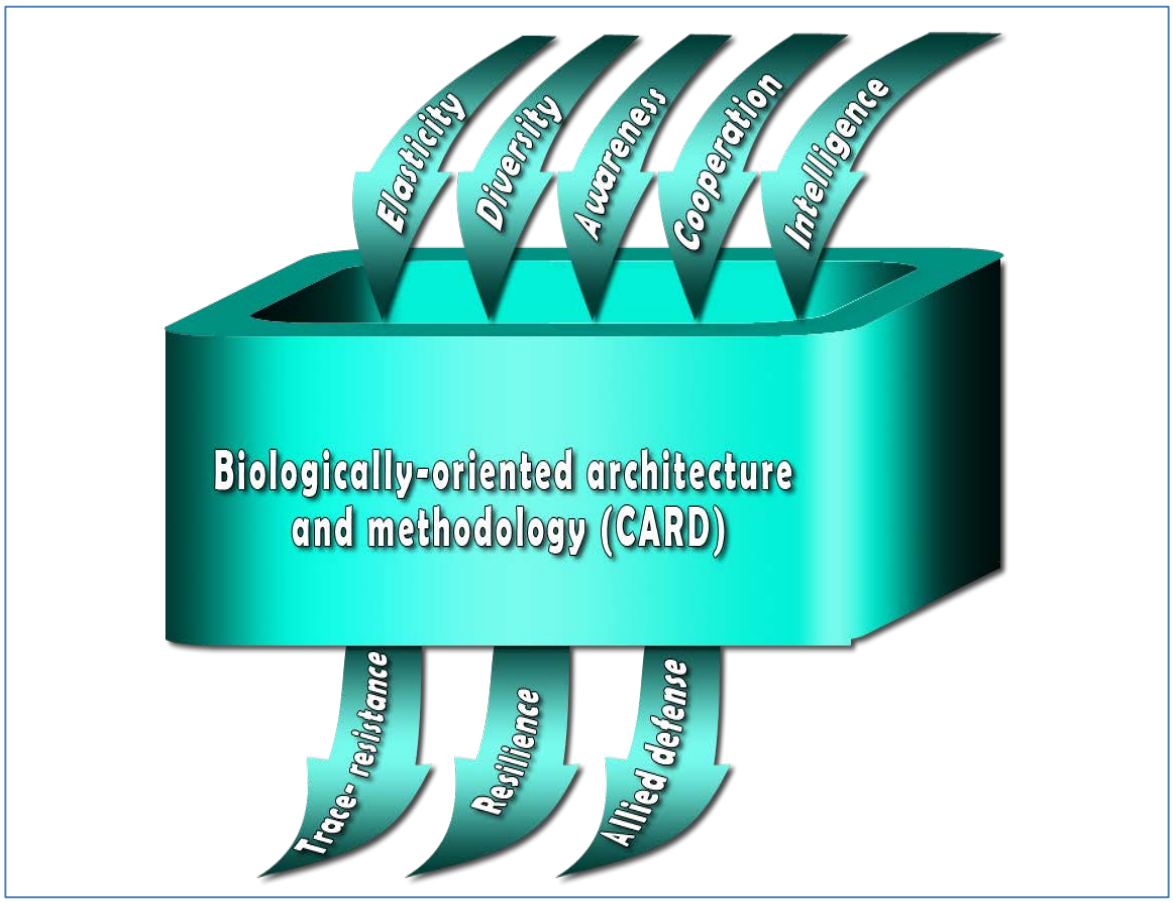


Figure 2.10 CyPhyCARD Conceptual View

2.4.1 *Intelligence*

There are multiple smart processors working within CyberX framework, these smart processors control the decision making process of many of CyberX management platform components, the Cell itself, and the communication framework. The ARMS for example, takes smart decision judging the working Cells performance to allocate any problematic, or maliciously behaving Cells. The GMS unit uses multiple smart processors to manage the Cell network. GMS take serious and critical decisions all the time, for example what is the best location to deploy or migrate to the Cells, which Cells needs to manipulate its current active quality-attribute-objective, and many more similar decisions . Even at the Cell level, the Cell takes many decisions that relay on smart control unit to guide. When to shuffle the current variant, do I need to request migration, and what is the best recovery mode for my current state are good examples for Cell based runtime decisions. The next subsection illustrates the details about such smart processors.

2.4.1.1 CyberX Intelligent smart processors

CyberX is designed of completely loosely coupled components at all levels. The components responsible of decision making through all CyberX framework is called smart processors. Due to the unified feedback uniformity within CyberX framework the smart processor is not bounded to a specific logic or architecture. The feedback within CyberX framework is in the form of score sheets, the same one that EvoSense uses to collect feedback from the ToD hosts as described later in Chapter 4.

CyberX smart processor will process the feedback in the form of score sheet and deliver recommendations or even commands to the execution units. The logic that the smart processor follows can include any static or dynamic decision making logic. CyberX can use and alternate different AI techniques to define such logic. The loosely coupled construction of the CyberX

components at all levels makes it easy for CyberX to alternate different code logic for the smart processor core at different locations.

For simplicity and for the purpose of illustrating the functionality of CyberX, we will present a smart processor architecture working on simple expert system based AI logic. The same representation will be valid regardless of the type of logic being used within CyberX smart processor.

Figure 2.11 illustrates the architecture of a typical smart processor. The processor gets three input parameters. The first input is an incident or consultation id, this id identifies the suitable logic within the rule/logic reservoir to process the incoming input. This id is automatically assigned by the requester based on the type of the guidance requested. For example, if the smart process is needed to evaluate the Cell performance with respect to memory usage, then the id of that specific event is send in the first field. The system will select the expert system logic or set or rules addressing this specific request.

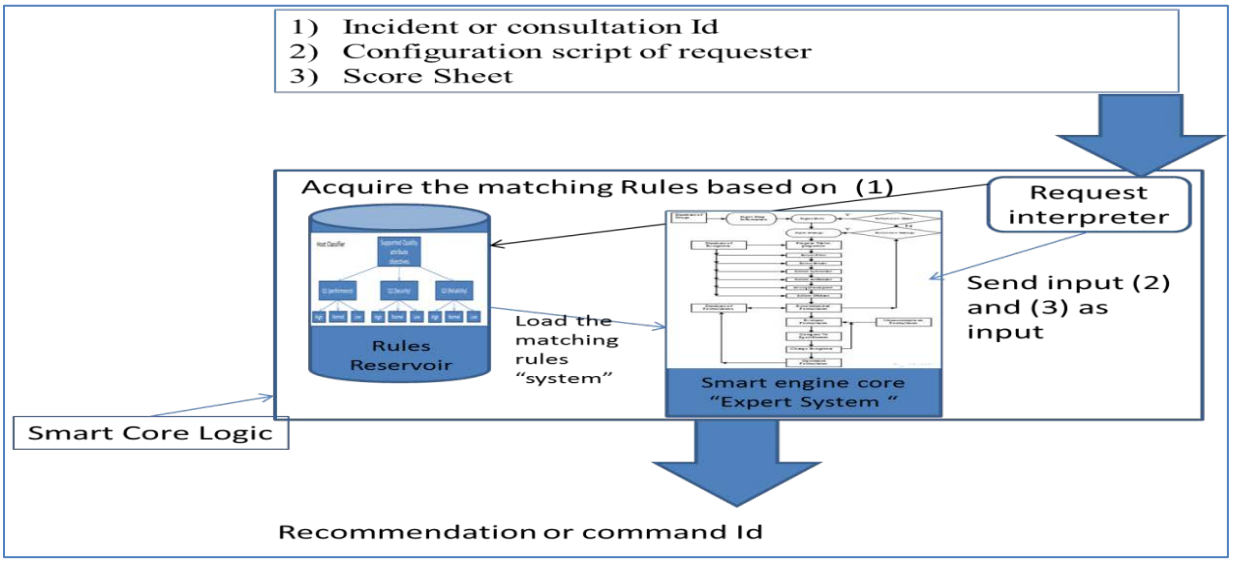


Figure 2.11 The architecture of a typical smart processor

The second field is the score sheet containing the senor feedback that will be used for evaluation. The last input is the configuration script of the element under investigation. For example, if we need to evaluate the performance of a Cell with respect to memory usage, then the Cell configuration script indicating the expected memory usage of the application executing in the Cell is attached as input three.

The output from the smart processor is an id for specific command or a set of commands that the requester should follow based on the given inputs. The output can be a direct command, or a request for further investigation. Figure 2.12, shows a flowchart representing one of the expert system rules defining the smart processor logic being used to evaluate a memory misuse in one of the Cells.

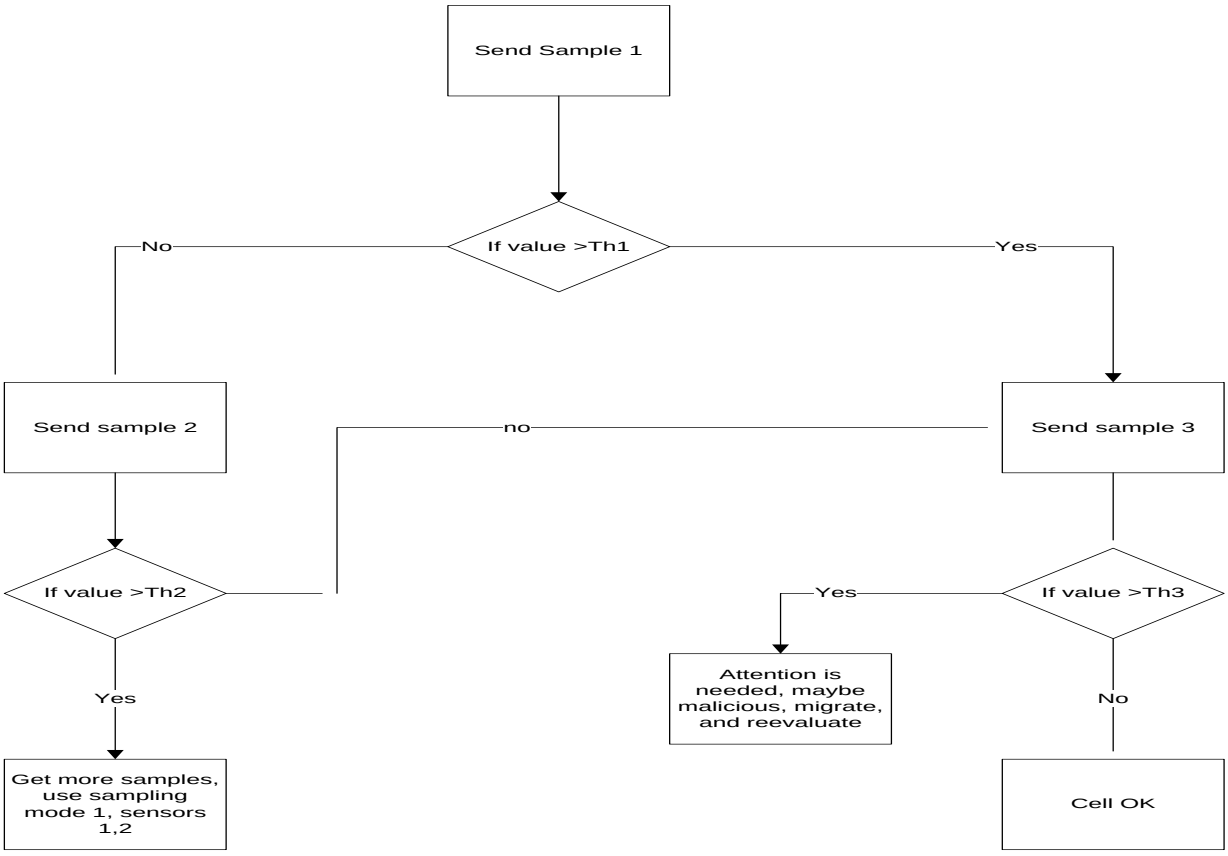


Figure 2.12 One of the smart-processor expert-system rules.

The logic presented in Figure 2.12, represents a case where this logic was selected based on the request id and acquired from the rule reservoir to be loaded to the expert system engine. The process involves comparing certain fields in the score sheet and the script to evaluate the memory performance and whether the Cell has to be moved from its current location or not.

The use of rule based expert system as the core logic of the smart processors is efficient and accurate but might not be so smart. As mentioned before, CyberX can use any smart logic as the core of its smart processor. We used this type of simple AI based solution for illustration purpose only.

2.4.2 *Situation awareness framework*

One of the main objectives of CyberX is its enable runtime application adaptation to the surroundings changes. In order to enable such adaptation, CyberX has to be fully aware of the surroundings of all active Cells. CyberX situational awareness can be categorized under three main levels; the first is the local situational awareness at the Cell level. By Local awareness we refer to the Cell being aware of the application needs and requirements all the time. CyberX maintain such level of awareness by enabling application and infrastructure message exchange at runtime through a dedicated communication channel and language. The executing variants can send messages to the host Cell informing it by its current state or request a certain change.

The Second level is the environment awareness at the Cell level. That level refers to the Cell being aware of what is happening around it. CyberX maintain such level of awareness at two scopes. The first is a local scope at the host level, and the second one is a global scope at the neighborhood or the network level. The first scope is maintained using a set of host resident sensors deployed as a part of the CCDNA on the host. These sensors monitor certain aspects on

the host and report its feedback to the CCDNA. The CCDNA group the feedback and post it to the ARMS and to the hosted Cells. The Cell uses this feedback as a source to guide the dynamic adaptation process.

The third level is a global situational awareness at the network level. ARMS unit collects the feedback from the CCDNAs, the feedback from the intelligent routing nodes, and the feedback from the local monitoring units monitoring the activities of all running Cells. The feedback is grouped and meaningful conclusions are extracted from it. Such conclusions are sent to the GMS to guide its decisions, and commands to the running Cells. The GMS can send further commands, or even awareness messages to the Cells based on such conclusions. Figure 2.13 presents the decision logic within the ARMS unit.

Messages exchange and inspection protocol

In order to support large scale Cell networks, CyberX is designed to scale in a hierarchal fashion. Figure 2.15 illustrates the hierarchal management of CyberX. The leaf nodes are in direct contact to manage a specific set of Cells. The management units “CyberX(s)” report to each other in hierarchal fashion to update the global situational awareness of the whole system.

The feedback in CyberX framework takes one form, a score-sheet like report. At the host level, the Score-sheet represents a report that compares the behavior deviation regarding the sensing target to a predetermined threshold. CCDNA Sensors are classified into different sets representing their targeted sensing objectives " ex, memory, communications, privacy, ..etc.". Thresholds are dynamically adjusted based on the nature of each host, and the number of false negatives/positives reported by the Sensor.

Score sheets from different sensors for the same host are sent to the ARMS of the leaf node to compose comprehensive score-sheet and to report malicious events to the GMS of the leaf node. Figure 2.13 illustrates the process within the leaf ARMS.

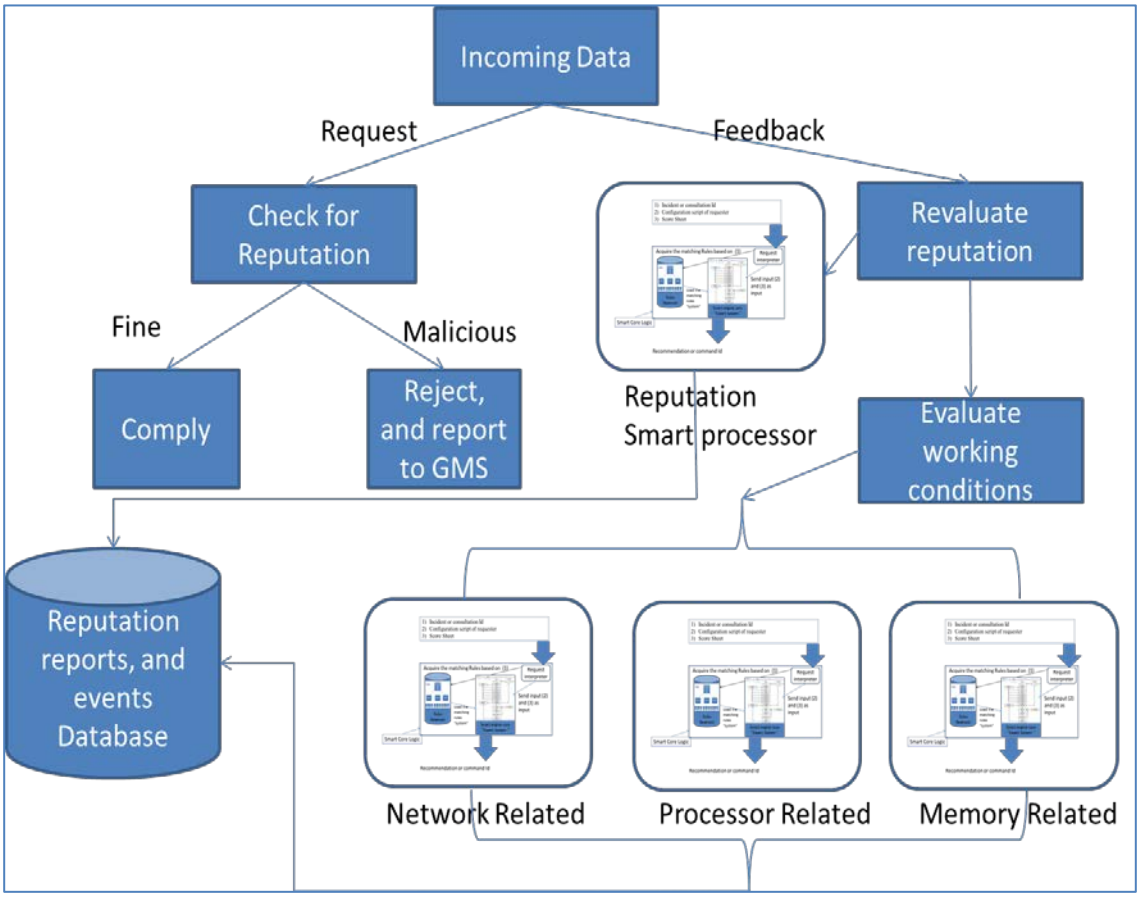


Figure 2.13 The decision logic within the ARMS unit.

The incoming input to the ARMS is either requests or feedback in score sheet format from specific source. The incoming feedback is sent to a set of smart processors to evaluate different aspects related to the source reputation, and performance. The output is stored in the local database for future reference, and the source is reevaluated. If the evaluation indicated maliciousness, the source is reported to the GMS for further actions. If the incoming input was a request or an inquiry about specific object, the database is checked and the source is evaluated. If

the evaluation indicated maliciousness, the source is reported to the GMS for further actions. If not the request is accepted. Figure 2.14 represents the ARMS reporting mechanism.

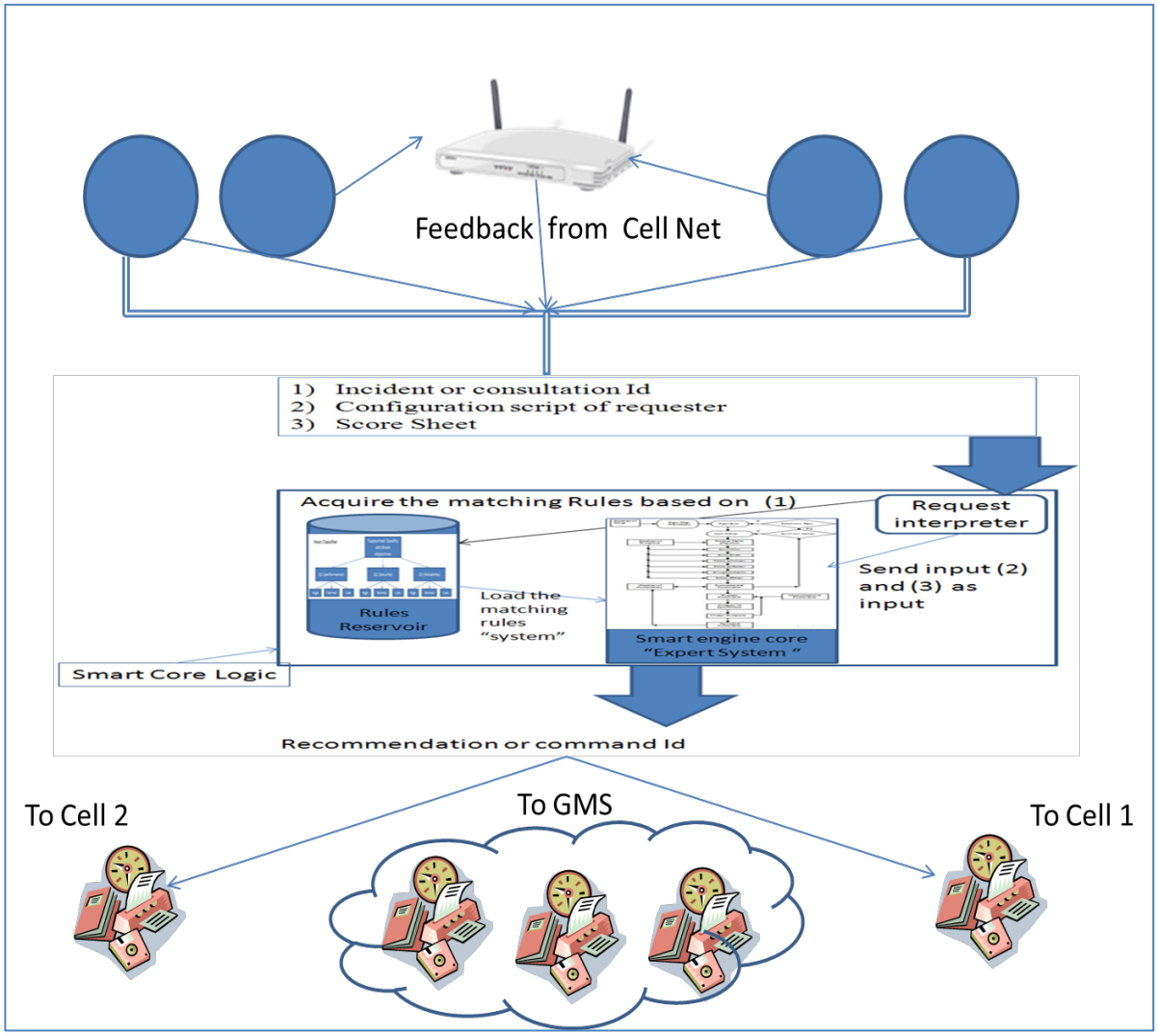


Figure 2.14 The ARMS reporting mechanism

Each smart processor has set of rules with specific thresholds in a score sheet like format. Rule-sheets have values for different objects "ex, memory, communication,.etc" reflecting the behavior patterns "ex: attack signature, resource starvation signature, platform instability signature, …etc" of each object in case of up normal behavior . Behavior pattern description can be discreet or continuous. Rule description also includes the host sampling procedure. Sampling

procedure describes the needed number of samples per object and the duration of each sample, and the sensors needed to takes such samples. The GMS can add new sensors to the CCDNA's or deploy temporarily sensors when needed.

The feedback is sent to the leaf GMS. Based on the feedback, up normal behaviors deviation might be detected, and the adjustment of the Cell activity based on such deviation as described in the rule will be followed by the source Cell. The GMS at the leaf node will send its commands to the active Cells within its jurisdiction based on that adjustment. The commands might ask the Cell to shuffle to target different quality attribute objective, or ask the Cell to migrate from this host to another identifying the destinations, or even ask the Cell to sop; slow; or stop shuffling.

GMS of leaf nodes send grouped and classified score sheet reports to the parent node to expand its awareness of the underlying Cell network. These units process the feedback and send guidelines to the child ARMS nodes as an adjustment of the list of rules, and relevant thresholds.

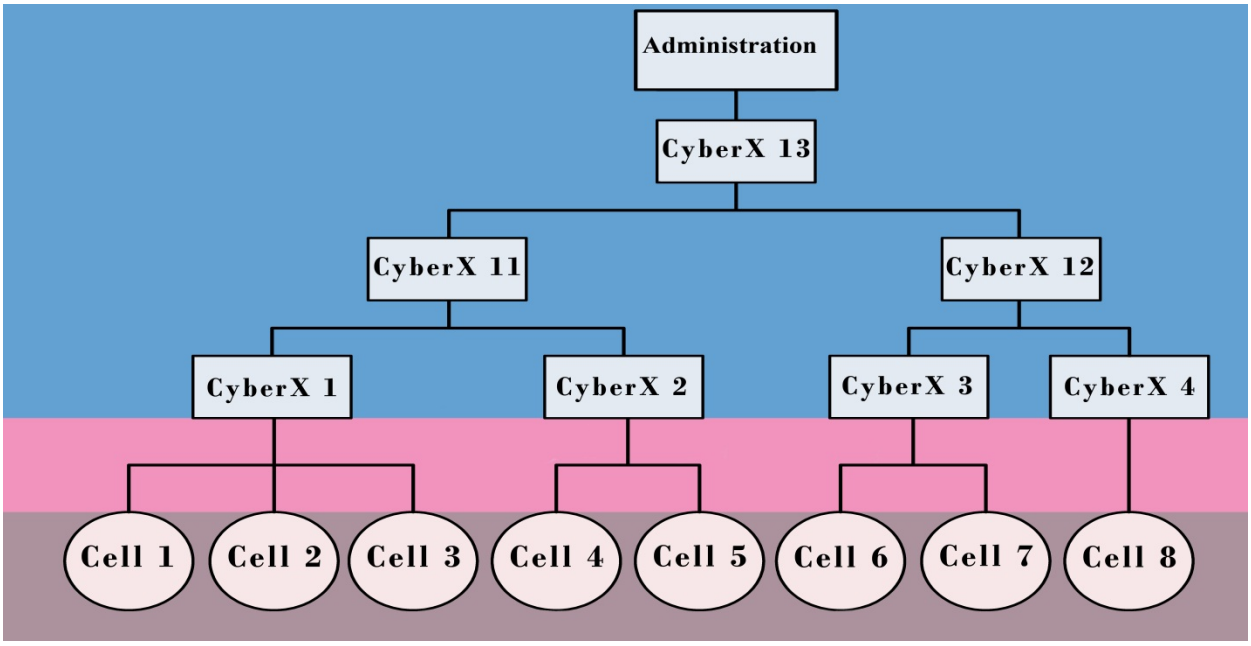


Figure 2.15 The management hierarchy

2.4.3 *The Cooperation framework*

At the application level, CyberX Cells are designed to work in a divide-and-concur fashion to increase the chance for survival for the whole application in case of partial failure. Cells are capable of exchanging messages and coordinate certain tasks between multiple Cells to achieve certain application objectives. Enabling such cooperation enhance the application resilience against failure, enable application level resource sharing, and enhance the application performance by enabling applications to distribute their workload over many Cells. Further, enabling Cell mobilization among hosts can have a good impact on the application performance as the cell can move to facilitate service delivery to consumers by moving within geographical proximity from them. Doing so, enables a single application to distribute its tasks in multiple geographical locations and seamlessly move between these locations when needed.

At the host level, CyberX Cells are designed to separate the main design concerns data, logic, and physical resources. Enabling such separation enable Cells to easily move between hosts regardless of the host configuration. CyberX gives the host a chance to share its resource among multiple long lived applications. In that case different hosts cooperates together to serve the needs of one application. CyberX can move Cells between hosts if the host configuration at certain times was not acting to the best interest of the application, like it has no resources, security levels are low, other applications working on the same host can induce conflicts, ..etc.

At the system "Cell network level", CyberX manage vast number of Cells hosted in many hosts distributed in many unrelated geographical locations. The design of the hierarchal management platform allows the management units to transfer workload between them by moving Cells between hosts to maximize the applications recourse utilization and the quality-attribute-objectives satisfaction. Further, the collected information about the host from the Cell

monitoring units can guide the management of the host network to be more aware of the host. For the diffusion management as described in Chapter 3, hosts share such information to guide the diffusion management process to the best interest of the application. With CyberX, hosts are cooperating even if they don't know about that. The next subsection gives more details about the Organism level resource sharing.

2.4.3.1 Organism level resource sharing

Due to the loosely coupled construction on CyberX application execution framework, a multi task application is presented as a lager organism composed of multiple Cells. In conventional architecture, automated resource sharing at the application level is not possible unless in one case where the application is designed and customized to support that. In CyberX the application is totally isolated from the underlying physical resources. Doing so enables CyberX to trick the application by creating a virtual physical resource layer that is actually hosted among multiple hosts. CyberX enable applications with no inherent support for resource sharing to share the physical resources of multiple hosts.

A large multithreaded application is usually played by one large organism of multiple Cells. Such organism can distribute its Cells among multiple hosts seamlessly and without any support from the application. There is no special support needed from the application if the entire organism was hosted on one or more Cells. Further, CyberX situational aware management platform can change the organism, Cell, host distribution seamlessly at runtime for any reason. For example, if the running Cell was starving for more resources in its current host, CyberX can seamlessly move this Cell to another host without any involvement from the application.

The process starts at the time of deployment, as mentioned in section 2.4, a COA ready application comes in a package of components designed based on certain aspects. The package

includes a set of variants and a configuration script indicating the configuration aspects, and parameters of each variant, and the general aspects, requirements, …etc of the entire application.

When the host joins CyberX network is classified in one of 3 categories “normal, high, low” under each targeted quality attribute objective and based on the available resources. Figure 2.16 represents the host classification process at time of attachment.

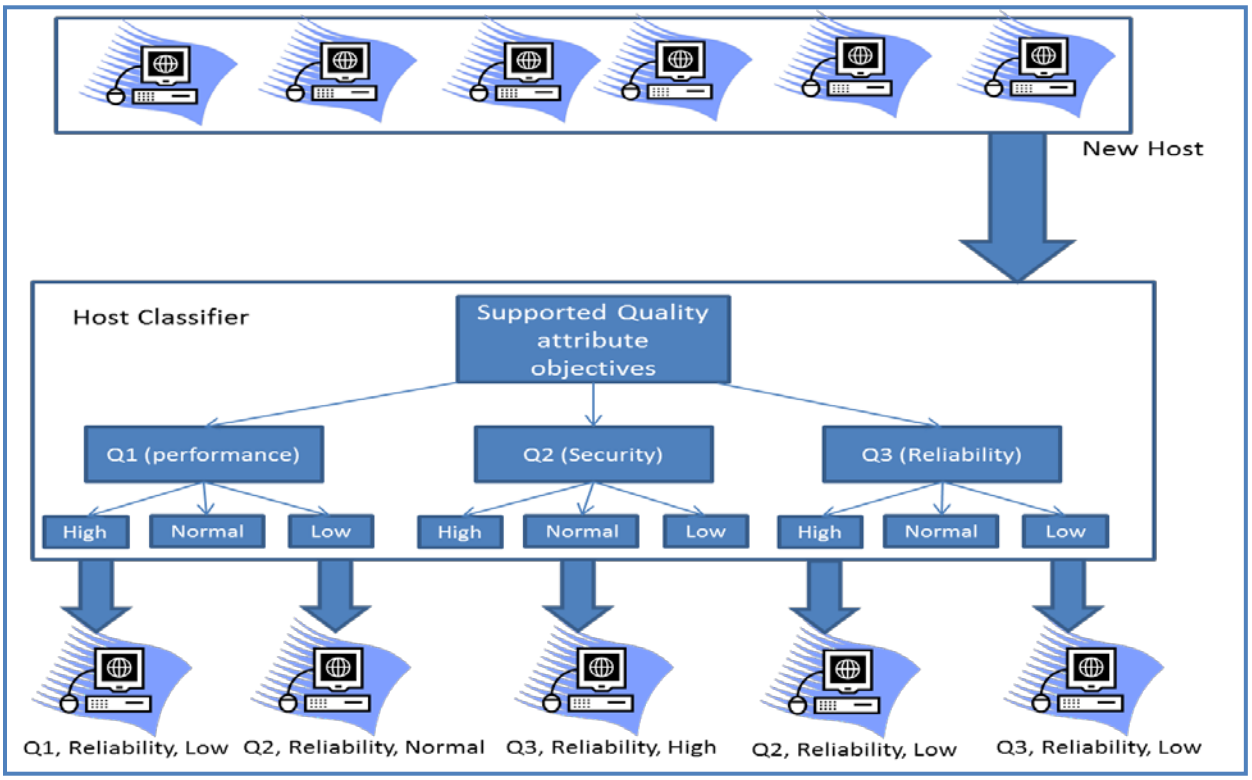


Figure 2.16 The host classification process at time of attachment

At the deployment time CyberX GMS process the application global configuration scrip against a set of rules. Based on that comparison, the application will be classified under the available classes based on the supports quality attributed by the system.

The application can be hosted in multiple hosts to satisfy the requested quality attribute objectives mentioned in the application configuration script. The GMS will select the next host

available that fits into the same classification and with available resources that can satisfy the application needs.

Based on the GMS selection of hosts, the application tasks identified in the packaged code variants will be encapsulated into set of Cells and the Deployment process begins. Each Cell will have its part of the requirements based on the task hosted. The Cell local monitoring unit will notify the ARMS if any of the needed requirements where not doable. Upon such notification, a report from the ARMS will be sent to the GMS indicating the problematic Cell, and the performance and trustworthiness evaluation of such Cell, and all the available details about the reported problem. The GMS will respond by selecting an alternative host capable of providing the needed requirement and migrate the Cell to it as described in section 3.4.

2.4.4 *Elasticity*

CyberX was designed to enable the application, and the host / host network to be elastic in terms of resource usage and availability.

At the application level, Working within one of CyberX Cells gives the application the opportunity to expand or shrink its resource usage without caring whether this change might or might not be possible or what will be the effect of that change on the hosting node. CyberX will handle all this details enabling high level of resource elasticity. The application designer can build the application to consume the resources that it needs as long as it is informing the host Cell for major resource usage patterns. Upon the reception of a resource usage change, the Cell will check with the CCDNA if the host will be able to afford this change. If the host was capable of providing such resources then the Cell will remain in place and the CCDNA will grant these resources to the Cell. If the host were incapable of providing the needed resources, the Cell will

ask for migration from this host to another host clarifying the reason behind request. The GMS will move the Cell to another host and will resume application execution as described in section 2.6.

At the system/host level, CyberX Cells acting as a buffer between the host resources and the application enable the host to change its resource availability and configuration profile at runtime without any worries about possible application failure. CyberX will always adjust the Cell needs based on the available host resource as long as it is possible. One of the techniques that CyberX uses to adjust the resource usage based on the available resources, is to shuffle the active variant to a more resource efficient variant if that change will not conflict with the application requirements. In case that the host is no longer capable of hosting the Cell, CyberX will simply migrate the Cell from this host to another host seamlessly and with minimal operation interruption. The next subsection illustrates the details of Cell migration to support different quality attribute objective, like resource elasticity, diversity, and resilience against failures.

2.4.4.1 Cell migration protocol

CyberX utilize COA intrinsic separation of design concerns to migrate active Cells between hosts in order to balance the workload of the whole network. The migration process also targets other objectives, these objectives and more technical details about the migration process will be illustrated in ChameleonSoft Chapter, Chapter 3. ChameleonSoft uses Cell migration to induce special confusion and diffusion to realize special diversity needed for the moving target defense approach presented in the Chapter. In this section we will briefly describe the technical process of migrating a life Cell between different hosts.

The migration process has two modes depending on the level of resources available at the destination Cell, and the time frame available before terminating the source Cell. These modes are cold and hot migration modes. CyberX always uses the hot migration mode as a default mode, because it provides minimal transition time, and zero execution steps losses.

The process starts by the arrival of a migration request. This request can be issued by three entities, the Cell itself, and the ARMS, or the CCDNA on the host. The CCDNA can request Cell migration if the Cell was requesting too much resources than the available resources with an increase crossing a certain threshold. Crossing such threshold indicates that the Cell is a threat to the other Cells, and either this Cell or the other Cells hosted on the same host might face serious failures if the Cell is not removed from this host. The ARMS issue Cell migration if the Cell was marked dangerous due to the analysis of the feedback collected from the sensors hosted on the CCDNA hosting the Cell. The Cell can ask for migration from the current host to another one if the host was not capable of provisioning the needed resources to support the hosted application within the Cell.

Regardless of the source or the reason behind the migration request, all these requests are sent to the GMS to process and execute. When an authentic migration request comes to the GMS, it comes with a report justifying the reason behind migration. Based on the reason GMS selects an appropriate host for the Cell to migrate to. The logic behind this selection is illustrated in section 3.4.

The Cold migration mode:

In this mode the Cell can be terminated upon the issuance of the migration command, and the GMS replaces the Cell with a new fresh Cell in another host. The new Cell will be initialized with a Cold migration mode status and the last known Check point for the source Cell will be provided

upon initialization. The GMS will get this information from the RCTS. The new Cell will start with the same variant that was running on the source Cell. The variant ID will be a part of the datasheet provided to the Cell at bootstrapping. Upon startup, if the application was in communication with any other Cells, they will be notified that the Cell was migrated, and execution progress synchronization protocol will start.

The process ends by communication redirection at the DNS by changing the DNS record of the Cell to point to the new host.

The Host migration mode:

This is the default migration mode in CyberX as it is more effective and provides a 0% execution steps losses and minimal transition downtime. The GMS starts this process by replicating the source Cell. The source Cell recovery mode is explicitly changed to hot recovery mode. In this mode the Cell is forced to synchronize all its action with a replica Cell. The GMS selects the appropriate host for the replica, starts the replica and informs the source Cell of the replica virtual id. The details of Cell replication is illustrated in section 2.5.

Upon successful synchronization, the source Cell is terminated, and the virtual id of the source Cell will point to the replica and the routing nodes will be informed by that change. The replica will be resurrected to live mode, and the original recovery mode that the source cell was using before migration will be restored.

The main advantage of this mode is its ability to keep the source Cell running until the new Cell takes over. The estimated transition downtime for this process is the time needed to update the DNS record for the Cell virtual id with the real physical host id of the replica, which is a very small time, and it can be negligible leaving us with a zero transition downtime. Figure 2.17 illustrates the two different Cell Migration modes.

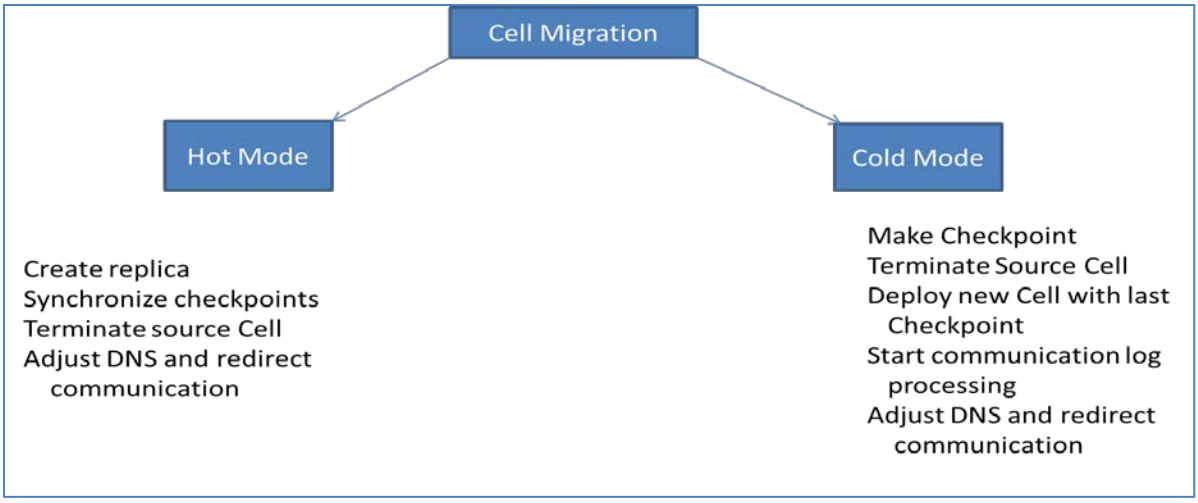


Figure 2.17 COA Cell migration process

2.4.5 *Diversity*

CyberX employs diversity to enable dynamic adaptation to surroundings changes as illustrated in section 2.4.5. A more complicated multidimensional employment of diversity for software behavior encryption and moving target defense is presented in Chapter 3. Chapter 3 illustrates the details of using CyberX separation of design concerns and the loosely coupled infrastructure to enable multidimensional diversity employment by ChameleonSoft. At the application level ChameleonSoft employ temporal diversity by shuffling a set of similar function different behavior variants inducing enough confusions and diffusions to encrypt the execution behavior of the running software. Further, at the system level, ChameleonSoft uses CyberX management platform, to move the running Cells between heterogeneous platforms to increase the complexity of the encryption process. We will not go further into the details of this process in this chapter as it is illustrated in details in the Chapter 3.

2.5 The CyberX managed multi-mode failure recovery

CyberX applies diversity techniques to enable autonomous adaptation and performance optimization. Applying diversity might involve multiple interruptions of the executing variants. Doing so might lead to multiple coincident failures. Therefore, CyberX is designed to equip COA based applications with an autonomous, dynamic, and situational-aware multi-mode failure recovery mechanism to resolve possible coincident failures. A major outcome of this recovery mechanism is the failure resilience enhancement not only against coincidental failures, but also against malicious induced failures by adversaries.

CyberX dynamically and autonomously changes the Cell recovery-policy to switch between different fault-tolerance granularity levels. Such levels might target reliability, survivability, and resource usage optimization. For fine-grained recovery “Hot-recovery” against logical failures, the Cell can have one or more replicas on the same physical host. Further, for a finer-grained recovery against logical or physical node failure, the Cell might have one or more replicas on different physical hosts. The fine grained recovery comes in two modes, the resource saver, and the fast-recovery modes.

In the resource-saver mode, replicas need to only replicate the STM, I/O unit and local data store units of the Cell. The remaining Cell components stay in hibernation waiting for resurrection when the replica takes over. These replicas will have one variant all the time and no shuffling or recovery policy change until resurrection. We do that to minimize the resource usage by these replicas. This mode do save the resources but on the account of increasing failure downtime by the time needed to resurrect the Cell.

The fast-recovery mode can achieve virtually no task-transition downtime by using a fully-alive replica Cells. Replicas mimic all the actions of the source Cell except outgoing communications and data change. The execution-transition in this case is a simple network rerouting by a DNS record update. The failure downtime is only the time needed to detect failure. The only disadvantage of this mode is the resource duplication needed to keep both Cells alive.

In a resource-constrained environment, CyberX can follow a more coarse-grained recovery "cold-recovery" that might save some of the resources used by replicas while compromising some of the execution states, and increasing the failure downtime.

The default Cell design forces COA Cells to send a periodic beacon messages to the RCTS containing the last executed Checkpoint, some sensitive data, and the currently executing variant to be saved on the secure remote data-store. In case of failure, the RCTS notice the delay in beacon message arrival, and investigates the possibility of failure. If failure was detected then the last recovery procedure will be executed as follows:

In case of a failed Cell that follows a **fine-grained recovery mode** then the RCTS will inform MGS to send a resurrection signal to the replica and notify the routers, and start deploying a new replica to replicate the resurrected one. After successful restoration, DNS entry will be adjusted.

If the Cell was following a **coarse-grained recovery mode** then the management will deploy a replacement of the failed Cell and the last checkpoint received by the RCTS is attached to the deployment package. After successful restoration, DNS entry will be adjusted, and the Cell will start execution as a recovered-Cell mode. This mode involves negotiating with all Cells in communication to resynchronize any lost execution steps.

The coarse-grained recovery mode is always-on by default enabling the support of multiple concurrent recovery policies. The remote safe store is updated regularly with beacon messages

from all working Cells. Each Cell will independently and dynamically set its own message update frequency. Such update frequency could be influenced by the change of the current recovery policy. The update frequency might decrease in fine-grained recovery mode; while they should increase with lower granularity recovery.

CyberX can dynamically change the cell recovery policy at runtime. The change is guided by the application requirements and host conditions. In a stable situation with non-mission critical application, a coarse-grained recovery policy can be used, while in a more hazardous situation, a fine-grained recovery is preferred. The cell utilizes the available information about the current working environment with the application profile to decide the appropriate recovery policy to use. As the surroundings change, the cell changes the current recovery policy to suit these changes.

2.6 **A CyberX-managed application**

The COA-Cell can be built in different techniques based on the targeted resource virtualization depth. We implemented the simple and fast version of the Cell to enable quick development of a prototype. We are in the process of realizing a more complex version of the Cell utilizing one of the application virtualization techniques mentioned in [4].

The main differences between these two versions are: The Slow and complex version of the Cell is a computationally heavier Cell, with a thin and uniform hardware virtualization layer. Variants are built to target a uniform virtualized platform. The main advantage behind enabling such uniform application design are: Reducing the cost of software production, management, and maintainability, widening the scope of special shuffling in order to increase the system security and reliability, and reducing the effort involved in system upgrades and/or changes. The main

disadvantages are the added workload, and higher risk of failure when compared to the simple version.

CyberX migrate Cells between hosts for recovery purposes, and for Moving target defense as presented in Chapter 3. The main difference between CyberX Cell migration and virtual machine migration is that virtual machine migration is a computationally heavy process, and it needs complex modifications to enable the kind of real-time migration and diversity employment-dimensionality provisioned by CyberX. Working with virtual machine concepts as known in the literature is not feasible, because of the cost of diversity employment, communication bandwidth, and cost of failure recovery for such huge capsules.

CyberX uses fine-grained application development and single task capsules with a total separation between the main design concerns, Data, Logic, and Physical resources. Such separation facilitates runtime shuffling with minimal computational, and communication cost. In addition, CyberX handle failure recovery intrinsically and with a minimal resource usage, and downtime. The cost reduction are mainly the outcome of the COA fine-grained application design, the utilization of lightweight capsules, high level of automation, intrinsic consideration of failure recovery, and the separation between the data and the mobile capsule itself. In Table 2.2 we present a detailed comparison between two of the virtual machine techniques, and the COA Cell illustrating the main aspects regarding composition, construction, and diversity application methodology and cost differences.

	System Virtual machine	Process Virtual machine (Application	Fast COA Cell	Slow COA Cell

		virtualization)		
The definition of the Sandbox used	completely isolated guest operating system installation within a normal host operating system	Unified platform-independent programming environment that abstracts away details of the underlying hardware or operating system enabling the execution of a pre-encapsulated single process that runs as a normal application inside a host OS.	Partially isolated, CCDNA monitored and controlled program execution enabling the execution of a single Chameleonized application that runs as a normal application inside a host OS.	Completely isolated, CCDNA monitored and controlled program execution enabling the execution of a single Chameleonized application that runs as a normal application inside a host OS.
Sandbox tasks	Mainly hardware abstraction and virtualization layer	Limited hardware abstraction and virtualization layer	hardware abstraction and managed direct access	hardware abstraction and partial hardware virtualization
Sandbox size	Full OS with multiple applications	Single application	Single application	Single application
Utilization for Diversity	Migration	Migration “ can be used to enable	Migration // Internal shuffling	Migration // Internal

		CBE internal shuffling”		shuffling
Resource usage	Huge	Less than System Virtual machine	Very limited	Limited
Complexity	Very complex	complex	Very Simple	Simple
Composability	None or explicitly	None or explicitly	Intrinsically	Intrinsically
Type application	Normal application with suitable OS within the sandbox	Normal application encapsulated in a container than needs to be rebuilt each time the platform change	CyberX application with a dedicated variant for each targeted platform. Simple variant change when platform change.	CyberX application with a uniform targeted platform. Seamless platform change.
Virtualization layer	Complex hypervisor	Hardware abstraction layer	Simple CCDNA	CCDNA
Application / virtualization awareness	Applications unaware of virtualization	Applications unaware of virtualization	Application aware virtualization	Application aware virtualization
Separation of design concerns	Only physical resource isolation	Only physical resource isolation	Data, logic, and physical resources (multiple implementations for different hardware platforms)	Data, logic, and physical resources

Sandbox instance creation, and deployment	Manual/ application specific automated	Manual/ application specific automated	Automated	Automated
Sandbox intelligence	Limited or none	None	Intelligent	Intelligent
Elasticity	Static	Static	Dynamic	Dynamic

Table 2.2 Comparisons between CyberX Cell virtualization and conventional system virtualization

2.6.1 *The simple and fast version of the Cell*

In the simple and fast version of the Cell, variants always match the targeted deployment platform. In this mode variants have a controlled direct access to the actual host hardware. The Cell instantiates, monitors, and controls all the runtime aspects of the variant as descried latter. All communications and data access are only permitted through the dedicated units/channels within the Cell. No hardware virtualization is needed. The main advantage of this approach is its simplicity, and lightweight with respect to the amount of consumed host resources to enable virtualization. In order to enable emergency Cell-relocation, the variant pool should contain variants matching all the targeted platforms.

As mentioned before a COA-ready program is a program that enables check-pointing with at least two different objective variants enabling quality-attribute manipulation. The checkpoint reporting location has to consider data integrity requirements especially in case of failure. All data has to be committed before checkpoints.

At the deployment time, a new DNS record will be created by the GMS for each Cell indicating the application virtual name to be used for inter-variant communications “if needed by the application designer”, the Cell unique id for inter-Cell communication, and the IP of the physical-host hosting the Cell.

The deployment starts by the CCDNA receiving the deployment package from the GMS including the Cell globally unique ID(s), the initial checkpoint value, variant pool setup “variant binaries, names; numbers; sets; variant-classification” , the configuration script describing the specs of each variant, the global objective of the application, and any specific specs added by the developer to be considered at time of execution “number of application fractions; fraction-names; ..”, the initial shuffling and recovery policy, the needed security level, and the list of security parameters and encryption keys.

The CCDNA starts the Cell by constructing the components mentioned in section 2 with the provided unique id. Then the CCDNA starts to interpret the deployment configuration file in order to generate separate configuration files for each Cell unit describing any modification in its default task assignment, or special considerations to be taken care off at the time of execution.

The execution starts when the execution unit asks the STM for the starting checkpoint, the STM will get this information as a part of the deployment configuration file. STM will repeatedly provide this information to the execution unit at each shuffling event. The execution unit starts to launch the first variant while passing the appropriate bootstrapping parameters.

The last executed checkpoint value will be held by the STM locally, and remotely at the RCTS that will receive it via the Cell beacon messages.

At runtime, variants will update the STM frequently with the checkpoint advance and any other special needs via a dedicated communication channel.

At the time of shuffling, the Cell diversity manager gives the shuffling signal to the STM and the execution unit, which will start the process after the next reported Checkpoint and based on the provided shuffling orientation as follows;

Quality attribute manipulation: let us take an example, an attacker might be able to induce a change in the system surroundings, like a DOS attack to overload the network to force the system to shuffle the currently executing variant. The CyberX will ask Cells close to the induced event to change their variant to target a different quality attribute (e.g. performance) that suits the induced change in the environment. We have two main realization modes for the shuffling operation the **greedy** and the **light** modes. The system designer can select either one of them based on the available deployment-platform host resources, and the criticality of the application. **The greedy-mode** with seamless handover offers virtually no-downtime but duplicates the resource usage at the time of shuffling, and the **lightweight-mode** offers no-resource increase at the time of shuffling on the account of increasing the transition time by the time needed for variant loading and synchronization. We will briefly describe both.

The greedy-mode "local replication": Upon reception of the shuffling signal, the execution unit starts to load the new variant in freeze "ideal" mode. The new variant will connect to the STM that will locally synchronize the execution checkpoint with it. The communications unit will duplicate all the inputs to the old and the new variant. Upon reception of the ACK Signal from the STM and the communications unit confirming that the synchronization is completed, the execution unit sends pause signal to the old variant, and a resume signal to the new one followed by a termination signal to the old variant.

The lightweight-mode: Upon the reception of the shuffling signal, the execution unit starts by local synchronization with the STM for the checkpoint update. Then it pause the old variant, and

informs the STM and communication unit about the execution hold. The communication unit will buffer incoming messages for the duration of the handover. The execution unit will terminate the variant, and starts loading the new variant with the last known checkpoint, and informs the communication unit and the STM about the successful loading to resume execution. The communications unit will send any buffered messages to the new variant.

2.7 CyberX role in mitigating the BlackWidow attack

In this section we intend to discuss the ability of CyberX to invalidate the attacker assumptions on the case study “blackwidow attack scenario” presented in Chapter 1. We list part of the assumptions listed in Chapter 1. We focus only on the assumptions that CyberX participates in disputing. The rest of the assumptions are disputed by the reaming contributions of CyPhyCARD, EvoSense or ChameleonSoft.

Attacker assumptions:

- The defense system shares the same network or host with the target of attack/defense system.
- The system is not capable of being fully situation aware of all its components in a massive-scale network in real time.
- The defense system management workstations (that the administrators use) share the same network with the target of defense.
- It is not feasible to monitor all the host behavior patterns while sharing the same workstation that is performing user tasks.
- Defense systems are not resilient against attacks, and have weak recovery mechanisms.

[Note: most of them assume that they will not be the target of an attack as long as they were able to secure their ToD. Additionally, usually they have no intrinsic failure recovery.]

CyberX major contributions that participates in disputing such assumptions are, the online autonomous adaptation to changes, and the intrinsic resilience of the building blocks, the full time monitoring and surveillance of working Cells, the enhanced Self and situation awareness of the platform and the Cell itself. These contributions will work against the aforementioned attacker assumptions, or the goals behind such assumptions.

The dynamic adaptation to changes will work against the goal behind assumptions 1,and 3. The attacker assumed that if the defense applications are sharing the ToD platform or network, it can be affected by attacking it, or it can be utilized to disrupt the operation of the ToD application. The attacker can induce certain changes to static applications working on defense provisioning causing them to fail. Using CyberX dynamic adaptation work against that, as the defense services hosted on a CyberX managed platform will be able to adapt to any sudden changes and to adjust its working requirements and configuration to match the current state of change.

The intrinsic smart situational awareness of the platform building blocks, and the platform as a whole, works against assumptions, 1 and 4. The CyberX hierarchal management framework is capable of handling large scale networks and being fully aware of what is really happing within such networks.

The intrinsic recovery enabled by CyberX works against assumption 5 and the goals behind the five assumptions that the attacker was targeting. The main target for the attacker was to fail the defense services. With CyberX automated fast recovery, the attacker will not be able to easily

fail an application. The failed Cells “application thread” will be recovered from any failures autonomously. Additionally, the recovery system by itself realize a partial part of the moving target concept. The technique used for recovery intentionally deploy the replacement cells in a different geographical location, and with different platform configuration from the failed Cell to minimize any chances of re-failure. Doing so, move the attacker target which the application within the Cell away from the attacker. Giving this mechanism, it is not even in the attacker benefit to try to fail any of the COA Cells as doing so will make it almost impossible for him to target this Cell again.

The CyberX management framework uses a secure communication protocol that works against identifying the Cells hosting certain applications. Using such secure communication protocol is intrinsically needed to make sure that the attacker will not be able to allocate the Cell replicas. If the attacker was able to do so, it can disrupt the hot recovery system. Even if we assumed that the attacker will be able to go through all that, a single Cell can have multiple replicas running. Further, the Cell is always protected by default by the cold recovery mode and it will eventually recover.

2.8 **Conclusion**

In this chapter, we presented the CyberX platform designed to enable the CARD concept through supporting five main aspects: elasticity, diversity, awareness, cooperation, and intelligence. CyberX utilized the COA capability to induce autonomous execution elasticity and adaptability, and to enable adjusting the system’s shuffling and recovery policies at runtime matching the continual operational-environment changes. Further, CyberX used its situation-aware, autonomic adaptation and dynamic failure recovery mechanisms to enhance software resilience against

failures and attacks. Results showed that CyberX-managed COA-based software systems can efficiently adapt to maintain the desired reliability, sustainability, and resilience objectives even in hazardous, unstable environments at a reasonable overhead. There are several interesting challenges still to be addressed. These include utilizing application-level virtualization to enable seamless Cell migration across heterogeneous platforms, autonomous detection and profiling of environment changes; adjusting shuffling and recovery settings based on context; formalizing an automated variant generation system, and providing alternatives for legacy non COA-ready software.

Chapter 3

ChameleonSoft: Software Behavior Encryption for Moving-target Defense

> "Philosophy: A route of many roads leading from nowhere to nothing."
> Ambrose Bierce

3.1 **Introduction**

Biological inspiration in computer security dates, at least, to the definition of the term "computer virus" in the early 1980's [55]. Self-propagating malware and computer worms have clear life-like properties [56]. In nature, diversity provides a defense against such self-propagating threats by maximizing the probability that some individuals will survive and replenish the population with a defense against that particular threat. It has been noted that much of the vulnerability of our networked computing systems can be attributed to the monoculture or lack of diversity in our software systems [57]. It is practically inevitable that software will contain flaws. The software monoculture makes it easier for attacks to spread thus exposing the systems to large-scale attacks by well-informed attackers.

Inspired by the resilience of diverse biological systems in the sea chameleons, we propose a diversity-based defense mechanism against software attacks, termed ChameleonSoft. Sea chameleons or cephalopods employ multi-layer diversity for different purposes. For example, they leverage their capability to change their body color, texture and appearance to induce

diversity. Diversity is used to camouflage for defense, disguise for hunting, and change color for communication [58]. Similarly, ChameleonSoft utilizes spatiotemporal software diversity to enhance software system security, survivability and resilience.

ChameleonSoft is founded over Cell-Oriented Architecture (COA) based infrastructure managed by CyberX. As mentioned before, COA is a biologically inspired architecture with active components called Cells that support the development, deployment, execution, maintenance, and evolution of software. Cells separate logic, state and physical resource management. Cells are dynamically composable into organisms that are bound to functional roles at runtime. Such construction supports online programmability, hot code swapping and automated recovery. These features together enable what we term as "ChameleonSoft Behavior Encryption (or CBE)" akin to message encryption.

CBE applies spatiotemporal diversity in a way that makes the attack target in continual random motion evading attackers. CBE leverages the COA intrinsic separation of concerns to realize temporal and spatial diversity. Temporal diversity is applied by shuffling multiple functionally-equivalent, behaviorally-different software variants at runtime. In addition, CBE realizes spatial diversity by enabling runtime seamless migration of Cells from one physical host node to another. The goal behind that is to hide the potentially targeted software flaws that might be used to penetrate the system.

CyberX divides the missions of a huge software program into smaller tasks. Each of these tasks is assigned to one or more Cells executing sets of similar function and different-behavior executable variants. These sets might have different objectives targeting different quality attributes. Reliability, performance, robustness, and mobility are examples of such attributes. ChameleonSoft shuffles variants and sets to induce diversity. The scope of diversity application

extends beyond security goals to the other quality attributes. The system might shuffle to a variant that aims at high system performance in highly-loaded but low security risk situations. Alternatively, the system would resort to a higher security, perhaps lower performance variant in higher risk situations.

Researchers in [59] mentioned that multi-variant systems without appropriate recovery mechanism might face a larger amount of coincidental failures. ChameleonSoft relay on CyberX autonomous recovery system to handle any coincidental failures that might occurs due to diversity application. Such support increases the system resilience against international and unintentional failures.

Inspired by the sea chameleon dynamic change occur in response to frequent changes in the environment, ChameleonSoft autonomously and seamlessly changes the shuffling policy at runtime to suite the continual dynamic changes of the surroundings.

For the purpose of illustrating the details of our CBE prototype, we will present some details about software Chameleonization in section 4. The discussion will also clarify the main rules needed to enable software Chameleonization.

ChameleonSoft main contributions presented in this chapter can be outlined as follows:

1) CBE mechanism that applies multidimensional spatiotemporal diversity to mobilize attack target;
2) An elastic software platform that dynamically and autonomously changes shuffling policy to match the surroundings frequent changes.

Further, in chapter 5 we used analysis and simulation, to study the performance and security aspects of the proposed system. This study aims to evaluate the provisioned level of security by

measuring the level of induced confusion and diffusion to quantify the strength of the CBE mechanism. We also simulated the computational cost of security provisioning, and enhancing system resilience in ChameleonSoft with regards to the amount of failure downtime with and without CBE.

3.2 **ChameleonSoft moving-target defense**

In biology, sea chameleons, or chameleons for short, are well known for their capability to induce diversity. We promote the novel moving target approach by ChameleonSoft as a defense mechanism against software attacks. Inspired by the chameleon diversity employment for camouflaging, ChameleonSoft encrypts software behavior by employing multidimensional diversity. The outcome is continuous spatiotemporal changes of the network behavior to, in effect, move the attack target away from the attacker. ChameleonSoft is founded over our unique elastic CyberX managed Cell Oriented Architecture (COA) that enables spatiotemporal diversity employment.

Chameleons employ different diversity techniques to increase the resilience of their camouflaging process against attacker visual observation. Changing body color, texture, and appearance are examples for such techniques. They recover from a technique failure by switching to another technique. Similarly, ChameleonSoft applies different diversity techniques for camouflaging to enhance the system resilience against attacker utilization of possible software flaws. Applying diversity might involve multiple interruptions of the executing variants. Doing so might lead to multiple coincident failures. Therefore, ChameleonSoft leverages the CyberX autonomous, dynamic, situational aware, multi mode failure recovery mechanism to resolve

possible coincident failures. A major outcome of this recovery mechanism is the failure resilience enhancement not only against coincidental failures, but also against malicious induced failures by adversaries.

The CyberX autonomous, situational aware, multi-mode failure recovery mechanisms enhance the system resilience against both intentional and un-intentional failures. The details about CyberX managed automated recovery are illustrated in chapter 2. Figure 3.1, illustrates ChameleonSoft behavior encryption concept in realizing a moving target defense against software based attacks. ChameleonSoft uses and partially manages CyberX online configurability and the loosely coupled foundation to enable runtime multidimensional diversity application, in time by induce attacker confusion to encrypt the software execution behavior, in space to realize trace resistant moving target defense. ChameleonSoft uses the CyberX multimodal autonomous recovery to insure the resilience of its defense approach against coincidental or intentional failures.

Figure 3.1 ChameleonSoft reliable behavior encryption

3.3 **ChamelonSoft behavior encryption**

Typical encryption entails transforming the plain text into an unrecognizable message to the interceptor. Strong encryption schemes have two major properties namely confusion and diffusion. The confusion property virtually prohibits interceptors from predicting the ciphertext resulting from changing one character in the plaintext. An effective confusion is enforced via a complex functional relationship between the plaintext, key pair and the ciphertext. Confusion aims at maximizing the time that the attacker consumes to determine the relationship between the plaintext and the key pair. Diffusion is the other property of strong encryption schemes. Diffusion enables the cipher to spread the plaintext information over the entire ciphertext so that the changes in the plaintext affect many parts of the ciphertext [60].

Behavior encryption in ChameleonSoft is analogous to typical encryption in the way it exhibits the confusion and diffusion properties. ChameleonSoft induces confusion by dynamically changing the behavior of the executing software variant using stationary runtime shuffling of code variants “Temporal” and live-migration of Cells between heterogeneous hosts “Spatial”. The dynamic software behavior change makes it more difficult for an attacker to generate a profile with the possible flaws of the executing variant. The shuffling pattern is a supervised reflection for the continuous change in the environs. In ChameleonSoft, an effective confusion is determined by how complex to correlate the change in the output behavior relative to a single induced change in the environment.

ChameleonSoft works above CyberX platform that manage all the details of the Cell and the Cell network, and maintain its resilience against failures. ChameleonSoft add one new component among some changes to the management framework of CyberX presented in Chapter 2 in order to

support the software behavior encryption process. Figure 3.2 illustrated the new architecture of the management framework of the Chameleonized CyberX.

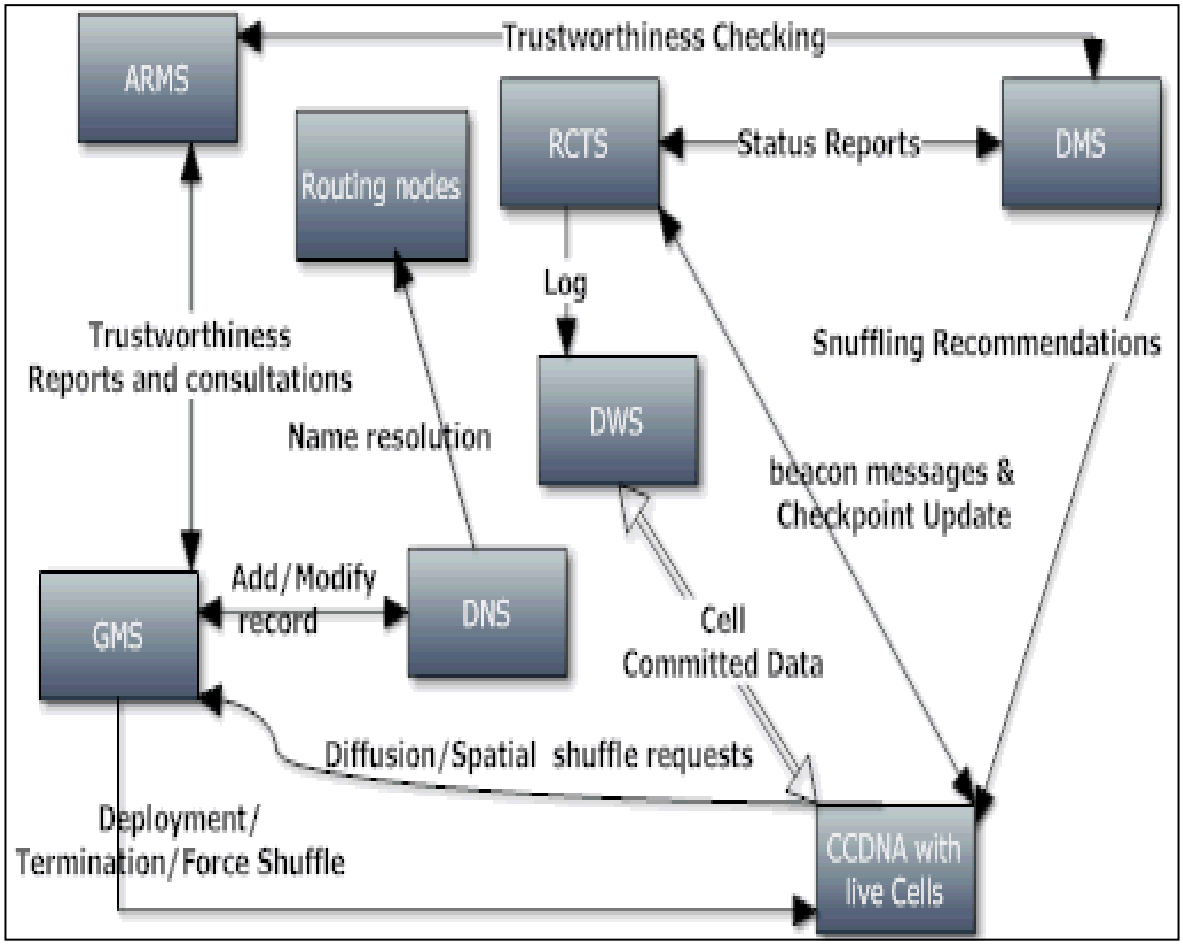


Figure 3.2: Chameleonized CyberX management architecture

The Diffusion Management Servers (DMS) are the main component added to the basic platform. DMS main task is to manage diffusion broadcasting messages. It is the only server authorized to send shuffle for diffusion messages to the platform Cells. The details of diffusion shuffling process are illustrated as follows:

ChameleonSoft induces diffusion by generating none-uniform random virtually untraceable significant change in the spatiotemporal network behavior using the Cell independent decision-making capability. Cells send diffusion shuffling requests to the DMS either in response to an

induced change or based on a predetermined policy as part of the encryption process. DMS receives diffusion shuffling requests from the Cells. It cooperates with the ARMS to make sure that the request is trustworthy, and the source has a good reputation and justification for the request. Then it cooperates with the auditing server to make an informed decision about the best area to send a diffusion shuffle recommendation messages. Cells who receives a diffusion-shuffling recommendation decides independently whether to comply or not, when to shuffle the current variant, the shuffling frequency, and the variant selection for the next shuffle. These decisions are guided by the situational awareness unit's frequent reports regarding the application requirements, and the host condition.

We propose a variant layout randomization technique to increase the level of CBE's confusion induction. The system assigns the variant shuffling index based on a predetermined sequence. Variants' indices are shuffled internally within each Cell based on a Cell independently generated random number that changes over time. This random number is used to shift the next executing variant selection index to a random location in the variant layout space.

Software behavior encryption by runtime hot shuffling of software variants is a realization of ChameleonSoft temporal diversity. ChameleonSoft realizes space diversity by seamlessly moving the Cell at runtime among different physical hosts. During this process, CyberX autonomously maintains communications, Cell sensitive data, and state logic.

ChameleonSoft can follow different shuffling policies at runtime to suit the dynamic change in the surrounding environment. A policy change might induce a change in the shuffling frequency for more security, or the shuffling orientation to favor time over space diversity or vice versa. Figure 3.3 illustrates the aforementioned software behavior encryption protocol variations. These

variations can be classified in either confusion or diffusion induction within the Cell network by manipulating the application behavior across space, time, and platform heterogeneity.

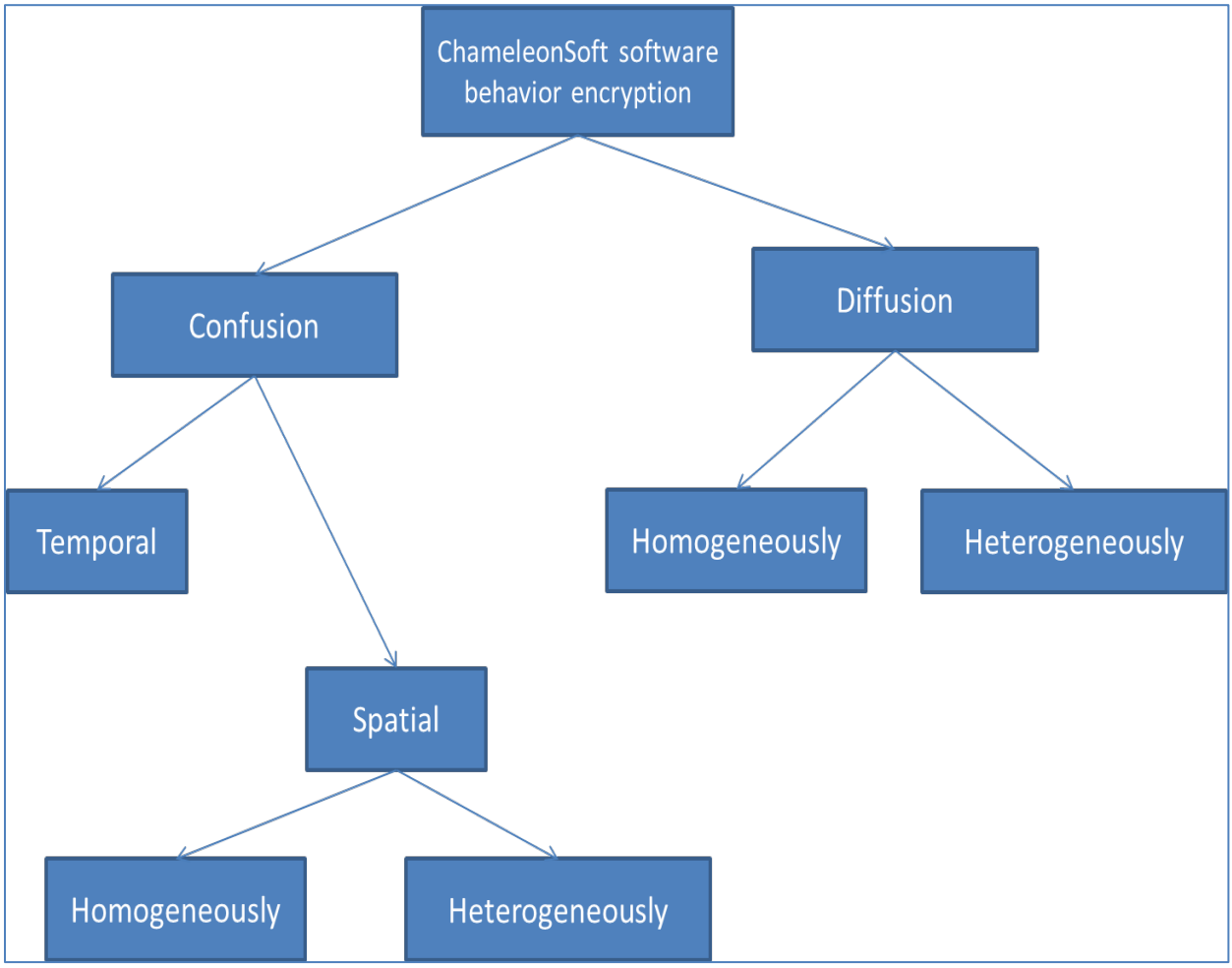


Figure 3.3 The software behavior encryption protocol variations

Figure 3.4 illustrates the software Chameleonization process. A COA ready application tasks are defined as multiple similar-function different-behavior variants grouped in different objective sets targeting different quality-attributes. These variants are loaded into different Cells at runtime. ChameleonSoft shuffle these variants locally for temporal diversity. ChameleonSoft migrate the live Cells between heterogeneous/homogeneous platforms to realize the spatial.

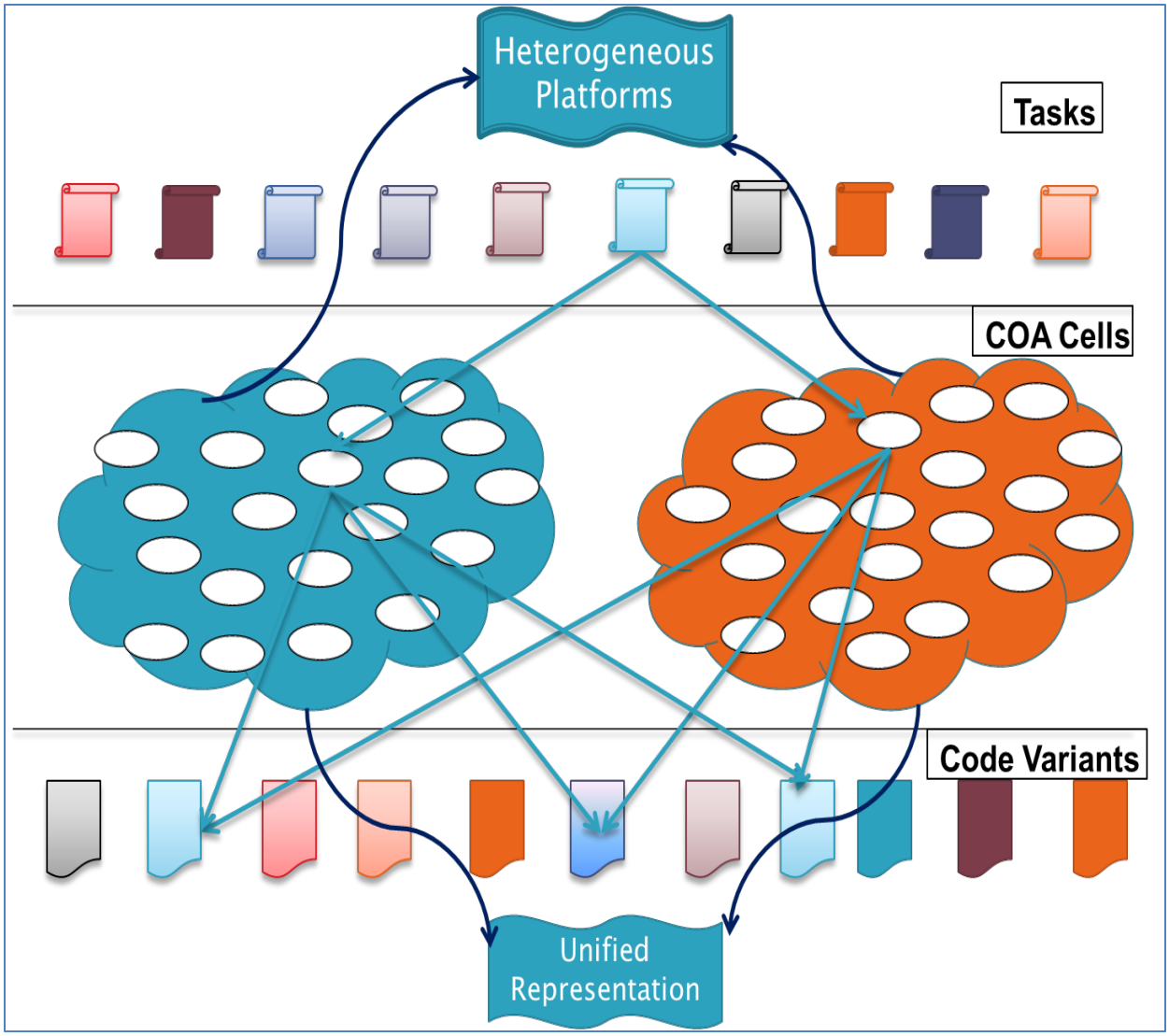


Figure 3.4 Application Chameleonization

The overall diversity induced by our system can be expressed in the form of ***X*** missions represented in ***Y*** roles. These roles are played by ***M*** organisms, composed of ***K*** Cells. Each Cell has ***P*** quality attribute sets containing ***Z*** software variants, to be executed and migrate between ***Q*** nodes with ***W*** different configuration-combinations and an average of ***R/S*** shuffling events/sec.

We will provide more details about CBE realization methodology through the description of ChameleonSoft implementation presented in section 4.

3.3.1 *Variant generation*

The whole idea of software Chameleonization depends mainly on the availability of similar function different behavior variants that can be grouped in different objective sets. Recent research work presented multiple techniques to automatically or manually generate variants with similar functionalities and different configurations and compositions in order to produce different execution behavior at runtime [61].

Automated code variation techniques have focused on creating code diversity (e.g., instruction set randomization [62, 56] and reordering of allocated memory objects or blocks (e.g., address space layout randomization [37, 64]. Other possible variations that could provide a considerable level of diversity was mentioned in [65] including varying, scheduling, system calls, calling conventions, configuration properties, and Instruction set randomization.

Researchers in [41] utilized one or more diversity technique to diversify variant execution in order to detect behavioral deviation of simultaneously executing variants processing the same input, as a way to detect attacks. Diversity is utilized by [67, 56] to detect attacks like buffer overflow attacks. Instruction set tagging and memory space portioning is used by [64] to generate variants with no specific common flows that might be utilized by a specific category of attacks. The authors' target was to use this diversified versions for attack detection, and fault tolerance.

In ChameleonSoft we intend to utilize multiple variant generation techniques to satisfy the requirements needed to enable behavior encryption, and runtime dynamic quality attribute manipulation. For example, we can use the mechanical transformation approach presented in [64] to generate variants sets with specific resilience against certain class of attacks. Within each set we can use the N-version programming originally presented in [67,70] for fault tolerance and

proposed for security by [59,73] to generate multiple similar function different behavior variants using different independent development groups, or at least different compilers processing the same program-specifications. Additionally, we will use formal behavior computation techniques like the ones presented in [74,54] to verify the functional similarity, and to compute the behavioral difference between the generated variants. These computations will assess evaluating the strength of our CBE.

The mandatory requirements needed to support COA based Chameleonization can be summarized as follows, The application developer has to build a checkpoint enabled application and reports execution advance to the underlying infrastructure using a dedicated name pipe. All data has to be committed before all Checkpoints. Applications should support random startup from a bootstrapping-time provided checkpoint. Applications will be given direct access to the host memory to save only noncritical temporarily data, all critical data should be accessed through the infrastructure dedicated data path “Static, remote, and separately managed Data warehouses". Developers should provide at least two similar-function different-behavior variants to enable temporal shuffling. Developers have to inform the host Cell about all uninterruptable tasks using a provided communication script. Application designers also provide a brief description about the application tasks, any special requirements, the needed security level, and the estimated resource usage… etc “syntax provided”.

Behavior encryption strength depends mostly on the following; the behavioral distance between the variants, number of variants in each set, number of hosts available for spatial shuffling, the level of host configuration-diversity, and the granularity of the application design “number of fractions”.

3.3.2 *Decision making in ChameleonSoft*

In chameleons, color shuffling decision making source and location depends mainly on the targeted changing speed. In fast changing chameleons, shuffling decisions are mostly controlled by the brain with dedicated connections “nerves”, or through distributed decision making Cells all over the body. In ChameleonSoft, we favor the later approach as it is more realizable and computationally cost effective from the communication and resource consumption point of views. The decision-making unit in ChameleonSoft is an intrinsic Cell component enabling independent decision making. More complex decisions affecting a group of Cells or organisms are handled by GMS. These units are responsible for directing the network behavior change for global purposes. The decision making unit depends mainly on the situational awareness unit to guide its decisions.

There are multiple levels of local situational awareness, between the host Cell and the variant executing over it, and between the Cell and the CCDNA hosting it. A dedicated channel between the Cell and the executing variant will allow the Cell to be aware of the application requirements. Certain syntax is provided to the variant designer to be used for message exchange between the variant and the host Cell to inform it with its needs and requirements, like holding shuffling process until a cretin non-preemptive task complete. Another level of situation awareness is achieved by the use of a group of sensors in the form of API’s. These sensors are frequently used between Cells and the CCDNA hosting them to sense any phenomena of interest. The sensors’ feedback, incoming application requirements, and the GMS regular global report feeds are the main source of information supporting shuffling policy change to be discussed in the next subsection.

ChameleonSoft uses a set of smart processors similar to the one described in Chapter 4. To guide the Cell decisions based on various feedbacks from the aforementioned sources “The neighborhood Cells, application, and management servers” . Figure 3.5, illustrate the confusion

and diffusion shuffling process regarding, when to shuffle, how to shuffle “space or time”, and which variant to swap to in the next shuffle. ChameleonSoft uses different smart processors that have similar design and shares the same input but executes different logic to handle each decision. The logic description is similar to the one presented in Chapter4 “EvoSense”, and it can vary depending on the designer needs and representation. For simplicity we use a simple rule based expert system to represent the logic within such processors. The same technique is used to guide decision making in regards to changing the shuffling policy as illustrated in the next subsection.

The confusion induction process starts at the Cell level; the Cell uses built-in analyzers “smart processors” analyzing the incoming feedback illustrated in Figure 3.5 to take confusion or diffusion decisions. The smart processors logic is loaded to the Cell as a part of its deployment package.

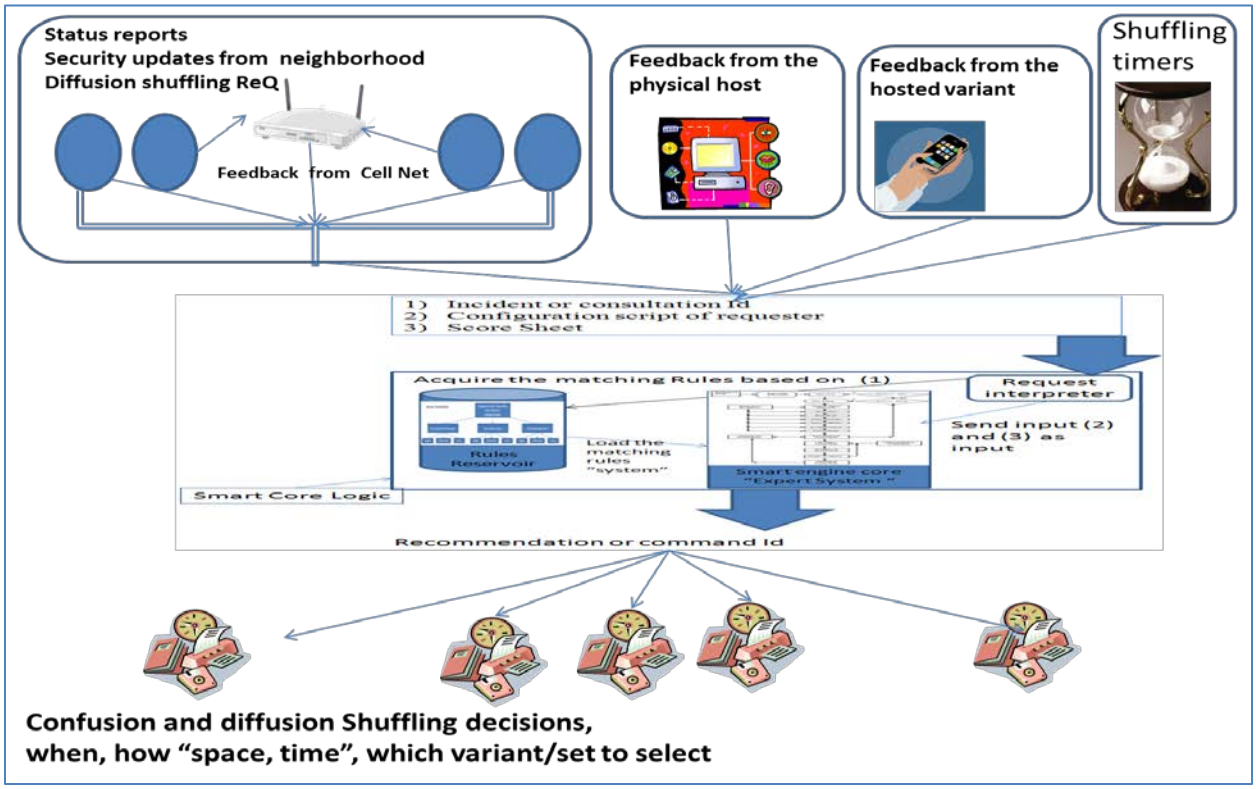


Figure 3.5 The Cell, confusion and diffusion shuffling

Figure 3.6 illustrates the encryption protocol and the decision making process involved. The smart processors get the feedback from the different inputs at runtime and decide if a shuffling event is needed or not, and what type of event to be executed. Then the current state of the application is checked if it had any holds or reduces of shuffling frequency requests or not. The host condition also is checked for needed resources for the next variant after selecting this variant by another smart processor. Special shuffling would require sending a request to the DMS to handle it with all the details about the reasons for that request. Such reasons are described as a summarized log of the Cell and application state. If the situation permits the execution of the shuffle or the migration then it occurs automatically as described in section 3.4. if not, then the process will be repeated after a predetermined threshold and the failure event will be recorded and poster to the ARMS.

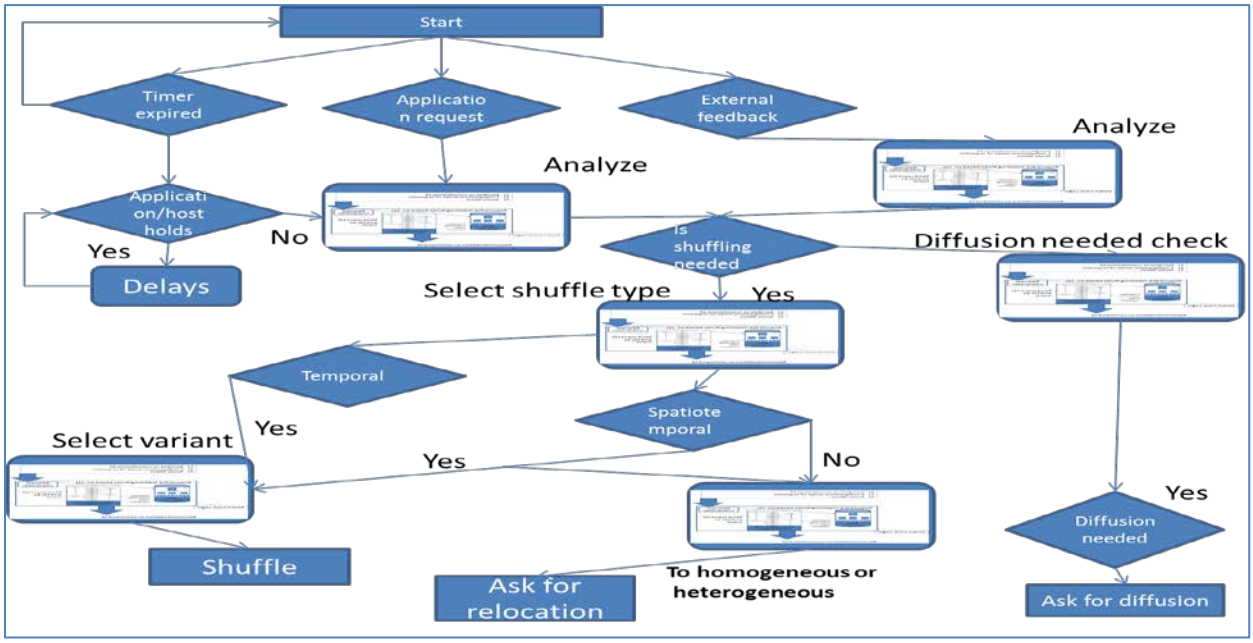


Figure 3.6 The encryption protocol and the decision making process

The diffusion management servers receive diffusion requests from working Cells and process these request as illustrated in Figure 3.7. The DMS receives requests, and consult the ARMS for the requester reputation, and the latest report about the Cell net. The reports are analyzed and the best locations are spotted. The recommendations are diffused through the master routing nodes within the selected areas. The Cells within this area decides whether to comply with the request or not based on the analysis of the various inputs and the application status as presented in Figure 3.5Figure 3.6.

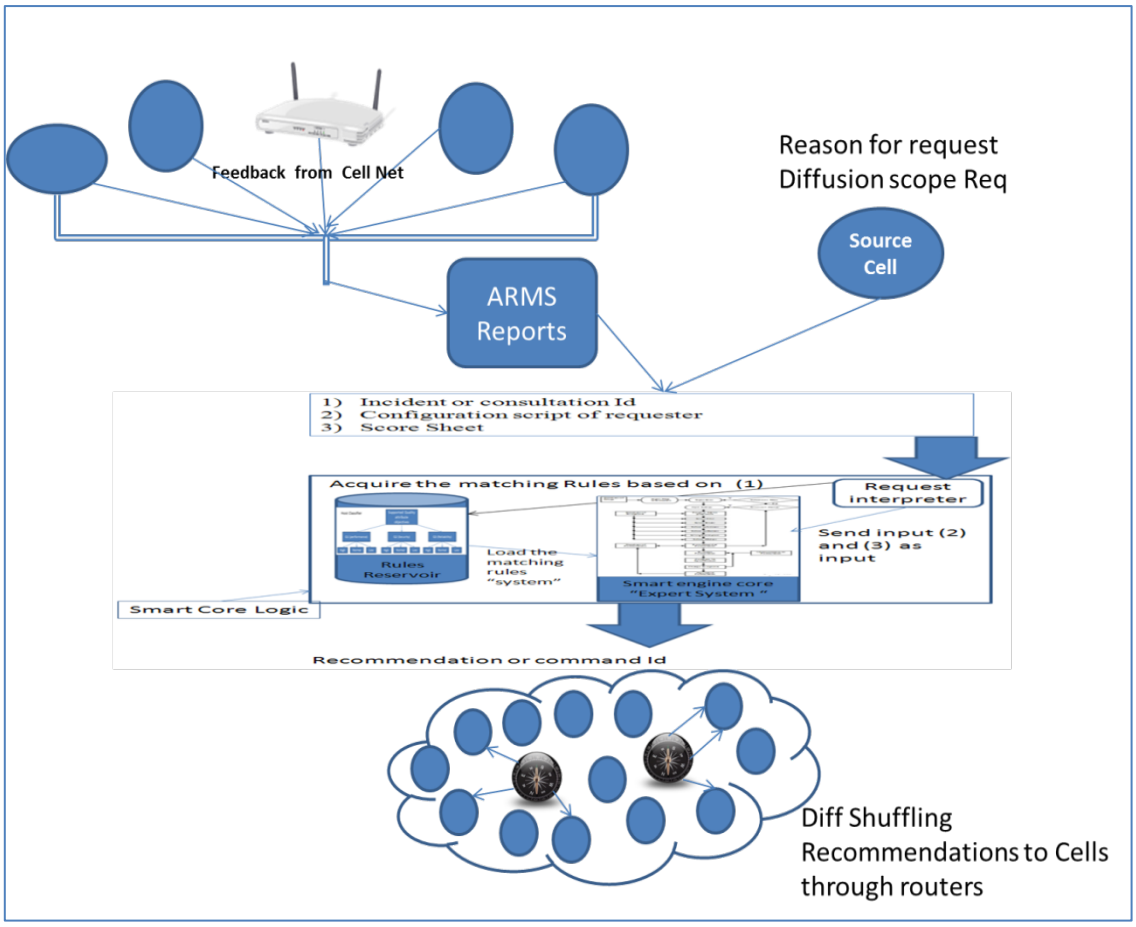


Figure 3.7 DMS diffusion recommendation process.

3.3.3 *Shuffling dynamic policy change:*

ChameleonSoft shuffling policy is the main guide for the decisions being taken by all the diversity management units locally within the Cell, and at the management level. The shuffling policy defines the shuffling type “temporal or spatial” and the shuffling frequency for each type. The shuffling policy also includes when, and why to ask for a diffusion shuffling. In addition, shuffling policy also determines the nature and the roles of variant-set shuffling for quality attribute manipulations. The configuration of the Cell shuffling policy is always in continuous change in-response to changes within the Cell surroundings and the application requirements. As illustrated in Figure 3.6, the shuffling policy is always analyzed at runtime to decide the next confusion induction process will be temporal or spatial, and the level of heterogeneity required in the spatial shuffling.

3.4 **ChameleonSoft Implementation**

As mentioned before in chapter 2, the COA Cell can be built in different techniques based on the targeted resource virtualization depth. We implemented the simple and fast version of the Cell to enable quick development of a CBE prototype. We intend to realize a more complex version of the Cell utilizing one of the application virtualization techniques mentioned in [76] as a more advanced version of the CBE prototype.

The main differences between these two versions are illustrated in chapter 2, we will not go through the details in this chapter. However, we will illustrate through Table 3.1, a quick comparison between the conventional virtual machine migration, and CBE on CyberX spatial shuffling as it is an intrinsic part of the software behavior encryption realization procedure. The main objective behind such comparison is to illustrate the inherent need for COA features to enable CBE spatial shuffling.

The main difference between CBE spatial shuffling and virtual machine migration is that virtual machine migration is a computationally heavy process, and it needs complex modifications to enable the kind of real-time diversity employment-dimensionality provisioned by ChameleonSoft. Working with virtual machine concepts as known in the literature [43,44,45] is not feasible, because of the cost of diversity employment, communication bandwidth, and cost of failure recovery for such huge capsules.

CBE uses CyberX and COA fine-grained application development and single task capsules with a total separation between the main design concerns, Data, Logic, and Physical resources. Such separation facilitates the temporal and spatial shuffling with minimal computational, and communication cost. In addition, CyberX handle failure recovery intrinsically and with a minimal resource usage, and downtime. The cost reduction are mainly the outcome of the COA fine-grained application design, the utilization of lightweight capsules, high level of automation, intrinsic consideration of failure recovery, and the separation between the data and the mobile capsule itself. Table 3.1 illustrates by comparison the difference between conventional virtual machine migration and ChameleonSoft behavior encryption.

	Virtual machine migration	CBE
Definition	Moving “by cloning or startup-time replication” a live machine “whole OS and applications” from one host to another with the exact similar configuration	Live employment of multidimensional software diversity to, in effect, induce spatiotemporal “software behavior encryption” and a moving target defense.
Diversity technique	Migration	Migration of single Cell “one application” either by swap-time cloning or replication// and Internal

		shuffling "swapping code variants at runtime"
Resource usage	High	Limited
Granularity	Full OS	Fine granularity, application fraction migration
Methodology	OS stalling, physical transformation, Resurrection " excessive bandwidth usage"	Local/Remote replication while execution, communication rerouting, and source termination. "no excessive bandwidth usage"
Shuffling downtime	1) High 2) relative to the OS size, number of application, and the whole machine load time.	1) Virtually none, Limited to the time needed for communication redirection "negligible" 2) worst case scenario it will be relative to the load time of a single application.
Ability for temporal shuffling	Possible explicitly with large computational and downtime cost by enabling check-pointing for most of the OS components.	Intrinsically possible, minimal computational load.
Diversity application related bandwidth usage	High	Virtually none
Resource usage involved	High	Limited

Table 3.1 Comparisons between Cell spatiotemporal shuffling / virtual machine migration

3.4.1 *Software Chameleonization process*

As mentioned before a Chameleonized program is a program that enables check-pointing with

at least two similar-function different-behavior variants to enable temporal shuffling. The checkpoint reporting location has to consider data integrity requirements especially in case of failure. All data has to be committed before checkpoints.

At the deployment time, a new DNS record will be created by the GMS for each Cell indicating the application virtual name to be used for inter-variant communications “if needed by the application designer”, the Cell unique id for inter-Cell communication, and the IP of the physical host hosting the Cell.

The deployment starts by the CCDNA receiving the deployment package from the GMS including the Cell globally unique ID(s), the initial checkpoint value, variant pool setup “variant binaries, names; numbers; sets; variant-classification” , the configuration script describing the specs of each variant, the global objective of the application, and any specific specs added by the developer to be considered at time of execution “number of application fractions; fraction-names; ..”, the initial shuffling policy, and the needed security level.

The CCDNA starts the Cell by constructing the components mentioned in sec 3.1.1 with the provided unique id. Then the CCDNA starts to interpret the deployment configuration file in order to generate separate configuration files for each Cell unit describing any modification in its default task assignment, or special considerations to be taken care off at the time of execution.

The execution starts when the execution unit asks the STM for the starting checkpoint, the STM will get this information as a part of the deployment configuration file. STM will repeatedly provide this information to the execution unit at each shuffling event. The execution unit starts to launch the first variant while passing the appropriate bootstrapping parameters.

The last executed checkpoint value will be held by the STM locally, and remotely at the RCTS

that will receive it via the Cell beacon messages.

At runtime, variants will update the STM frequently with the checkpoint advance and any other special needs via a dedicated communication channel.

At the time of shuffling, the Cell diversity manager gives the shuffling signal to the STM and the execution unit, which will start the process after the next reported Checkpoint and based on the provided shuffling orientation as follows;

Temporal shuffling, we have mainly two realization modes for temporal shuffling the greedy and the light modes. The system designer can select either one of them based on the available deployment-platform host resources, and the criticality of the application. **The greedy-mode** with seamless handover offers virtually no-downtime but duplicates the resource usage at the time of shuffling, and the **light-mode** offers no-resource increase at the time of shuffling on the account of increasing the transition time by the time needed for variant loading and synchronization. We will briefly describe both.

The greedy-mode "local replication": Upon reception of the shuffling signal, the execution unit starts to load the new variant in freeze "ideal" mode. The new variant will connect to the STM that will locally synchronize the execution checkpoint with it. The communications unit will duplicate all the inputs to the old and the new variant. Upon reception of the ACK Signal from the STM and the communications unit confirming that the synchronization is completed, the execution unit sends pause signal to the old variant, and a resume signal to the new one followed by a termination signal to the old variant.

The light-mode: Upon the reception of the shuffling signal, the execution unit starts by local synchronization with the STM for the checkpoint update. Then it pause the old variant, and

informs the STM and communication unit about the execution hold. The communication unit will buffer incoming messages for the duration of the handover. The execution unit will terminate the variant, and starts loading the new variant with the last known checkpoint, and informs the communication unit and the STM about the successful loading to resume execution. The communications unit will send any buffered messages to the new variant.

Spatial shuffling: the Cell diversity management unit starts by asking the GMS to deploy new Cell as a new spatial shuffle destination and reply with the distention temp id. The new Cell will be deployed in replication mode with 2 ids the Source id, and a temp id valid only for the duration of the handover. The source Cell will treat the destination as a replica, and inform the routing nodes to forward a copy of all incoming messages to it. The GMS sends a resurrection signal to the destination Cell when the process completes, followed by a termination procedure to the source Cell. The GMS will fix the DNS and erase the temp id entry.

Diffusion induction and set change: let us take an example, an attacker might be able to induce a change in the system surroundings, like a DOS attack to overload the network to force the system to shuffle the currently executing variant-set. The Cells close to the induced event change their variant set to target a different quality attribute (e.g. performance) that suits the induced change in the environment. Based on the configuration policy this incident requires a diffusion shuffling. The affected Cells send their request and incident report to the DMS. DMS will check for the trustworthiness of the request by the help of the ARMS, then it selects the most appropriate destination based on the current policy, security requirement, and load balance threshold. DMS will forward the request to the appropriate routing nodes in the targeted area for broadcasting.

The diffusion scope and direction depend on the ARMS feedback, and the situation evaluation

at the requester area" possibility of attacks, network problems,…..etc". The destination area will be selected to the best interest of the whole network taking into consideration the performance aspects of the Cells in the destination area.

Cells who receive shuffling recommendations will take their own decision independently whether to comply or not. The decision is guided by the current local Cell policy and the status of its working-context reported by the situational awareness units. Cells who decide to shuffle will replace the current active variant by another variant from the same set preserving the previously targeted quality attribute.

3.5 **Security analysis**

In this section, we discuss the security aspects within ChameleonSoft. A threat model will be presented identifying the list of assets ChameleonSoft have from the application, and the infrastructure point of views. We list some of the imminent threats facing moving target defense. Additionally, we will present ChameleonSoft as a countermeasure for these threats. Finally, we present an analytical study of CBE strength.

3.5.1 ***Identifying the assets***

ChameleonSoft as a moving target defense is designed to protect the application survivability, stability, and integrity, and to minimize the cost of development. ChameleonSoft reduce the time and effort of software verification needed to locate hidden implementation and design vulnerabilities that can be exploited to launch attacks. The only added requirement needed to support Chameleonization is the necessity of respecting the separation between data, and logic. Accomplishing that, enables ChameleonSoft to focus only on maintaining a valid execution

workflow. The application **main assets are application survivability, execution maintainability, and data safety**.

3.5.2 *Identifying the threat*

Many of the well-known software attacks that has been identified for more than two decades "Ex, Buffer overflow,.." are still being used by attackers to gain control-of or to crash an executable software. The current software defense mechanism has proven to be inadequate to insure software execution safety. Moving target defense was introduced as a solution minimizing the attack surface by mobilizing the attack entry points "exploitable vulnerability" away from attackers.

The authors of [77] argued that the actual benefits of the conventional applications of moving target approach are in fact often much less considerable than one would expect. They analyzed the security properties of a few example defenses and attacks/ attack-classes, and identified scenarios where moving target defenses are and are not effective. Table 3.2 summarizes their effort towards estimating the effectiveness of dynamic diversity presented by the current technology against five important attack classes. The study illustrated that conventional dynamic diversity yields no benefit for circumvention and deputy attacks, since the attack does not depend on guessing the randomization key. For brute force and entropy reduction attacks, the benefits of conventional dynamic diversity are marginal and only increase the attacker's workload by at most a factor of two. Dynamic diversity holds the most promise for probing and incremental attacks. Giving the current technology, the rate of re-diversification required to obtain tangible benefits, especially against probing attacks, appears to be very high [78].

The authors concluded by illustrating that the effectiveness of the diversity utilization for moving target defense could be enhanced by combining more than one diversification technique and by utilizing the N-variant approach.

Circumvention attacks	No advantage
Deputy attacks	No advantage
Entropy reducing attacks	At most doubles expected attack time
Probing attacks	Very high rate of randomization may thwart attack
Incremental attacks	May provide significant advantage

Table 3.2 Effectiveness of moving-target defense

3.5.3 *ChameleonSoft as a countermeasure*

In the subsections, we will briefly describe each attack class presented in [77] and how ChameleonSoft can, be an adequate solution to mitigate or at least extremely complicate applying such attack. We will focus on illustrating that using multidimensional spatiotemporal runtime diversification can be sufficient to mitigate this list of attack-classes within the lifetime of a Chameleonized software variant.

The circumvention attack is one of the most dangerous attacks; where attacker strategy is to circumvent the diversification entirely [77]. This can be done if the attacker finds any exploit that does not depend on the properties altered by the diversification. A good example of this attack strategy is the return-oriented programming that was presented in [76]. Return-oriented programming can be considered as a more general form of the well-known return-to-libc attack

[79]. The main difference is that instead of relying on the functions provided intentionally by libc, return-oriented programming exploits fragments of code found in the binary to provide a Turing-complete programming system without needing to inject any code.

CBE presents multidimensional spatiotemporal diversity with a frequent and Cell-independent change of diversity-employment frequency, and orientation. Such dynamic unpredictable changes massively widen the diversification scope increasing the attacker uncertainty about the spatiotemporal location of any utilizable attack point. Giving the COA fine grained application design, and ChameleonSoft, multidimensional diversity application it is very hard for an attacker to trace and synchronize her attack with such complicated diversification methodology.

In addition, utilizing CBE non-uniform random special diversity where the Cell itself has no idea about the migration destination adds a physical barrier against such attacks.

In a confused deputy attack [80], an attacker finds a way to use a benign program in a malicious way. For the conventional defenses, the main fear was that an attacker will be able to find a way to use the program to apply the randomizing transformation to the attacker's data. doing so reveals the details about the diversification mechanism being used.

This attack might be executed in our Chameleonized environment if the attacker was executing a malicious Chameleonized application working as a probe or data collector nodes to reveal the diversification mechanism used by ChameleonSoft. Fortunately, such attacks will not succeed, as each program executes in an encapsulated environment within a dedicated host Cell. The Cell takes independent real-time local decisions about the diversity employment methodology. Knowing the details about the diversity application methodology of one or more Cells, does not have any impact on, or reveal any information about others Cells. In addition, maliciously utilizing one of the Cells will be detected and resolved by the infrastructure ARMS.

For the brute force and entropy reduction attacks where the attacker simply attempts all possible randomization keys until an exploit is found that succeeds. Researches in [77] started that if the key space is small enough, such attack may be practical.

In ChameleonSoft, the search space is supposed to be large enough to circumvent such attack. The application is fractionized into parts, encapsulated into Cells, and deployed randomly on remote hosts. These Cells constantly apply stationery temporal shuffling, and frequently migrate between heterogeneous platform/hosts. An attacker targeting specific vulnerability that can be exploited aiming to crash certain service or a host has to resolve this dynamically changing puzzle within the lifetime of the code variant that holds such vulnerability.

A probing attack attempts to overcome a diversity defense by using probe packets to learn properties of the randomized execution. A probe attack is distinguished from a standard entropy reduction attack in that the probe packets are designed only to obtain information about the target, rather than to produce the desired malicious behavior [77]. An incremental attack is a form of probing attacks where more than one successful probe is needed to obtain sufficient information to construct the exploit.

The attacker can utilize the mechanism illustrated before by deputizing malicious Cells to collect information about the shuffling scheme, or even use local probes in the host for the same objective. In all cases, the information collected will not be useful as each Cell takes its own decision about the diversity employment mechanism and orientation.

We finally conclude that using ChameleonSoft multidimensional spatiotemporal diversity employment increase and widen the attacker search space for an utilizable vulnerability that can be used to penetrate the system. The independent runtime behavior change complicates the mission for probes to collect useful information about executable software. Even with incremental

distributed multithreaded data collection over large portion of the COA-Cell network, it is too hard to synchronize the attack to cope with ChameleonSoft induced uncertainty and independent decision-making.

After all, even if the attacker was successful, chameleonized software are always protected by the CyberX multimodal dynamic recovery. If the attacker managed to crash specific Cell/Host, the Cell will anonymously be recovered with a minimal downtime.

Additionally the system is built to emphases the idea of global situational awareness, where infrastructure management is aware of what is happening in all the executing Cells. A partially successful malicious attempt on any part of the network will most probably be avoided in the future not only on that part but also over the entire network.

3.6 **ChamelonSoft behavior encryotion mechanism "The Key"**

Claude Shannon in 1949 introduced the confusion and diffusion properties of operation as a way to quantify the strength of secure cipher. We present a key like mapping between the key in message encryption and CBE key to enable quantification. We built our mapping based on the fact that both keys have similar semantics but they are not exactly equivalent. Message encryption depends on a key, while CBE depends on a set of parameters that maybe constructed as a key or genetic material. We used measures of confusion and diffusion to evaluate the strength of our CBE. CBE is an encryption technique with added unique characteristics.

We are not the first to make this mapping for the sole purpose of quantification [81]. Otherwise, it would be hard to find a methodology to quantify the level of provisioned security. There might be other methods, but this was the one that we choice.

In normal encryption, the main objective is to produce an unbreakable cipher by prohibiting the readability of the plain text, while in our case our objective is to prohibit the tractability of the temporal or spatial location of any software vulnerability. We do not use the term encryption for confidentiality; we address it in the terms of un-traceability.

In Table 3.3 we present a comparison between the CBE and the conventional message encryption. The table presents the points of similarities and differences, in addition to clarifying the definition, objective, methodology of realization of the term confusion and diffusion in both CBE, and message encryption.

		Message encryption	Behavior encryption
Definition	Definition	it is a conditionally reversible transformation of information (plaintext) using an algorithm (cipher) to make it unreadable to anyone except those possessing special knowledge needed to reverse the operation, usually referred to as a key.	it is an untraceable multidimensional transformation across time, space, and platform heterogeneity of the software execution behavior by employing runtime spatiotemporal diversity over executable code variants. The process must be irreversible within the lifetime of the software execution.
	Objective	Unbreakable cipher	Untraceable cipher
Confusion	Definition	refers to making the relationship between the plaintext and the ciphertext as complex and involved as possible	it is the process of inducing random and intentional extremely un correlated changes "across time, space, platform heterogeneity" in the runtime execution behavior of software to complicate establishing a relationship between the encrypted behavior and the expected normal change in the execution

			behavior in-response to an induced malicious event.
	Method	Enforced via a complex functional relationship between the plaintext, key pair, and the ciphertext, single change in the input result in massive change in the output.	dynamic repeated multidimensional runtime transformation of executable code variants across time, space, and platform heterogeneity
	Objective	Maximizing the time that the attacker consumes to determine the relationship between the plaintext and the ciphertext key pair.	Maximizing the time needed to allocate in time and space the exact location of a specific flaw-full software
	Evaluation	Measuring the bit mixing property as an indication for the complexity of making a relation between the key and the cipher text . “Ex, the frequency test”	Measuring the amount of the overall spatiotemporal changes in the output behavior of the network over time, and in response to induced changes.
Diffusion	Definition	Refers to the property that redundancy in the statistics of the plaintext is "dissipated" in the statistics of the ciphertext.	Refers to the non-uniformity in the distribution of the behavior change in response to malicious induced change in the input behavior should be redistributed into the non-uniformity in the distribution of much larger structures of the output behavior to dissipate the change across the whole network, which is much harder to detect.
	Objective	Complicating statistical attacks by making any changes in the plaintext affect many uncorrelated parts of the ciphertext	Complicating statistical tracing attempt by making intentional and non-intentional changes in the network execution behavior independently affect many uncorrelated locations in the network.
	Methods	Dissipating the statistical structure of the plaintext in the long-range statistics	Dissipating multiple independent random behavior changes across

		of the ciphertext.	time and space "by shuffling" in response to any intended or random changes in any part of the network.
	Evaluation	The amount and the level of change dissipation in the output in response to a single change in the input “Ex, Avalanche criteria”	The amount and the level of dissipation in the distribution of change in the output behavior in response to a single change in the input behavior

Table 3.3 Comparison between message encryption and ChameleonSoft behavior encryption

We consider CBE as a multi round encryption mechanism, where the output of a single round is always valid for use directly without decryption within the lifetime of the round. Additionally, CBE has no limits for the number of rounds needed to produce the final output. CBE use the output of each round as an input to the next round in order to increase the complexity of the cipher linearly overtime.

CBE is an encryption scheme that does not need any key management or exchange mechanism. The reason behind that resides in the fact that CBE output is useable without decryption. Additionally, CBE is an event driven encryption scheme; where the encryption key, the inputs, and the outputs have multiple elements representing different events affecting the encryption processes.

In our system, rounds are time slots. A single time slot might contain multiple temporal or spatial shuffles for confusion induction, or random independent distributed variant changes for diffusion induction.

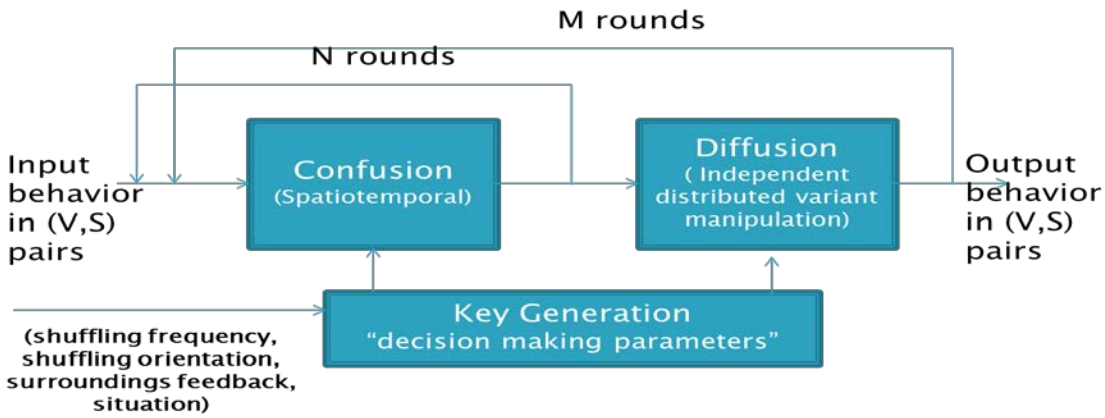


Figure 3.8 Behavior encryption process

Figure 3.8 gives an abstract view of the behavior encryption process. The process involves multiple rounds of confusion and diffusion guided by dynamic parameters. These parameters represent the key to our encryption mechanism. The confusion process is responsible for temporal and special change in the current behavior by manipulating the Cell location and/or the executing variants. The diffusion process is a random change in the execution behavior of multiple Cells based on independent decisions guided by distributed recommendations to diffuse the changes all over the network in an intractable way. The whole process is guided and controlled by the "key" or the decision making input parameters.

The level and the nature of the induced confusion, and diffusion depend on the input parameters and a set of runtime events. The following is a list of inputs, outputs, and events that can significantly affect the decision-making process controlling the diffusion and confusion processes.

The encryption module input parameters are mainly the {*Current variant Id, Set Id, and location Id*}. CBE has two main incoming-events triggers that affect the encryption process {*the current temporal shuffle time stamp, and spatial shuffling time stamp*}. The encryption module output parameters are {*Next variant id , Destination Cell X,Y location*}. The output new list of

events triggers are {*Change the current active Set , current active recovery mechanism, Shuffle for diffusion requests, Next temporal shuffling time stamp, and Next spatial shuffling timestamp*}.

The encryption key that determines the configuration for the next encryption round is composed of the following; Application requests regarding shuffling " *uphold, hold, delay, speed*" , Application request to match new quality attribute, Current temporal shuffling frequency " *time frame for the next shuffle*" , Current spatial shuffling frequency " *time frame for the next shuffle*", Incoming Shuffle for diffusion requests, and Change current targeted quality attribute request based on external input. The encryption module output is mainly the input of the next round.

3.6.1 ***Evaluating the strength of CBE***

Many algorithms are computationally strong, or practically unbreakable, in the sense that the resources required for timely cryptanalysis are either unavailable or prohibitively expensive [82]. In practice, a system needs only to be strong enough to provide a level of security commensurate with the risk and consequences of breakage in some specified period of time. Increasing the strength of the cryptographic system usually increases its cost and degrades system performance, so no more resources than the expected resource loss resulting from breakage should be invested in encryption

One of the methods used to assure the strength of a certain algorithm is determining the key strength against brute force attack considering the lifetime of that key.

Let us assume that the following is a list of possible values for the key presented in the previous section for a Cell ***i*** in a network of ***k*** Cells.

Variable	Possible range of values
Application requests regarding shuffling	{u,h,d,s}+ possible 2 digit decimal number holding the time out for the current command or the shuffling speed; total of 6
Application request to match new quality attribute	{number of available quality attributes} ; Z
Current temporal shuffling frequency	2 digit decimal number
Current spatial shuffling frequency	2 digit decimal number
Incoming Shuffle for diffusion requests	{S,NULL} , total of 2
Change current targeted quality attribute request based on external input	{number of available quality attributes} ; Z

Table 3.4 CBE strength evaluation parameters

We used the values in Table 3.4 to estimate the total number of permutations needed to crack the key of this single Cell ***i***. We assumed that the attacker has a prior knowledge of the current input, the encryption algorithm, the Cell independent parameters, the variant shuffling selection mechanism, the variant pool and the current indexing mechanism and seed.

The total number of permutations needed to crack that key for Cell ***i*** is a minimum of $Z*Z*2*2*2*6= 48Z^2$ permutation.

The attacker has to collect all these information and carry her calculations for that key within the time frame of a single shuffle for that particular Cell ***I*** out of ***k*** Cells.

Regardless of the impracticality of such attack within such time constraints, Cells are highly dynamic, and many of these parameters will be changing over time. Based on the fact that Cells do not need to maintain any static information to be used for decryption, they can frequently change the random seeds they use for local memory indexing, variant selection, and migration destination selection.

In a mono-variant static system it is easy for an attacker to diffuse the same attack all over the network with all nodes executing the same variant. Even if there was a recovery mechanism, attack reapplication will cause a denial of service due to the excessive downtime due to multiple failures. In CBE compromising a Cell is independent of compromising other Cells. An attacker with full control over a certain Cell will be harmless to other Cells. On the contrary, any malicious usage of this compromised Cell will result in increased level of security provisioning all over the network.

3.7 **ChameleonSoft role in mitigating the BlackWidow attack**

In this section we intend to discuss the ability of ChameleonSoft to invalidate the attacker assumptions on the case study “BlackWidow attack scenario” presented in Chapter 1. We list part of the assumptions listed in Chapter 1. We focus only on the assumptions that ChameleonSoft participates in disputing. The rest of the assumptions are disputed by the reaming contributions of CyPhyCARD, EvoSense or CyberX.

Attacker assumptions:

1. The defense system shares the same network or host with the target of attack/defense system.
2. The system is not capable of being fully situation aware of all its components in a massive-scale network in real time.
3. The defense system management workstations (that the administrators use) share the same network with the target of defense.
4. Defense systems are not resilient against attacks, and have weak recovery mechanisms. [Note: most of them assume that they will not be the target of an attack as long as they were able to secure their ToD. Additionally, usually they have no intrinsic failure recovery.]

ChameleonSoft major contributions that participate in disputing such assumptions are: ChameleonSoft software behavior encryption and trace resistant moving target defense, online autonomous adaptation to changes, and the intrinsic resilience of the building blocks, the enhanced Self and situation awareness of the platform and the Cell itself. These contributions will work against the aforementioned attacker assumptions, or the goals behind such assumptions.

ChameleonSoft software behavior change will work against the goal behind assumptions 1,and 3. The attacker assumed that if the defense applications are sharing the ToD platform or network, it can be affected by attacking it, or it can be utilized to disrupt the operation of the ToD application. The attacker cannot target specific vulnerability in the executing software within the host machines to lunch her attack. It is almost impossible with such runtime dynamic spatiotemporal change, to induce certain changes to static applications working on defense provisioning causing them to fail. Using ChameleonSoft works against that, as the defense

services hosted on a CyberX managed platform and protected by ChameleonSoft will be resilient against any utilization of any available flaws.

The intrinsic smart situational awareness of the platform building blocks, and the platform as a whole, works against assumptions, 1 and 2. The ChameleonSoft and CyberX hierarchal management framework is capable of handling large scale networks and being fully aware of what is really happing within such networks.

ChameleonSoft trace resistant moving target defense and CyberX intrinsic recovery works against assumption four and the goals behind the four assumptions that the attacker was targeting. The main target for the attacker was to fail the defense services. With ChameleonSoft moving target defense it will be hard to target any available entry points for the attacker. Further, with CyberX automated fast recovery; the attacker will not be able to easily fail an application. The failed Cells “hosting application threads” will be recovered from any failures autonomously. Additionally, the Chameleonized recovery system by itself contributes partially realizing the moving target concept. The technique used for recovery intentionally deploys the replacement Cells in different geographical locations, with different platform configuration from the failed Cell to minimize any chances of re-failure. Doing so, moves the attacker target which is the vulnerable application executing within the Cell away from the attacker. Giving this mechanism, it is not even in the attacker benefit to try to fail any of the COA Cells as doing so makes it almost impossible for him to target this Cell again.

3.8 **Conclusion**

In this chapter we presented ChameleonSoft as a moving target defense mechanism against software attacks. The system is built over our novel CyberX-managed Cell-Oriented Architecture.

ChameleonSoft leverages CyberX to apply multidimensional spatiotemporal diversity and hot shuffling of variants, hence effecting software execution behavior encryption. ChameleonSoft relay on CyberX multi-mode, autonomous, situation-aware recovery system to enhance the defense process against coincidental and intentional failures. Further, it adjusts system shuffling and recovery policies at runtime to meet the continual change in the operational environment. There are several interesting challenges to be addressed in the future. These include autonomous detection and profiling of behavior; adjusting shuffling decisions based on that profile; software Chameleonization including formalizing an automated variant generation system, and presenting alternatives for legacy non Chameleonized software.

Chapter 4

Bio-inspired Evolutionary Sensory System for Cyber-Physical System Defense

> When you aim for perfection, you discover it's a moving target" .. Geoffrey Fisher

4.1 **Introduction**

Major physical infrastructure systems such as the water distribution systems and the electric power grid are large-scale complex systems that are expected to be highly reliable and trustworthy. Modern versions of these infrastructure systems go far beyond simple measures to integrate intelligence and automated control into the system through tightly coordinated and integrated cyber components constructing large-scale Cyber-Physical Systems (CPS).

CPS safety and security are prerequisites to assure stability, reliability, and survivability of such mission-critical systems. Defense services for CPS are highly dependent on the promptness and accuracy of the Monitoring and Analysis (M&A) mechanisms employed. Traditional M&A approaches do not treat sensing and effecting for cyber components and physical components seamlessly.

The current M&A mechanisms were designed based on a set of assumptions that unintentionally neglect the real-time interaction and the tight coupling between these converging components. The **assumption** was that physical components were protected by isolation and parameter

defense while real-time response was not a primary factor for cyber components. Further, they assumed that there is no need to employ privacy preservation techniques as the Target of Defense (ToD) privacy is implicitly protected by cyber and physical parameter defense. Additionally, they assumed that resource heterogeneity and scale could still be resolved by a distributed set of heterogeneous, pre-deployed platform-dependent defense tools with fixed resource profiles.

Research works in [83,84] as well as our own have disputed the validity and correctness of such assumptions as they lead to drastic **problems and limitations** negatively impacting the quality and promptness of the CPS defense service provisioning. Current CPS Defense Service Providers (CPS-DSPs) fail to provision trustworthy robust and reliable **monitoring and evaluation** of the ToD components due to the use of scattered, uncoordinated, uncooperative, unaware, isolated and heterogeneous monitoring tools, and reporting mechanisms. Such limitations increase the use of resources due to redundancy, increase the risk of conflicts, and failures due to limited awareness and coordination, lower the defense quality due to the poor, and boundary limited feedback, increase the latency in defense provisioning and in detecting attacks giving the attacker the advantage to spread the attacks through multiple networks, the tool heterogeneity and uncooperative nature massively complicates automating its management, the static nature of such tools complicates attempts to autonomously adapting to changes in the surroundings.

The problem is not only at the monitoring and evaluation phase of the defense provision process, but also at the **analysis and investigation** phase where the collected feedback gets analyzed searching for attack signs and indications. When the network scale grows exponentially, it becomes almost impossible to analyze the feedback from all the sensors, compile that feedback

together, and extract valuable information from it efficiently, and promptly. Additionally, due to the isolation between the monitoring technology and the analysis technology, and the lack of computational power needed to expand the sources of feedback, the current analysis technology does not have enough data to be fully aware of what is globally happening in the network under investigation.

Having most of the conventional analysis mechanisms designed to share the same host/host-network for the sake of protecting their privacy, lead to serious limitations. The limited investigation search space, being easy to be targeted by attackers, Can be used to cause a DOS attack, and cannot cooperate in analyzing feedback or share information with out-of-perimeter nodes are examples of such limitations.

In addition to the presented set of limitations in the field of monitoring and evaluation, and analysis of feedback, **the control phase** has another set of serious limitations too. Control phase represents the stage where the defense system takes actions regarding detected threats face a serious set of limitations.

Control related limitations are mainly the result of lack of cooperation and awareness that limit the defense tools capability to resolve or even contain persistent fast spreading attacks. For example, it is too hard for such uncoordinated, scattered tools to marshal and coordinate task force to hunt down the attacks spreading all over the network or a set of interconnected networks. The reason behind such complexity is the difficulty in autonomously and promptly controls and coordinates both the DSP, and the ToD tools and equipment to block attack access given the current centralized management technology. Further, without appropriate global control, and situational awareness it is too hard to block the source of dynamic fast-changing

remote attacks. Such limitations can be utilized to cause DoS attack by keeping the DSP busy treating infected files and strike more and more files.

This chapter presents an Evolutionary Sensory System (EvoSense), designed to induce a new paradigm for defense service provisioning that intrinsically and comprehensively addresses the aforementioned challenges facing conventional techniques. EvoSense is a biologically-inspired, intrinsically-resilient, intelligent, situation-aware sense and response system to effect biological-immune-system-like defense provisioning. We address ToD heterogeneity and scale by enabling dynamic defense resource elasticity.

EvoSense is designed to separate the main defense provisioning concerns; the tool logic, management and control, delivery mechanism, and physical resources. EvoSense Utilize our smart, biologically inspired, resilient, adaptable, self and situational aware, elastic, and autonomously managed building blocks (the Cell) to construct mobile, dynamic, and runtime-reprogrammable defense carriers to pervasively distribute accurate, trustworthy, and prompt defense services. EvoSense acts as a middle layer between the defense service provider(s) and the ToD creating a uniform defense interface that hides ToD's scale and heterogeneity concerns from control and management.

This uniform representation enables interoperable and cooperative defense. Further, such isolation maintains defense provisioning survivability in case of ToD failure and DoS attacks. Additionally, EvoSense autonomously and dynamically profile ToD hosts and direct defense services based on the host dynamic behavior and attachments. EvoSense shares the same biologically-inspired, intrinsically-resilient, adaptable foundation of the remaining CyPhyCARD pillars, the CyberX managed Cell Oriented Architecture (COA) described in chapter 2. The

COA provides intrinsic dynamic, distributed, resilient resource management and allocation needed to support EvoSense pervasive M&A.

EvoSense manages a vast number of elastic and intelligent containers (Cells) to host/abstract cyber/physical sensing and effecting tools. EvoSense mimics the human blood stream circulation effect by utilizing its adaptable infrastructure to circulate these context-driven, functionally customizable sensor and effector Cells into the ToD body to pervasively monitor, analyze and control the TOD components. EvoSense sensors and effectors are used to execute defense missions provisioned by DSP. A defense mission is a mixture of sensing and effecting tasks involving information gathering, partial analysis, control, and manipulation of the ToD elements.

EvoSense can alternate/mix different defense/control missions from different DSPs to provision defense services to the same ToD in a process called vaccination. The vaccination process involves sharing defense experience and tools between DSPs in terms of abstract missions, and sensing and effecting packages. Vaccines are autonomously checked for privacy violations and maliciousness before utilization or storage. It is exactly like in biological systems where antibodies can be extracted from one immune body to another to create a healthy up-to-date defense community.

EvoSense's main contributions presented in this chapter can be outlined as follows:

- Enable pervasive autonomously managed monitoring and analysis;
- Uniform defense service provisioning for heterogeneously-composed multi-enclave CPS systems;

- Enable trustworthy, interoperable multi-organization cooperative, dynamic, autonomous defense; and
- Facilitate early failure/attack detection and resolution.

4.2 Evolutionary Sensory System (EvoSense)

4.2.1 *The Foundation*

EvoSense is an evolutionary sensory system designed to enable real-time pervasive monitoring and analysis towards autonomous context aware defense service provisioning. EvoSense defense provisioning platform is composed of three main layers, the management layer, sensor and effector abstraction layer, and sensor and effector tools layer as presented in Figure 4.1. The three layers are founded over our CyberX managed COA. The management layer rules are played by a set of organisms composed of Cells. The abstraction layer uses COA Cells to encapsulate attack investigation and resolution tools defined as binary code variants (APIs) constructing a set of platform independent sensing and effecting capsules. EvoSense constructs a biological immune system like, defense environment by circulating generic streams of such capsules into the (Target of Defense) ToD body to induce a blood stream like effect. The following subsection illustrates and provides more technical details about the defense-capsules creation process, and the defense provisioning methodology of EvoSense.

4.2.1.1 EvoSense organisms and Capsules composition

EvoSense leverages the COA ability to abstract, encapsulate, and virtualize heterogeneous physical and logical resources into unified programmable objects “Cells.” Cells are sandboxes internally construct a suitable working environment for heterogeneous tools. Externally, it is

capable of changing its characteristics to work with many targeted architectures. Regardless whether the sensing target was a computer in a network, or a physical sensor with COA-ready digital interface COA Cells will hide these differences from the enclosed sensing/effecting API.

EvoSense uses the middleware (CC-DNA) installed on the ToD host machines to instantiate, deploy, and host sensor and effector Cells. EvoSense sensors and effectors are a set of precompiled APIs with specific sensing or effecting tasks. Sensors and effectors come with a detailed specification file describing the targeted platform, estimated computational Wight, needed libraries to support it, possible conflicts, ... etc

EvoSense defense mission (organism role) is defined using a custom-made programming language used to generate scripts defining the structure, workflow, and the set of tasks for the sensing and effecting organisms. Additionally, it also defines the type of sensors and/or effectors needed to execute that mission.

Organism creation starts when the host CC-DNA receives the logic script. CC-DNA interprets this logic to construct the organism sensor or effector Cells. In COA, resources can easily be acquired when needed. CC-DNA might ask the local or remote logic reservoir for any sensor or effector APIs that are required to execute the designated sensing or effecting mission in case they were not already available on the targeted host.

4.2.2 *EvoSense defense provisioning methodology*

EvoSense defense provisioning includes two main modes, a DSP-guided mode where EvoSense blindly execute predetermined defense missions provided by the DSP; and an evolutionary-mode that involves evolutionary sensing, and effecting. The DSP-guided modes use EvoSense as a

delivery platform that executes certain commands blindly without being involved in the details of the process. EvoSense collects runtime commands and deliver real-time feedback to the DSP. This mode is highly un-scalable and not recommended for large systems. As this is a limited version of our Evolutionary sensory system, we will move forward and illustrate more about the more generalized mode the evolutionary-mode.

4.2.2.1 Overview and initial configuration

Evolutionary sensing aims to detect malicious up-normal behaviors without prior knowledge of that behavior. In both modes, EvoSense maintains minimum level of security by maintaining the normal work mode of the currently-being-used tools on the ToD. EvoSense treats such tools as part of its sensing and effecting arsenal. The Evolutionary sensing process involves analyzing and correlating different information feeds from multiple sources to magnify up-normal behavior deviation identifying possible attack indications. Evolutionary effecting involves utilizing the pervasive control feature of EvoSense to autonomously deploy safe-resolution tools "that doesn't conflict with the running applications," or to contain such attacks within certain perimeter while waiting for administrators to provide clear resolution procedure to execute. The deployment or containment mechanism works based on an intelligent and dynamic profiling mechanism.

The profiling mechanism works on the fact that attacks can be directed attacks working towards certain objective, or undirected attacks that seek maximizing the victim losses. Even for undirected attacks they can be considered as directed attacks at certain levels. For example, at the operating system levels, windows based attacks cannot infect Unix operating hosts, Java based attacks cannot infect C based software packages, ….etc. The working environment of a certain host can significantly limit the type of attacks infecting this host even for undirected attacks. Based on that, we can easily classify attacks into groups based on certain classification protocol.

Such classification can be attributed with a set of parameters determining the likelihood of having an attack under this class/group in one of the hosts within certain boundaries.

EvoSense has a set of pre-deployed manually/automatically generated profiles used to direct the sensor circulation and the basic deployment package for each host. EvoSense adapt such profiles all the time to maximize the efficiency and accuracy of defense provisioning. The dynamic adaptation of profiles adjusts the definitions defining the needed type and density of defense missions within each profile. Each profile is configured to match the host, host network, attached organization or enclave settings.

4.2.2.2 Joining EvoSense Network

Joining an EvoSense-equipped DSP network procedure starts by installing the CC-DNA on the host machines, registering the hosts' physical IPs, hosts configurations, and their security and privacy policies to the DSP host database, and classifying the host based on one of EvoSense Profiles. Usually the host follows the general profile of the enclave/ organization that it belongs to. However, EvoSense can assign a more fine-grained profile to a certain set of hosts within the same network if they are supposed to behave differently at runtime. The configuration of such fine grained profiles can change over time if the behavior of the host or the surrounding changes.

The host configuration profile illustrates all the details regarding the host platform, computational capabilities, the organization /enclave id(s) for that host if any, and any special consideration regarding the applications running on it. The security and privacy policy defines the needed security level, the scope of cooperation, and the type of allowable sharing materials.

Upon registration of a new cyber/physical host, EvoSense is notified to start the initial evaluation of the host to determine the profile that the host will follow, identifying the basic sensor

deployment-package composition-profile. EvoSense interprets the host record in the DSP database to identify the appropriate types of sensors and effectors APIs that match the host configuration-profile. EvoSense will deploy the evaluation-sensors package to initiate the initial checkup to verify that the host the minimum requirements needed to join the DSP network. In case of any problems, EvoSense will autonomously deploy the appropriate effectors to resolve it.

EvoSense frequently change the basic deployment-package by circulating new sensors to replace old ones. The process is guided by the global sensing feedback not only at the host but also through the network, and the defense provisioning profile that the host is following. EvoSense use a grading system to continuously evaluate sensors on each host based on their success to detect up-normal behaviors. In normal situations, at each evaluation-round, one of the new most successful sensors within each profile replaces the least successful senor in the basic deployment-package of the host. The details about sensor circulation are illustrated in subsection 4.2.6.1.

4.2.3 *Evolutionary sensing and effecting framework*

After joining EvoSense Network, the host defense related aspects will be handled by one of EvoSense management units. The management unit works as a part of EvoSense sensing and effecting framework presented in Figure 4.1. EvoSense sensing and effecting framework is classified into three main layers management layer, sensing and effecting abstraction layer, and the defense delivery tools as illustrated in Figure 4.1.

The sensing and effecting tools layer is a set of logical sensing or effecting APIs stored in the local reservoirs. These tools are autonomously abstracted at runtime into uniform sensing and effecting Cells participating in the construction of organisms playing certain defense missions.

EvoSense organisms are anonymously constructed, managed, and controlled at runtime by EvoSense management layer. This layer is responsible for collecting, correlating, and analyzing sensor feedbacks. Additionally, this layer is responsible for taking decisions based on the sensing feedback, previous historical events, and DSP guidelines. Such decisions might involve composing more capable effecting defense missions for resolution or new sensing missions for deeper investigation.

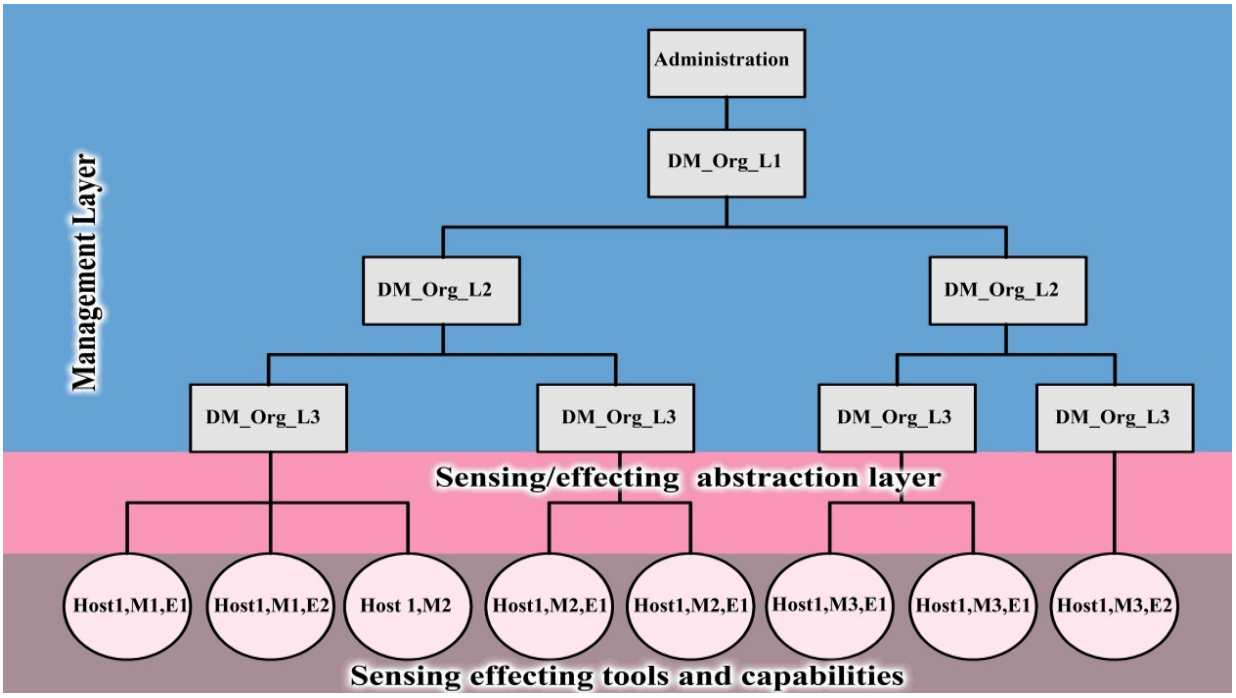


Figure 4.1 EvoSense Abstract View

EvoSense management layer is a tree-like hierarchical construction, where hosts are connected to leaf-Brains "decision making organisms" to be monitored and controlled as presented in Figure 4.1. Based on EvoSense administrator settings, each leaf-Brain manages a specific number of hosts. leaf-brains frequently reports to their parents "Higher-level brains" for more comprehensive guidelines.

4.2.3.1 Feedback management and representation

EvoSense Sensors feedback is a **score-sheet** like report that compares the behavior deviation regarding the sensing target to a predetermined threshold. Sensors are classified into different sets representing their targeted sensing objectives " ex, memory, communications, privacy, ..etc.". Thresholds are dynamically adjusted based on the nature of each host, and the number of false negatives/positives reported by the Sensor.

Score sheets from different sensors for the same host are sent to the leaf-brain to compose comprehensive score-sheets to be checked against the defense rules database. Each defense rule has a score-sheet attached to it. Rule-sheets have values for different objects "ex, memory, communication,.etc" reflecting the behavior patterns "attack signature" of each object in case of infection . Behavior pattern description can be discreet or continuous. Rule description also includes the host sampling procedure. Sampling procedure describes the needed number of samples per object and the duration of each sample, and the sensors needed to takes such samples.

The leaf-brain checks the partial similarity between the sensor feedback "score-sheet" and the existing rules-sheets to allocate the most useful rules for the next deployment round. Based on that selection, the leaf-brain will compose a new organism, holding the list of sensors mentioned in the rule description, with a set of preprogrammed tasks based on the rule sampling procedure. Based on the feedback, threats might be detected, and the resolution mechanism described in the rule will be followed. The leaf-brain will compose a new effector organism with list of effectors mentioned in the rule description and the execution workflow described in the rule.

Rules are under full time update by the parent-brains, and the DSP. Leaf-brain experience is frequently reported to the parent-brains for further guidelines. Parent-brains can construct a more comprehensive view of the whole network by correlating the leaf-brain feedbacks. Such views can magnify certain behavior deviation across the network, which will guide the composition of new defense roles to be executed by the leaf-brains for further investigations.

4.2.3.2 EvoSense information sharing (vaccination):

At the higher levels of the hierarchy "the parent-brains", the collected incident reports with the rules, sensors, and effectors used are archived for sharing. The Clearing-House Organism (CHO) role will be easy in this case, as it will check the ToD privacy policy against sharing of such materials. As described before, the shared materials carry no indications about the specific incident source. It is similar to defense related tips that encourage DSP to apply a specific rule because it might reveal certain threats. The reason that motivated DSP to share such information might be the rule successfulness to detect a threat at one of her ToDs or it is a new rule that was developed with promising results. However, the ToD privacy policy will be checked in case it prophets even that level of information sharing.

The CHO role becomes more significant when the DSP asks cooperating DSPs to provide a solution for a problem she has. In this case, the reported suspicious score-sheet and the sensors used to extract it will be shared with other DSPs. CHO will check that material to make sure that the information enclosed dose not contradict with the ToD privacy policy. CHO rejected authorizations are reported to the administrator to manually override or discard. CHO will also check the DSP feedback regarding such requests, new rules, sensors, and guidelines might be provided as a resolution for the problem. Authorized solutions will be deployed, and rejected

ones will be reported to the administrator for further guidelines. The details and the framework of defense mission sharing will be described later in section 4.2.3.2.

4.2.4 *EvoSense brain Architecture*

Figure 4.2 illustrates EvoSense brain architecture and composition of organisms and the interactions between these organisms to achieve EvoSense goals. I will briefly describe each component and its dedicated task.

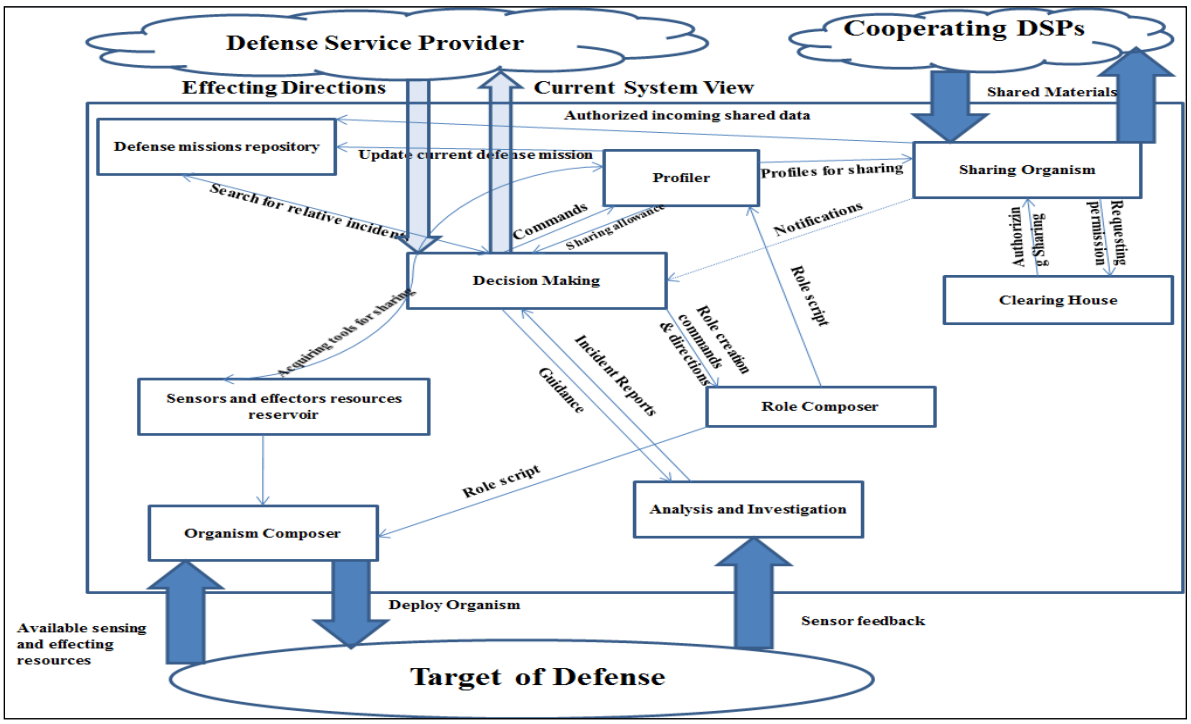


Figure 4.2 EvoSense Architecture

The **Analysis and investigation** organism is responsible of analyzing the continuous feedback from the ToD deployed sensors. The analysis and investigation organism sends reports to the **Decision making organism** to take decisions regarding composing new defense missions, authorizing the use; selecting the type of the effectors if needed. The Role **composer organism** will use the decision making organism guidance to compose new defense roles**.** The role

composition is described using our custom made mission definition language. The role script will be sent to the **organism composer.** The organism composer is a part of the CC-DNA installed on the host. The organism composer organism is responsible for the **resource virtualization process**. It will abstract the host resources to compose the requested organism Cells. These resources might be physical resources "memory, processor, .." or logical resources in the form of conventional defense tools, or any locally stored sensors or effectors. If any of the resources needed to compose the cell did not already exist on the host, the cell composer will acquire these resources from the **Sensor and effector Reservoir**. Upon generating and testing new defense missions the decision making unit might decide to instruct the **Profiler** to generate a profile for the mission with the used tools to be shared with other cooperating DSPs. The sharing process is handled through the sharing organism that will manage sending and receiving shared materials between DSPs. Sharing or employing shared materials has to be authorized by the **Clearing House organism**. The **Defense missions repository organism** will hold history of defense mission usage either locally within the same DSP or globally through feedbacks from other cooperating DSPs with an evaluation for such missions for future reference.

4.2.5 *Information sharing and exchange protocol within EvoSense*

One of the main contributions of EvoSense is the trustworthy information sharing and exchange. EvoSense share attack events and detection/resolution materials between different management organisms locally within the same organization, and globally between cooperation DSP's. Figure 4.3 illustrates EvoSense defense mission sharing protocol.

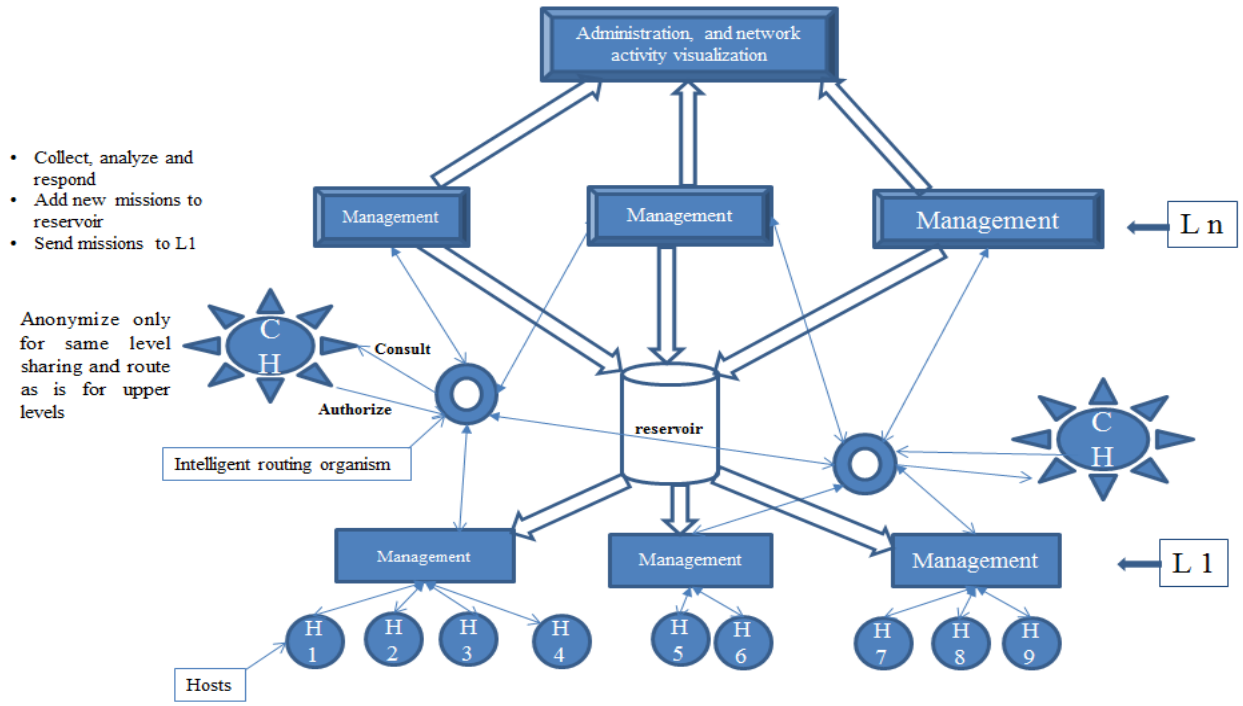


Figure 4.3 EvoSense defense mission sharing protocol

As mentioned before, the defense provisioning process is managed by a hierarchy of management organisms. At the leaf nodes we have the management layer that manages defense mission circulation and execution on the ToD hosts. Such layer collects the sensor feedback, and the detected list of events that was sent to it through the score sheet technique described before. The collected materials are forwarded to one of the distributed routing organisms.

The routing organisms collect the incoming reports and send them identifying the report-source to the higher layer management. Additionally, all reports that was classified as important events by the report source is anonymized and checked for privacy policy violation by the clearing house organism to be forwarded directly to the others local management organisms. The clearing house organism, ask the local broadcasting organism to handle this task. The broadcasting organism will remove any duplicated reports regarding same event, or same defense mission and instruct the leaf management units to raise the score of the missions related to the reported events

based on the severity of the event. Raising the score of a certain mission will increase the chance of applying it in the next mission circulation round.

At the upper level of the management hierarchy, the collected reports from the lower level management units are correlated and analyzed to be presented to the network administrator through visualization software. Using such software will make it easier for the administrator to visualize the attack activity, the defense provisioning process, and to provision manual override or further guideline if needed.

The higher management units will analyze the collected reports providing further guideline to the lower layers. It is also responsible for generating new missions to be stored in the missions repository.

The upper layer management organisms layer provides guidelines to leaf management organisms. The upper management layer organisms can add a new package of sensing or effecting missions to be executed on a specific or a set of ToD hosts at specific time event; or by prioritizing or de-prioritizing “in case of many false positives” one of the missions already being used by this management unit.

DSP level sharing is the responsibility of the higher management layers only. This level of sharing occurs upon the reception of a highly suspicious incoming event. The management organisms mark such event and its relative defense mission for sharing. A dedicated organism named as the sharing organism, anonymizes the shared material, and authorizes it for sharing via the clearing house organism. If the shared materials were not in violation of any of the ToD privacy policies, the shared materials are sent to all cooperating DSP’s either as an alert or asking for a resolution guidelines.

DSP attack alerts should include details about the attack and detection/resolution methodology in terms of defense missions and sensing/effecting tools. If the DSP was asking for an advice related to certain malicious activity, the abstract information related to how such activity is identified and formalized as a sensing only mission, all the tools needed to execute the mission is added to it to be broadcasted to the cooperating DSPs.

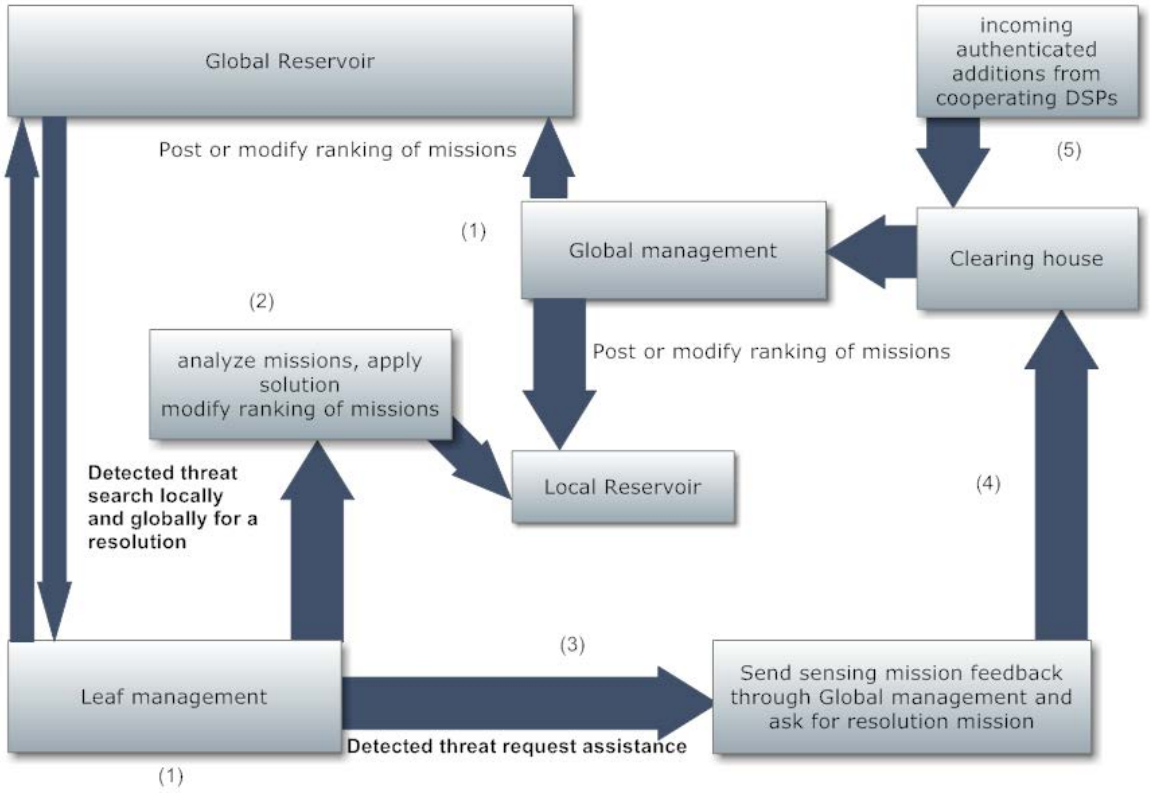


Figure 4.4 The defense mission lifecycle

Upon reception of such missions, the receptor makes sure that there are no privacy violations. The mission is then tested in a controlled environment before applying it to the DSP attached ToD's. If the request was for a resolution guideline, and the receptor DSP has the resolution methodology, the resolution is composed in a defense mission format attaching all the needed sensing and effecting elements to execute it and sent back to the source.

The source test the resolution tools in a controlled environment after making sure that there is no privacy violations then based on the administration opinion it might be deployed to resolve the reported attacks. Figure 4.4 represent the defense mission life Cycle.

4.2.6 *Intelligent attack detection and resolution*

The main objective of any attack detection mechanism is to accurately and properly direct the correct defense tools towards matching attacks. Researchers proved that accurately identifying attacks is an NP-Complete problem [96], while others proved that it might be considered as an NP-hard problem as well [97]. The conclusion is that the problem cannot be solved in realistic time as the problem space expands exponentially over time. The use of Heuristics is always considered to be a good solution for such problems [98].

The most successful malware detection mechanisms uses heuristic scanning and signature based mechanisms to detect attacks. Heuristics scanning in its basic form is an implementation of three metaheuristics mechanism, the pattern matching, automatic learning, and environment emulation. Due to the high computational cost of running such heuristics based techniques, modern anti-malware techniques that are usually shares the same host or the host network of their ToD use a limited set of the available metaheuristics techniques. The reason behind that is to save the computational resources and to speed the process of classifying the executable tasks without interrupting their execution.

EvoSense is designed to work in total isolation of the ToD, and to isolate the main design concerns of malware detection and resolution, sensing, effecting, and control logic. Working in isolation from the ToD enabled EvoSense to use more heuristics techniques and increase the depth of learning and investigation of such techniques without any negative impact on the ToD

performance, or resources. Most of the workload on the analysis and investigation is waved to the DSP platform, and the ToD participates only in hosted sensing and effecting elements for a limited time frame.

As presented in section (C), separating the defense provisioning design concerns enabled EvoSense to optimize the process of sensing and effecting saving more of the ToD resources. EvoSense intelligent sensor reuse mechanism uses the same sensor to feed multiple heuristic techniques, to save a considerable amount of the computational power of the ToD.

The next subsections illustrate the sensor circulation, selection and reuse mechanisms of EvoSense.

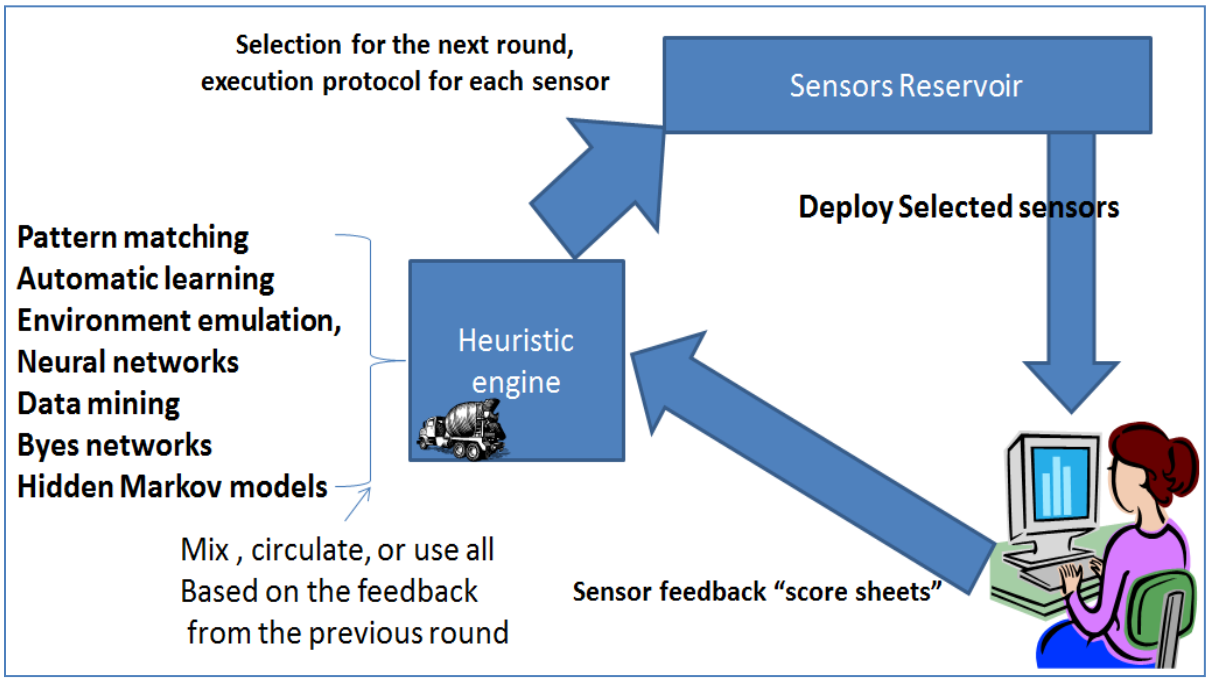


Figure 4.5 Sensor selection and deployment

4.2.6.1 Profile guided Sensor circulation

EvoSense circulates its sensors and effectors to execute defense missions. The sensor circulation protocol depends on the tool requesting sensor deployment. The defense provisioning process involves cooperation between multiple detection and resolution tools. Each tool submits a sensor deployment request to the sensor repository to deploy the requested sensors. The requests pass through optimization unit to remove sensors that were recently deployed before. Figure 4.5 illustrates the sensor selection and deployment procedure.

4.2.6.1.1 Classification based on profiles

As mentioned before, EvoSense attach hosts to certain profiles based on the host engagement with its organization and enclave, host configuration, behavior, usage pattern, ..etc. some of these profiles are static and preloaded with EvoSense management units, and we call them the coarse grained profiles. Such profiles focusses on static classification aspects, like organization id, enclave id, platform configuration , network protocols, ….etc. this profiles determine the general defense provisioning pattern for the host. The second type of profiles is a fine grained dynamic profile that determines the usage behavior of the host and mostly reflects the user behavior using the host. The type of applications being frequently used, the hours of operation are good examples for the aspects controlling such level of profiling. This profile type is dynamically adjusted based on changes on the usage behavior. EvoSense uses host resident sensors to monitor such changes.

The main objective behind using such profiling system is to optimize the utilization of defense provisioning tools by directing only tools with high success ratio.

Attacks are somehow targeted, either from the application-objective perspective, or from the technical perspective. EvoSense circulates defense missions while favoring the activation of defense tools targeting attacks that match the host profile. EvoSense do not limit tool activation to only those who match the host profile to cover any unexpected out of profile attacks. Using profiles to guide tool activation minimize the search space, enhance the detection accuracy and promptness and optimize resource usage on the host and on the DSP.

The detection mechanism relays on multiple control techniques. Signature based technique is used by the resident unit to maintain minimum level of security at all times. The evolutionary sensing mechanism use heuristic technique to guide and control sensor circulation and to analyze sensor feedback to detect unknown attacks, or to identify maliciously acting components.

4.2.6.1.2 Identifying unknown threats

The utilization of heuristic / metaheuristics is necessary to enable attack prediction, and detection of unknown attacks. EvoSense utilizes its ability to isolate the main defense provisioning concerns “sensing, effecting, and control logic” to extract abstract, privacy friendly information regarding running processes. The feedback is safely sent to remote analysis units to apply whatever logic is needed to detect attacks.

The selected logic “metaheuristics technique” determines the type of sensors to be used, execution pattern for such sensors, sample collection protocol. Additionally, that logic is the one responsible for processing the feedback coming for such sensor to determine whether the host is safe or not. The process starts when the selection mechanism selects the metaheuristics technique to be used for the next detection round.

4.2.6.1.3 Metaheuristics selection mechanism

EvoSense can use single metaheuristics technique or multiple metaheuristics technique at the same time to investigate certain issue. The selection of how many techniques to be used and the type of techniques used depend on the severity of the situation under investigation. The default is the use of only one technique, while if the last utilized technique reported high level of attack certainty that is close, but do not cross the required safety threshold, the system use other techniques to enhance the quality of calculation. In EvoSense, metaheuristics techniques are ordered and have weights assigned to each one. The metaheuristics technique selection mechanism always prefers techniques with highest weight value. When an additional mechanism is needed the next highest value technique is selected.

The weights for each technique is not fixed, it is dynamic and gets assigned based on the success or failure of each technique to detect attacks, and the number of false positives or negatives of each technique. This evaluation occurs independently at each management unit within EvoSense framework.

Figure 4.6 presents a simple representation of the metaheuristics technique selection process.

```
(mechanism-id) Select-mechanism( last used, weight, severity level)
(Sensor-list, activation-protocol) Activate (mechanism-id, suspicious-item, item-type)
(deployment-package) Call optimization-unit ((Sensor-list, activation-protocol)
(feedback) Call deploy-sensors (deployment-package)
(res) Process-feedback (feedback, mechanism-id)
If res>threshold
fire-alert (item-type, mechanism-id, deployment-package, feedback)
else
if res < Dynamic(10)% threshold
(mechanism-id) Select-next-mechanism( last used, weight, severity level)
Repeat process
Else
Discard event
```

Figure 4.6 Example of the hubristic mechanism selection procedure

4.2.6.1.4 Sensor reuse

EvoSense optimize ToD resource usage by minimizing the number of deployed sensors and effectors on the host. EvoSense utilize it ability to separate the main concerns and the availability of abstract sensing and effecting elements to remove any duplication of sensor deployment requests. The separation between the tool and the control behind it enabled EvoSense to reuse previous sensor feedback within a certain time frame to feed multiple control components.

For example, let us assume that the selection protocol at time (t) selected pattern matching metaheuristics technique to investigate events occurring on host (h1). The investigations focused on suspicious memory behavior of a certain process. The sensors are selected based on that

objective, and the sampling protocol is automatically generated using the predetermined syntax. The list and the protocol selected are sent to the **optimization unit.** The optimization unit checks if there was any previous valid match for that request. If any, it removes it and report the recorded feedback directly to the analysis unit. If not, the list is sent to the reservoir to program and deploy the sensors. The feedback is sent back to be analyzed by the pattern matching algorithm to determine whether there was an attack or not.

The optimization unit applies certain aging policy to determine the validity duration for a certain senor feedback to be reused. The expiration date is dynamic and adjusted automatically based on the ToD host workload, nature, types of application, level of changes, amount of data flowing from and into it, number of application reinstalling, deployment, .. etc.

If the requested sensor list contain any sensor that was used before with a valid expiration date and compatible deployment protocol “measurement sequence, time, sampling rate, ..etc” then it will not be deployed again, and the old feedback will be reused. Doing so, is expected to save a considerable amount of host resources.

4.3 **Example of CyPhyCARD defense mission**

The task of each EvoSense component is further illustrated through a discussion of one of EvoSense’s automatically composed defense missions that search for memory behavior deviation within a predetermined timeframe, and the dispersion of such deviation through the ToD host networks.

Goals:

- Detect massive deviation in memory usage

- Locate the area under suspicion
- Collect information about processes with suspicious memory usage
- Identify critical and non-critical processes
- Resolve the problem

Tools:

- Memory usage monitoring sensors
- Processes information collection crawler sensors
- Process killing effectors

4.3.1 *Detection and resolution scenario*

This synthetic scenario illustrates the event of using X123 to secure organization *ABC*. *ABC* is a large organization composed of multiple enclaves. The following incident happened in enclave *E1*.

On the regular inspection round on *E1* hosts, with an active heuristic mechanism X, the feedback collected by the deployed sensing organisms and analyzed by the analysis organism indicated an unidentified strange behavior in host *A*. The decision-making organism calculated the weights from the score sheets and followed the heuristic rules and the comparison between the calculated weights exceeds the threshold. The decision-making organism sent its guidelines to the role-composer based on the analysis reports with the list of high similarity rules to the reported feedback score-sheet.

The role-composer composes new defense mission that mix the sensing part of all the similar rules. X123 was the newly generated mission that was built to investigate possible memory-related behavior-deviations within *E1*. X123 have three main roles played by three organisms, sensing, analysis, and effecting. The sensing organism uses the memory scanning Cells to take multiple snapshots of the memory usage within host A for a certain period. The analysis organism applies some predetermined statistics to evaluate the detected behavior deviation that will be evaluated and compared against certain threshold to determine the next step. Based on the result, further investigations might be needed.

These investigations will be handled by the sensing organism that will deploy processes-information-crawler sensor Cells. Crawlers will collect information about process with high memory usage. The collected data will be sent to the analysis organism that will generate a comprehensive report to the decision-making organism. The decision-making organism will decide whether to discard the incident, or to activate effector organisms on X123 to resolve the situation.

The decision-making organism might decide to share the mission profile with other DSPs asking for external feedback, or deploy X123 locally for further investigations. Based on the sharing command search-scope and the clearinghouse permissions X123 will be re-deployed. The deployment scope might be limited to only the hosts within enclave E1, allover ABC enclaves, or globally between DSPs searching for similar behavior deviation pattern.

If the decision was to activate X123 effecting organisms for quick resolution, X123 will be instructed to kill some of the suspicious non-critical processes and re-evaluate the situation.

If the redeployment came-out with multiple incoming alerts for the same memory behavior deviation, the decision-making organism will raise the severity level of the situation indicating global wide spreading attack.

Based on that, commands will be issued to the role composer to customize a new containment mission based on the attack reported parameter. Meanwhile EvoSense will be applying resolution effectors of X123. The containment effectors will be deployed over host A, other hosts in communication with host A, and the intermediate communication elements “routers, switches,..” to construct a quarantine area around the malicious host.

After successful containment and resolution the whole process will be profiled and stored in the defense mission repository for future reference. These profiles will hold details about the containment, and resolution methodology, and all the sensors and effectors API used to compose the used missions. Sharing authorizations and scope of these profiles will depend on the local clearing-house decision.

4.4 **EvoSense role in mitigating the BlackWidow attack**

In this section we intend to discuss the ability of EvoSense to invalidate the attacker assumption on the case study “attack scenario” presented in Chapter 1. The following are the list of the BlackWidow attack designer assumptions; we will list the assumptions and present how EvoSense evolutionary defense system works against such assumptions.

4.4.1 ***Attacker assumptions***

1. The defense system shares the same network or host with the target of attack/defense system. [Note: defense system might be exposed to attack by compromising the ToD]

2. The attack target defense system, or major parts of it, uses COTS security products.[Note: A majority of defense systems are signature based, so that is probably easily to bypass with custom code]

3. The system is not capable of being fully situation aware of all its components in a massive-scale network in real time.

    - Building a very slow motion worm will increase the log file sample size needed to detect it.

    - The attack will spread in small parts in the target network hosted by geographically remote locations. This will make it more difficult to detect attacker activity unless a deep nearly network-scale analysis can be conducted to correlate all disparate logs.

4. The defense system management workstations (that the administrators use) share the same network with the target of defense. [Note: Stolen passwords can simply be used to modify rules of IDS, routers, switches, firewalls, proxies, etc]

5. Attack hosts will not be manipulated in an undetectable way so as not to alert the host AM. These hosts will be used only to launch attack on the primary target. [Note: this might be possible by using zero day exploits and malware code never seen before]

6. Host-based defense systems usually use malware signatures as an indication for infection from various forms of malware.

7. It is not feasible to monitor all the host behavior patterns while sharing the same workstation that is performing user tasks.

8. Defense systems are not resilient against attacks, and have weak recovery mechanisms. [Note: most of them assume that they will not be the target of an attack as long as they were able to secure their ToD. Additionally, usually they have no intrinsic failure recovery]

9. Cyber security is oblivious of and is not coordinated with physical security to protect the target cyber-physical system. Human intervention is need to facilitate such coordination.[Note: the attack can make them conflict with each other to bypass both of them]

4.4.2 *EvoSense addressing attacker assumptions*

EvoSense is designed to work in total isolation from the ToD, invalidating assumptions (1, and 4).

EvoSense is a buffer between the DSP and ToD. Neither EvoSense nor the DSP share the network or the hosts of the ToD. The defense services are delivered to the ToD in a separate network that connects the ToD to EvoSense. The defense delivery vehicles are secured using our moving target defense described in Chapter 3, which invalidates assumption (8).

EvoSense is an active defense system founded over CyberX managed ChameleonSoft secured and resilient foundation. One of the main tasks of ChameleonSoft as presented in Chapter 3 is monitor and secure the COA based foundation against threats and attacks. Having ChameleonSoft and CyberX handling such details waves this workload away from EvoSense giving it more space to focus on provisioning defense services to the ToD. Additionally, EvoSense is designed to support large scale systems and computationally expensive tasks.

EvoSense design supports distributing the tasks over a hierarchy of independent management entities composed of fine grained components managed by CyberX. The fine granularity of such components and the isolation between its logic , data , and physical resources enabled CyberX to fractionize large tasks over multiple hosts constructing a cloud like platform with virtually infinite resources. With that unique feature, EvoSense invalidated assumption (7).

EvoSense uses signature based detection tools as a part of its arsenal, while the major part of that arsenal relay on an evolutionary sensory system. EvoSense evolutionary sensory system utilizes multiple intelligent mechanisms to detect unknown attacks based on monitoring suspicious activities and up normal behavior, and that invalidates assumption (6).

EvoSense is not a commercial product available for conventional users. Even though, the foundation of EvoSense is highly dynamic and autonomous inducing high level of dissimilarity between identical copies of the same system, and that invalidates assumption (2).

One of the main objectives of EvoSense is to promote the defense system situational awareness of the different ToD components and to isolate the platform composition heterogeneity enabling seamless defense provisioning. EvoSense pervasive monitoring and analysis, and the intrinsic trustworthy sharing and cooperative defense enhance the situational awareness all ToD components. EvoSense collect events from different entities of the ToD, correlate the collected information to generate a global image of the entire system to be analyzed by high management units. Doing so, enables EvoSense to detect slow moving attacks, and attacks using remote bots to lunch attacks on remote hosts. The aforementioned aspects successfully invalidate assumptions (3, 5, and 9).

By invalidating all the attacker assumption, it is hard for such attack to succeed in attacking an EvoSense protected system.

4.5 EvoSense detection and resolution model

The attack and cleanup model that we used to evaluate EvoSense performance is in the style of the epidemiological model. This model is being used for years to study the spread of biological diseases [99]. The same model was repeatedly used multiple times in the past few years to model computer networks attacks [100]. Researchers in [101] used this model to develop a more specific model that generated more accurate results when compared to real life attacks [102]. We used their model to construct our experiments. I will present some details about the modified model that we used in our experiments presented in Chapter 5.

4.5.1 *The detection model*

We used an epidemiological based model named the Progressive Susceptible Infections Detection Removal (PSIDR) to construct the experiments presented in Chapter 5. The model focuses on the progress and the dispersion of a contagious attack.

The model is named (PSIDR) or progressive susceptible infections detection removal based on the fact that these four states represent the progress of an attack acting based on this model. The attack progress is a consecutive transition between these states with different rates. Hosts are always in one of four states, susceptible (S) or vulnerable to attacks, which is the case before immunization. Infectious (I) which is the state of a host having an active attack on it. Detected (D), which is the case for an attack being detected and countermeasure being devised. Finally the (R) state which is the case where the attack is resolved or removed from the host, either by

removing the attack and immunizing the host before or after infection. The attack desperation, detection, resolution or containment can be modeled as shown in Figure 4.7.

The general attack process begins when the attack starts targeting victims. These victims might be randomly selected or intentionally selected as a start for a spreading infection. The second step is the dispersion of the infection; the attack clones itself into as many machines as it can. The third step is being detected or reported to the defense service provider. And the last step is the DSP resolving the problem and clearing the infection.

Without any consideration for the evolutionary sensing and effecting the last two steps can take a considerable amount of time. The DSP has to isolate the attack, generate a signature for it, start spreading the signatures all over the network updating the local antivirus database, and device a resolution methodology for the attack.

With evolutionary sensing, the system quickly shares the attack alerts, the sensor used for detection and the detection methodology among all participating hosts in the form of comprehensive set of defense missions. The pervasiveness of EvoSense sensing mechanism has a great impact on the time needed for detection. Further, the system can deploy the evolutionary containment effectors, to minimize the attack dispersion until a resolution methodology is devised.

We assume that EvoSense containment organisms are one of the usable resolution methodologies to mitigate attacks with a certain penalty on the resolution downtime. The mitigation missions are applied to machines independent of their infection state (i.e. to S and I).

We started our experiments based on a PSIDR like model to represent the case of using local none-cooperative defense service provisioning units. EvoSense evolutionary sensing and

effecting where modeled using a set of distributions representing the attack dispersion and resolution rates. The same technique was used to model virus throttling [103].

Virus throttling is a technique to automatically contain the effect of a fast spreading virus while searching for a possible resolution for the confirmed infections.

The PSIDR model calculates the downtime due to infection after determining whether the host was infected or not. If it was not infected then it will be immunized against this attack and it will not be infected with this attack in the future. The downtime is zero at this case. If the host was infected then the downtime is calculated by a penalty for each time slot passed between infection and detection plus the time needed for resolution.

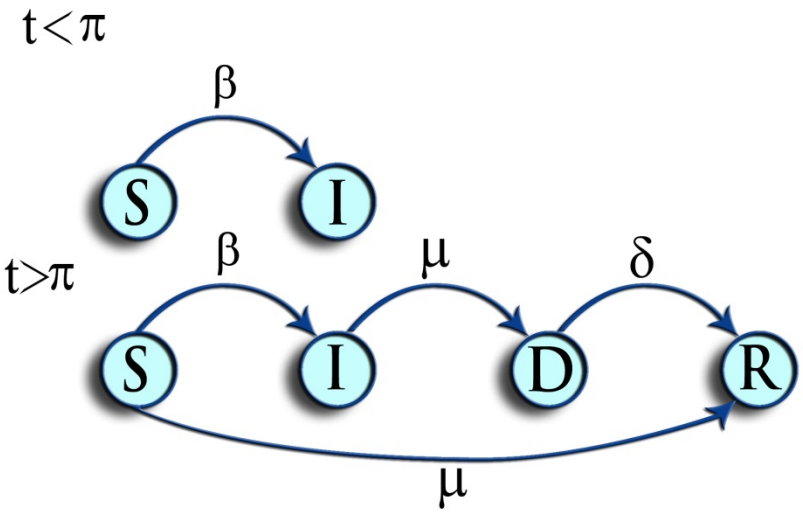


Figure 4.7 The PSIDR model

Through the model all machines are initially susceptible and there is no clear method of detection available to mitigate the attack. At this initial stage the attack can spread freely. The parameter (β) is used to indicate the attack attempts per time slot. The time needed to detect is

modeled by parameter (π). The parameter (μ) is used to indicate the resolution dispersion per time slot, which is the number of hosts having the appropriate effectors to mitigate/contain the attack or to immunize the host against future similar attacks. The parameter (δ) is used to represent the process of applying the effectors to resolve the infection and immunize the host.

The following summarizes the parameters used through this representation of the PSIDR model:

- (μ) time for permanent immunization by updating the local defense tools
- (β) Attack infection rate
- (δ) time for permanent immunization after attack by updating the local defense tools
- ($t< \pi$) Time before resolution mission composition
- ($t> \pi$) Time after resolution mission composition

The authors of the basic PSIDR model [99] calculated the following probabilities and used it to evaluate their model by comparing the extracted results with results calculated from real dispersion of a set of attacks. We used the same probabilities while controlling the aforementioned parameters by a set of distributions to represent EvoSense methodology in detecting and resolving attacks. The used probability distributions were controlled to generate parameters within the recommended range suggested by the PSIDR authors.

$$P(S->I) = \beta I/N$$

$$P(I->D) = \mu$$

$$P(S->R) = \mu$$

$$P(D->R) = \delta$$

Where N is the total number of hosts in the ToD network.

While the model was actually used to model vast number of threats, viruses, and attacks [104,105]. One of the problems raised against this model, was the assumption that the local defense mechanism will not be affected by the attack. This is not true for many attacks, and an attack like [106] is a good example for such cases. Such attack breaks the link between I-> D link.

Fortunately, EvoSense was designed to prohibit such cases making it a perfect choice to model EvoSense's defense provisioning process by modeling the attack dispersion and cleanup process. EvoSense was designed specifically to remain operational even with the existence of aggressive attacks that targets the DSP or the DSP network.

We used the model to measure the total time for detection, and the time needed for resolution after detection. With EvoSense pervasive cooperative sensing, the time needed for detection was much less than the normal case of using local defense provisioning units. Hosts move faster between I -> D and conventionally R states; which means that the overall downtime due to attack is also minimized using EvoSense evolutionary sensing and effecting.

EvoSense pervasive control of the ToD hosts and host network, allows the DSP to quickly, and in a full/simi-autonomous fashion contain a contagious attack. In addition to minimizing the attack negative impact by such containment, EvoSense can restore operation of an infected host until a resolution methodology is devised by allowing it to work in a controlled quarantine mode.

In such mode, EvoSense utilize its pervasive control feature through the ToD components "hosts, and network elements" to apply deeps scanning missions to all outgoing message coming from the infected host before sending it to the ToD network.

From that we can see that EvoSense protected ToD's hosts are expected to move faster from S or I to R.

4.6 **Conclusion**

In this chapter, we presented EvoSense as an evolutionary sensory system enabling pervasive and efficient, monitoring and analysis for heterogeneously composed targets like CPS. EvoSense is built upon our novel CyberX-managed Cell-Oriented Architecture. EvoSense acts as a trustworthy elastic intelligent middle layer interfacing between DSPs and ToDs. EvoSense unique construction and functionality enables pervasive seamless monitoring and analysis; interoperable and dynamic defense; early failure/attack detection and resolution; and trustworthy scalable cooperative defense. There are several interesting challenges to be addressed in the future. These include formalizing the runtime dynamic sensing and effecting autonomous management and mission generation framework; correlating sensors' feedback into comprehensive real-time global views; and adjusting sensors circulation based on dynamic changes of these views.

Chapter 5

CyPhyCARD Evaluation

> "Theory guides; Experiment decides." Izaak M. Kolthoff

5.1 **Overview**

CyPhyCARD utilizes unique defense provisioning platform founded over a loosely coupled dynamic components that host and deliver defense services to the ToD hosts. CyPhyCARD defense platform composition of CyberX-managed adaptive and resilient building blocks comes with a cost. The cost is expected to increase when ChameleonSoft software behavior encryption and trace resistant moving-target defense is used to secure CyPhyCARD infrastructure against software attacks. CyPhyCARD is designed to provision a ToD isolated defense services through EvoSense. Such isolation moves most of the workload from the ToD to the DSP side increasing the platform computational cost even more.

As mentioned before, CyPhyCARD pillars are interrelated contributions providing solutions to serious hard problems in the field of cyber and cyber physical security. In this chapter, we conduct detailed quantitative study including deep analytical analysis followed by multiple simulation experiments to evaluate the effectiveness of each pillar in achieving its design objectives, and the efficiency of the pillar in terms of added execution-time delays and the overall consumed resources. The study illustrates that CyPhyCARD pillars do achieve their

design objective adequately and with reasonable cost. The computational cost in terms of resources consumed to maintain platform resilience against attacks and failures, are justified when compared to the expected losses due to such failure and attacks as shown in the simulation results. The comprehensive evaluation of the entire framework of CyPhyCARD is not possible at this stage. This comprehensive evaluation will be a part of our future work after completing the construction of CyPhyCARD platform full test-bed.

CyPhyCARD platform was not designed to provision defense services for small scale networks or enterprises as the cost of realizing such platform would exceed the expected losses due to attacks and failures. However, we intended to develop a special version of our CC-DNA to be used over the commercially available clouds to enable such clouds of hosting the CyberX Cells. Doing so is expected to decrease the cost of building a dedicated CyPhyCARD cloud, and the platform would rely on its intrinsic defense mechanism "ChameleonSoft" to insure the resilience of defense provisioning and to maintain the privacy of the ToD hosts. However, that might not be sufficient for certain organizations that would prefer an isolated dedicated cloud regardless of the cost to maintain their privacy policy. The cost is application and ToD relevant, it depends on the type of the application, the criticality levels and scale of the network(s), and the privacy rules that needs to be enforced.

In order to evaluate the different performance and security aspects of CyPhyCARD pillars we devised multiple models as a base for a set of simulation packages. Based on the fact that the pillars are interrelated, and in order to increase the resolution of the study by clarifying the cost of enabling each single feature by itself and when combined together, we started by modeling the platform managed by CyberX alone to evaluate the cost of enabling mission oriented application design, dynamic application adaptation to changes, and enhanced resilience using CyberX

recovery. Then we studied the cost of securing this platform by ChameleonSoft multidimensional software behavior encryption and moving target defense. The study included multiple experiments to clarify the cost of enabling ChameleonSoft moving-target defense alone without CyberX added cost for recovery, and with the various recovery modes. Finally, we conducted a detailed study to evaluate the cost of using the presented secure platform to pervasively deliver defense services to the ToD hosts, and the cost of enabling trustworthy tipping and cuing. Due to the tight interrelation between the first two pillars CyberX and ChameleonSoft and the third one EvoSense, the study did not include an evaluation of provisioning defense services through an insecure, failure-vulnerable platform without ChameleonSoft, or without CyberX dynamic recovery, as these assumptions contradicts with CyPhyCARD design invariants and cannot be considered as a reasonable case for study.

The following subsections presents the aforementioned studies including system models, mathematical analysis and a set of experiments conducted using multiple MATLAB simulation packages. The results focused on testing the presented approach ability to achieve its design objectives effectively and efficiently.

5.1.1 *The simulator design*

In order to evaluate the effectiveness and efficiency of the platform and the supporting pillars, we built a prototype in C#. The prototype included a limited version of CyberX, ChameleonSoft, and EvoSense management platforms. The prototype proved the ability of CyPhyCARD pillars to enable Cell life adaptation to changes, automated failure recovery, the spatiotemporal diversification of software execution behavior. Additionally, we devised a simple vision of EvoSense sensors and effectors Cells. We managed to device a simple version of EvoSense

managing a pervasive circulation of such Cells to collect information and to apply remote configurations based on a preconfigured set of defense missions.

For the quantitative evaluation we devised a set of event-driven MATLAB simulator to simulate a hypothetical Cell network composed of multiple Cells distributed in random locations. The simulator also simulated a hypothetical host network distributed in random locations, and classified into different enclaves and organizations. The simulator is designed to mimic the actions of a real system founded over CyberX managed Cell network, secured by ChameleonSoft spatiotemporal moving target defense. Additionally, defense provisioning is simulated by an emulated environment mimicking the remote defense provisioning of EvoSense pervasive and cooperative defense. The system has multiple event generators as threads, working simultaneously to generate different events including but not limited to (attacks, failure inducing changes, Cell failure events, host failure events). These events are generated based on real-time changing settings that determine the frequency of generating each event at each point through the Cell and host networks. The main parameters controlling such stings are guided by a set of random distributions selected to mimic the nature of each event. Upon arrival of each event, the system responds automatically simulating the action of the actual network.

5.2 **CyPhyCARD Platform**

In this section, we present the results of multiple experiments that were performed using the first MATLAB simulator package simulating CyberX. These experiments have different objectives regarding evaluating the effect of enabling CyPhyCARD platform's autonomic adaptation and intrinsic failure recovery managed by CyberX on the system performance with respect to failure downtime and the amount of consumed resources. The simulator was designed

based on an analytical study of the recovery and dynamic adaptation process of the platform. The next subsection describes in details our analytical study followed by detailed description of our simulator and the extracted results.

5.2.1 *A study of CyberX dynamic adaptation*

One of the main advantages of CyberX is its ability to utilize the Cell capability to separate the main design concerns to enable application runtime adaptation. Adaptation may occur for multiple reasons and can be utilized to satisfy different quality attributes.

Figure 5.1 represents a model for CyberX dynamic adaptation process. The model guides the analytical study presented later in this section.

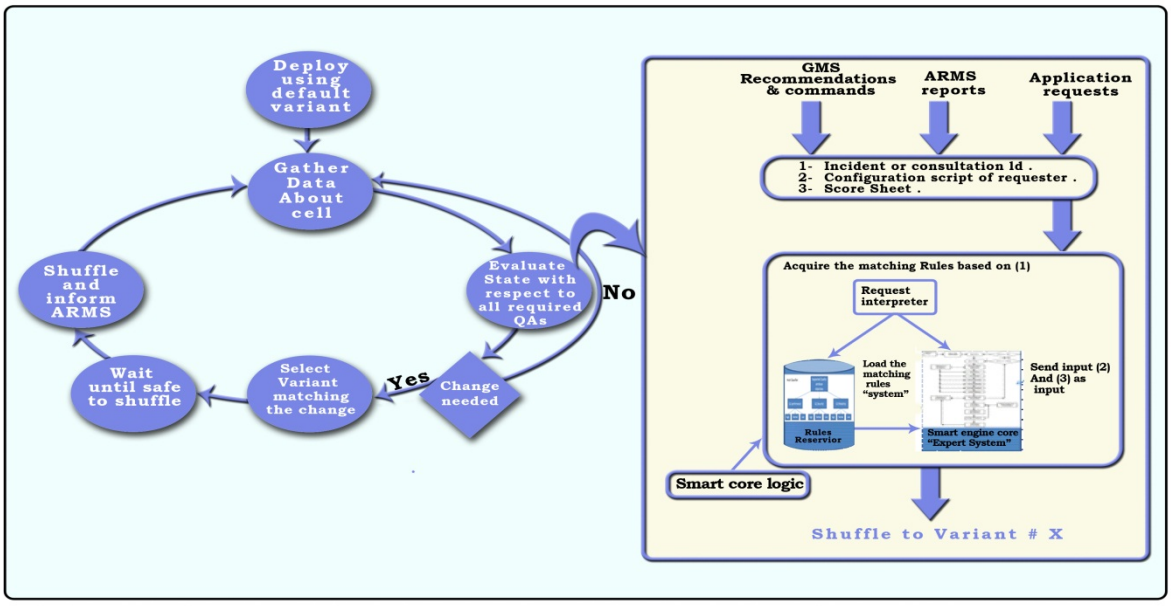


Figure 5.1 CyberX dynamic adaptation Model.

In order to device the mathematical representation of CyberX adaptation process, we will assume that the input behavior is a 2D matrix (***n, m***) where each point in the matrix represents a

Cell ***(i,j***) as an entry in the (***n, m***) plane. Each (***i, j***) entry in the (***n, m***) plan has a pair of values (***v, s***) representing the id of the currently executing variant and set.

The adaptation process is composed of multiple tasks. The formalization of the variant/set shuffling process involves multiple decisions that will be taken based on a set of distributions. The distributions and the random selection mechanisms presented here are for the illustration purposes only. We intend to present a more focused study aiming to select the most appropriate configuration, selection mechanisms, and set of distributions that can generate results closer to the real system. The details of this will be the focus of our future work.

The adaptation process is concerned with manipulating ***V*** using a predetermined distribution. We will use Poisson distribution F_P to determine the time frame between consecutive shuffling events. At the event time, another distribution will be used to represent the variant selection mechanism. We will use uniform distribution F_U to determine the new value of ***V***,

V € {0,1,…a}, S € {0,1,…b},I € {0,1,…n}, J € {0,1,…m}

$$\Delta t = F_p(q),\ \Delta t \neq 0$$

$$t_{x+1} = \Delta t + t_x$$

Where$F_P$is the function that we use to generate the distribution controlling ***t***.

Assuming that***FP*** is a Poisson distribution, it will be calculated as follows:

$$f(k;\lambda) = \frac{\lambda^k e^{-\lambda}}{k!},$$

(***k*** being a non-negative integer, ***k*** = 0, 1, 2, ...)

Δ***t*** is the time interval between shuffling events, each Cell can determine the value of ***q*** controlling the shuffling frequency

At each shuffle event, each Cell uses the following equation to select the next variant for execution.

$$v_{tx}=f_u(z),\ v_{tx}\neq v_{tx-1},\ v_{tx}<a,\ z \in \{0,1,\ldots a\}$$

Where f_u is the function used to generate the random number generator determining the id of the next variant for execution selected from the variant selection pool at each Cell, and *z* is the seed for the generation.

Assuming $F_u$is a uniform distribution it will be calculated using the following

$$f(x) = \begin{cases} \frac{1}{b-a} & \text{for } a \leq x \leq b, \\ 0 & \text{for } x < a \text{ or } x > b \end{cases}$$

Where ***a*** and ***b*** are the minimum and maximum number of valid values for ***V***

5.2.2 *A study of CyberX automated recovery*

Another advantage of CyberX is its enforced resilience through an automated Recovery system.

Figure 5.2 represents a model for CyberX automated recovery process. The model guided the analytical study presented in this section.

The recovery process is concerned with manipulating (i, j) location for each Cell in the matrix. We will use Poisson distribution FP to determine the time frame between consecutive failures events. At each event ***t*** the Cell follows a Uniform distribution to determine the new location $(i,j)_{new}$ that the Cell will migrate to after recovery.

$$\Delta t=f_p\ (q),\ \Delta t\neq 0,\ q=>0$$

$$t_{x+1}=\Delta t+t_x$$

Where F_p is the function that we use to generate the distribution controlling ***t.*** Δt is the time interval between failure events, each Cell can determine the value of ***q*** controlling the failure rate.

$$i_{x+1}=f_{in}(z),$$

$$j_{x+1}=f_{jn}(z)$$

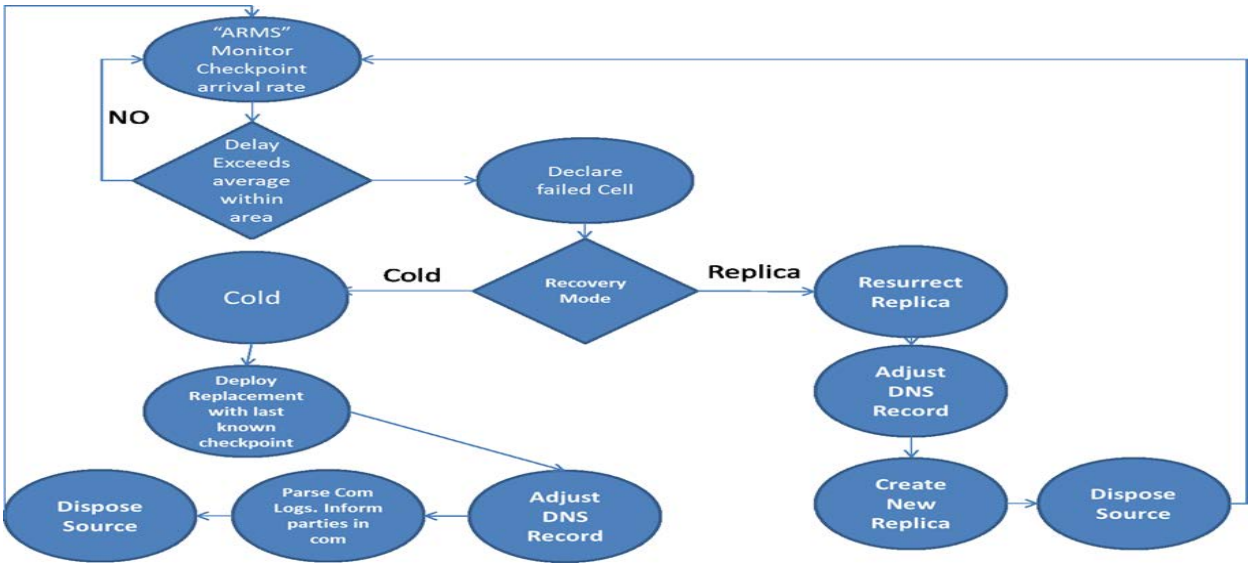


Figure 5.2 Represents CyberX automated recovery process model

Where F_n is the function used to generate a new location for the Cell to migrate-to in the (***n,m***) plane. And **z** is a random seed set to insure that the output range of ***i***, and ***j*** ranges from 0 to (***n,m***) respectively

Assuming that F_n will be a normal distribution it will be calculated as follows

$$f(x) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$

5.3 Simulation results

In this section, we present the results of multiple experiments that were performed using a MATLAB based simulator. These experiments have different objectives regarding evaluating the

effect of enabling autonomic adaptation and intrinsic failure recovery on the system performance with respect to failure downtime and resource consumption.

Table 5.1 shows the main parameters used in the simulation. The network parameters are mainly static parameters used to setup the experiments, except for the deployment of fresh Cells in the network. The dynamic part depends on a set of distributions mentioned in the column named “Generator “.

The failure or environment-change parameters show the spatiotemporal distribution of failure /environment-change events and the event type that necessities variant change in response to such event. The recovery parameters represent the initial recovery mode for each Cell, and the dynamic recovery change through the experiment lifetime. Deploy-new-Cell parameters represent the rate and location for the deployment of fresh Cells to replace dead or problematic Cells in the network. All experiments had the same period of 6 hours with a sample rate of six minutes giving us 60 samples (time slots) within the network of Cells. The presented parameters in (Run 1) were used to device Figure 5.3, and Figure 5.4 We used the parameters in the three runs to evaluate the effect of increasing the failure generation rate illustrated in Figure 5.4.

Classification	**Parameter**		**Generator**	**Run1**	**Run2**	**Run3**
Network	Network size		Static	100*10	100*10	100*10
	# shuffling variants		Static	8	8	8
	Exp_Time		Static	60	60	60
	Avg_App_exe_time		normal	35	35	35
	Deploy new Cell	Period	Poisson	23	18	14
		Location	normal	8,3 91,2	8,3 91,2	8,3 91,2

	Resource usage	Cell	Static	5	5	5
		For Replica	Static	3	3	3
		Cell failure	static	2	2	2
Recovery	Recovery at deploy		normal	8,3	8,3	8,3
	Mode change	Period	*Poisson*	20	18	16
		Type	normal	8,3	8,3	8,3
Event	Failure or environment event change	Timing (Period)	*Poisson*	24	20	16
		Location	Normal	8,3 91,2	8,3 91,2	8,3 91,2
		Type	Uniform	10	10	10

Table 5.1 CyberX simulator parameters

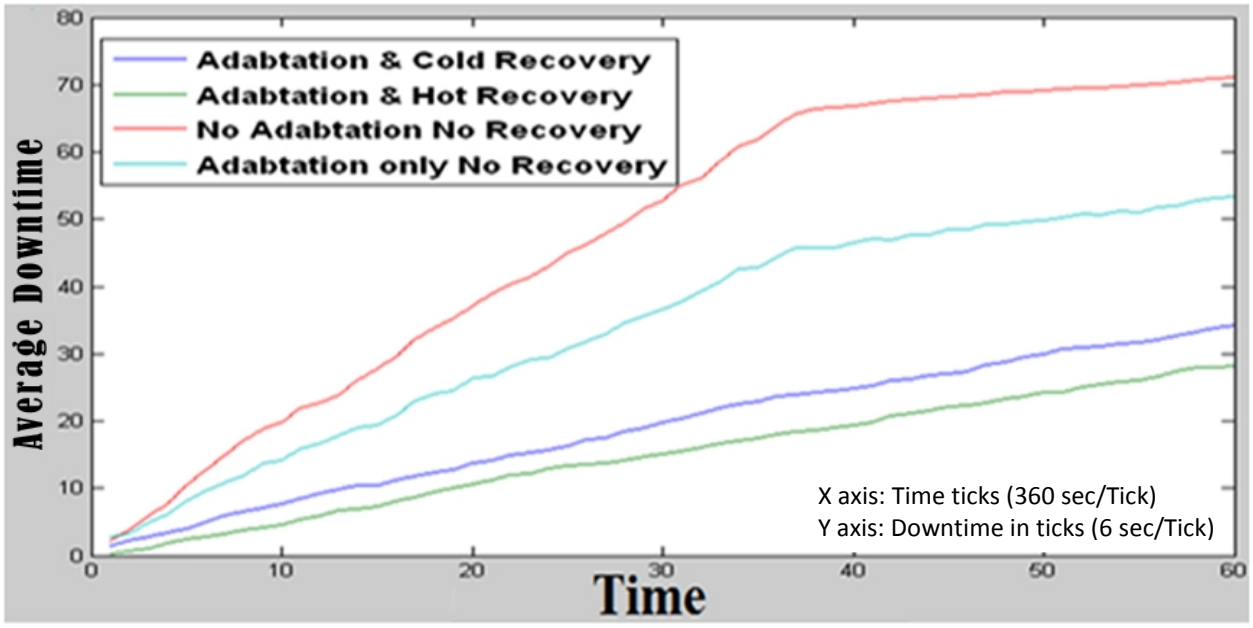


Figure 5.3 The average downtime in response to failures due to changes for different recovery modes with and without adaptation.

Figure 5.3 illustrates the effect of failure due to unexpected changes on the average downtime with and without CyberX autonomic adaptation. The average downtime is calculated as follows:

Average downtime per Cell = $(\sum_1^N Time\ spent\ while\ Cell\ is\ offline)/N$

Figure 5.3 reflects the different system responses to failures when we activate and deactivate CyberX autonomic adaptation with and without coarse or fine-grained recovery modes. The experiment shows significant improvement in minimizing the failure downtime when the CyberX autonomic adaptation is active as the system adapts autonomously to most of these changes minimizing the number of failures. Additionally, the average downtime significantly decreases when we activate CyberX fine or coarse grained recovery. Both recovery modes will rabidly recover failed Cells minimizing the overall failure downtime.

Figure 5.4 presents the effect of increasing the failure generation rate by increasing the number of changes over time "in an extended execution time mode" on the average downtime while utilizing coarse or fine-grained recovery modes. The experiment shows that CyberX fine grained recovery always minimizes the failure downtime when compared to coarse grained recovery. In coarse grained recovery mode, CyberX spends more time instantiating replacement Cells; while in fine grained mode, replicas take over and resume execution first then a new replica is instantiated without holding the execution restoration.

Figure 5.5 illustrates that fast recovery comes on the expenses of consuming more resources. This figure reflects the total resource usage through the experiment with different recovery and adaptation modes.

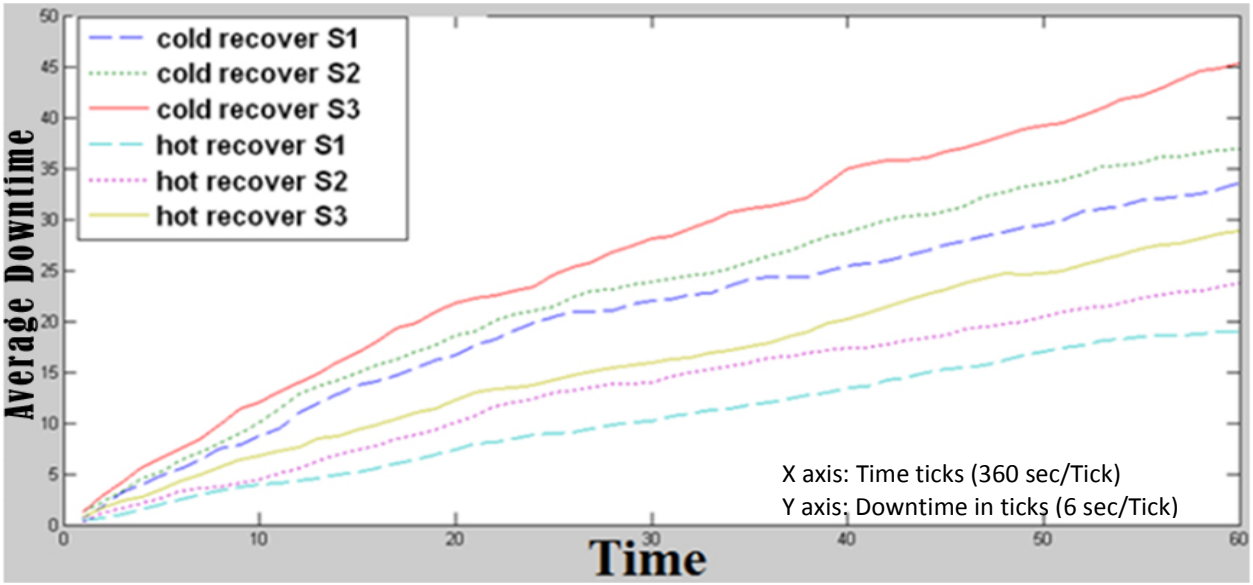


Figure 5.4 The average downtime in response to increasing failure generation rate for three different experiments and different recovery modes.

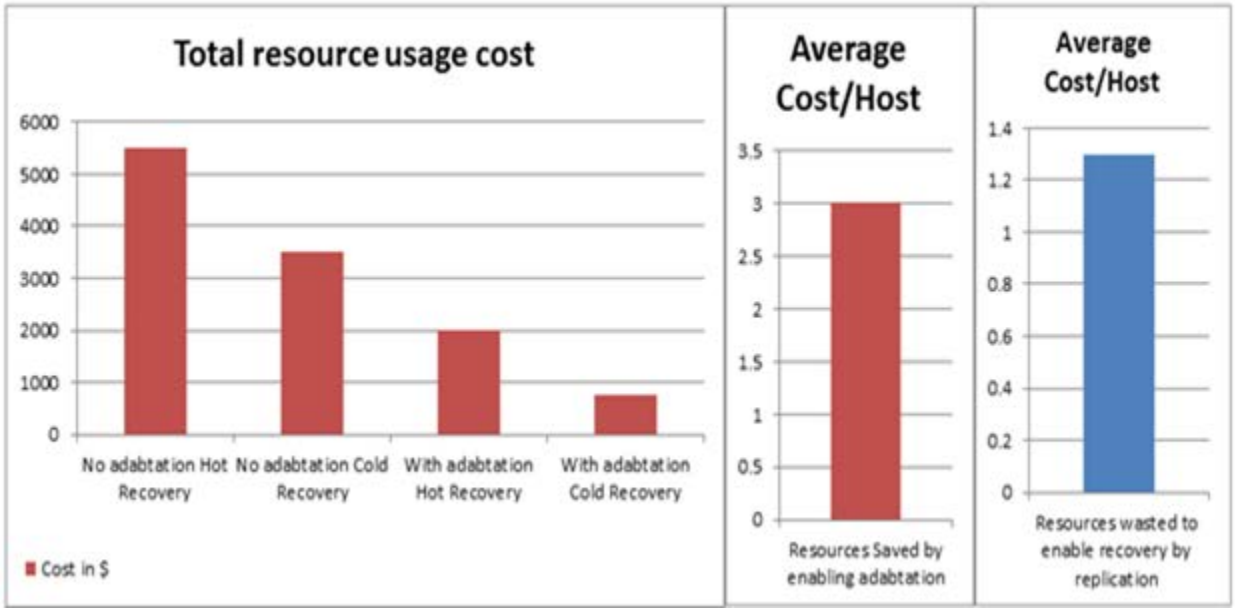


Figure 5.5 The total resource usage in case of failure with different recovery and adaptation modes

Figure 5.5 illustrates the effect of using CyberX autonomous adaptation to minimize failures, in saving some of the resource that would have been wasted in the recovery of such failures. The total cost of resources is calculated in money value as follows:

Total cost at each Cell = $\sum_{1}^{n}$ cost of peripheral usage in \$ $*$ consumed $value$

$$\text{Total Resource usage in \$} = \sum_{1}^{H} \text{overall cost of consumed resources at each host}$$

Enabling such feature provides some guarantees that the system will always consider using the right resources at the right time while maximizing the system quality-attribute satisfaction-scope. Further, CyberX attempts to recompense resources wasted due to failure-recovery by changing the system targeted quality-attribute towards optimizing the resource usage after each recovery event. CyberX usually favor using one of the resource efficient variants to resume execution after each recovery event. CyberX do that while maintaining the balance between the different application objectives and targeted quality-attributes to the best interest of the application while efficiently maximizing resource utilization.

5.3.1 ***Observations***

From the presented results we can conclude by illustrating the following list of observations:

- CyberX dynamic real time application adaptation to changes decreases the chance of failure, reduces the system downtime and wasted resources.
- CyberX multimodal failure recovery enhances the application resilience against failures. The effect was reflected in the noticeable decrease in the average Cell downtime.

- CyberX multimodal failure recovery increases the average host resource consumption; however CyberX was apple to compensate that by enabling runtime dynamic adaptation. Runtime dynamic adaptation saves even more resources that should have been wasted due to failures.

5.4 **A moving-target defense approach for CyPhyCARD platform security**

In this section we focus on evaluating the cost of securing CyPhyCARD platform using ChameleonSoft software Behavior Encryption (CBE) and moving target defense. We used analysis and simulation to evaluate the security and performance of ChameleonSoft. A comprehensive analytical study of the CBE is conducted to formalize the spatiotemporal diffusion and confusion processes. The study was the base for building CyberX-based CBE MATLAB simulator that we used to extract the presented results in section 6.2.

5.4.1 ***Analyzing the CBE approach***

In order to device the mathematical representation of the CBE process, we will assume that the input behavior is a 2D matrix (***n, m***) where each point in the matrix represents a Cell ***(i,j***) as an entry in the (***n, m***) plane. Each (***i, j***) entry in the (***n, m***) plan has a pair of values (***v, s***) representing the id of the currently executing variant and set.

The encryption process is composed of multiple different processes. Temporal shuffling and spatial shuffling occur separately, or combined together to form spatiotemporal confusion, and diffusion. The formalization of the confusion and diffusion processes involves multiple decisions that will be taken based on a set of distributions. The distributions and the random selection

mechanisms presented here are for the illustration purposes only. As a part of our future work, we intend to present a more focused study aiming to select the most appropriate configuration, selection mechanisms, and set of distributions that can generate results closer to the real system.

CBE temporal confusion: this process is concerned with manipulating ***V*** using a predetermined distribution. We will use Poisson distribution F_P to determine the time frame between consecutive shuffling events. At the event time, another distribution will be used to represent the variant selection mechanism. We will use uniform distribution F_U to determine the new value of ***V***,

V € {0,1,…a}, S € {0,1,…b},I € {0,1,…n}, J € {0,1,…m}

$$\Delta t = F_p(q),\ \Delta t \neq 0$$

$$t_{x+1} = \Delta t + t_x$$

Where F_P is the function that we use to generate the distribution controlling ***t***.

Assuming that***FP*** is a Poisson distribution, it will be calculated as follows:

$$f(k;\lambda) = \frac{\lambda^k e^{-\lambda}}{k!},$$

(***k*** being a non-negative integer, ***k*** = 0, 1, 2, ...)

Δ***t*** is the time interval between shuffling events, each Cell can determine the value of ***q*** controlling the shuffling frequency

At each shuffle event, each Cell uses the following equation to select the next variant for execution.

$$v_{tx} = f_u(z),\ v_{tx} \neq v_{tx-1},\ v_{tx} < a,\ z \in \{0,1,\ldots a\}$$

Where f_u is the function used to generate the random number generator determining the id of the next variant for execution selected from the variant selection pool at each Cell, and *z* is the seed for the generation.

Assuming F_u is a uniform distribution it will be calculated using the following

$$f(x) = \begin{cases} \frac{1}{b-a} & \text{for } a \leq x \leq b, \\ 0 & \text{for } x < a \text{ or } x > b \end{cases}$$

Where ***a*** and ***b*** are the minimum and maximum number of valid values for ***V***

Spatial confusion is concerned with manipulating (***i, j***) location for each Cell in the matrix. We will use Poisson distribution to calculate the time frame between two consecutive spatial shuffling events. At each event ***t*** the Cell follows a Uniform distribution to determine the new location $(i,j)_{new}$ that the Cell will migrate to.

$$\Delta t = f_p(q),\ \Delta t \neq 0,\ q => 0$$

$$t_{x+1} = \Delta t + t_x$$

Where F_p is the function that we use to generate the distribution controlling ***t.*** Δt is the time interval between shuffling events, each Cell can determine the value of ***q*** controlling the shuffling frequency

$$i_{x+1} = f_{in}(z),$$

$$j_{x+1} = f_{jn}(z)$$

Where F_n is the function used to generate a new location for the Cell to migrate-to in the (***n,m***) plane and **z** is a random seed set to insure that the output range of ***i***, and ***j*** ranges from 0 to (***n,m***) respectively

Assuming that F_n will be a normal distribution it will be calculated as follows

$$f(x) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$

Spatiotemporal confusion is a mixture of both

$$\text{Ⱥ}(i,j)_{tn} \begin{cases} \begin{cases} t1_{n+1} = \Delta t1 + t1_n \\ i_{n+1}, j_{n+1} = f_{in}(z), f_{jn}(z) \mid t = t1, H_{n+1} \neq H_n \end{cases} & \text{Spatial shuffling} \\ \begin{cases} t2_{n+1} = \Delta t2 + t2_n \\ v_n = f_u, \mid t = t2 \end{cases} & \text{Temporal shuffling} \end{cases}$$

Diffusion is induced in response to each incoming event by making a random change for the value of v in multiple locations (i,j).

In order to simulate that we use a predetermined distribution to guide the selection pool p for the Cells that will make changes in their current active V.

Each Cell (i,j) € p will apply to select the id of next variant to execute V_{tx+1}.

Time between generated events at each Cell (I,J) is calculated as follows:

$$\Delta t = f_p(q), \Delta t \neq 0, q => 0$$
$$t_{x+1} = \Delta t + t_x$$

where, q is event generation frequency

The type of event is calculated as follows:

$$e_{tx}(i,j) = f_u(z), e_{tx}(i,j) \neq e_{tx-1}(i,j), \ 0<z<b$$

Where Δt is the time interval between surrounding change/attack events at each Cell, based on that event s will be changed to match the e_{tx}, and V will reset to 1

Selection pool p is constructed using the following equation:

$$(i_{dif},j_{dif})=\{f_u(a,b)|i_{dif}\neq i, j_{dif}\neq j\} \rightarrow \text{Cell } (\boldsymbol{i,j})$$

with an ongoing event and ***a***, and ***b*** are random numbers >0 and <n,m

Where $\mathbf{f_u}$ is the function used to generate a pool of random selections for (***i,j***) Cells that will change their current ***V*** using $\boldsymbol{f_k}$

$$\forall(i,j)\in p,\ v(i,j)=f_k(z),\ v_x\neq v_{x-1} \rightarrow \text{the situation permitting the change; where } 0<z<a$$

Assuming the diffusion and confusion function is calculated based on the presented list of functions ***f*** at each **i,j** with a $\Delta\boldsymbol{t}$ as the duration between events then the encryption key controlling the diffusion and confusion is a combination of the generation functions ***f*** and the number of Cells in the plan with the variant and set pool size.

In this case the key will be:

$$k(i,j)=\{fp(q), fu(z), fn(z), fu(a,b), fk(z)\} \rightarrow v_0^a, s_0^b$$

If we include event generation then we should add ***fep (q), feu(z),***where ***fep (q), feu(z),***is the functions used to estimate the time and the type of an incoming change in the surroundings of each Cell necessitating a set change; then the key will be:

$$k(i,j)=\{fp(q), fu(z), fn(z), fu(a,b), fk(z), fep(q), feu(z)\} \rightarrow v_0^a, s_0^b$$

5.4.2 *Simulation results*

We designed a MATLAB simulator to mimic the Chameleonization process of a group of Cells organized in a 10*100 matrix layout. Version 1 of the simulator (V1) that was presented in [95] was a preliminary version with limited capabilities. V1 was designed to extract preliminary results illustrating the effect of the confusion and diffusion process.

We have developed a more advanced version of the simulator (V2) addressing the confusion and the diffusion processes more accurately. In V2, the diffusion induction in response to a single event in any part of the network starts by broadcasting multiple shuffling for diffusion requests within a predetermined scope to different Cells. Based on predetermined acceptance criteria, Cells will comply with the incoming requests and change their current active variant. Additionally, V1 was mainly concerned with the temporal shuffling; while V2 is designed to simulate both the temporal and special shuffling.

V2 is equipped with multiple performance monitors that will continuously monitor and records the performance aspects of the Cells within the life time of the experiment. The performance monitor feedback will be used to evaluate the effect of temporal or spatial confusion and diffusion on the task completion time presented in the next subsection.

Additionally we integrated a module to simulate the effect of the CBE multimodal recovery mechanism and the dynamic real-time recovery mode change. For evaluation purposes random failure events are distributed based on predetermined criteria to induce the effect of multiple failures. The system will automatically respond to these event based on the current recovery technique at the point of failure. The performance monitors will record the failure event, and the failure downtime at the point of failure.

V2 of the simulator is designed to simulate CBE effect for large network of Cells, with longer experiment time. The simulator is capable of generating different configurations for the Cells. Cells can have different application execution time and requirements and each Cell can change these requirements at runtime based on a predetermined criteria. The simulator will be activating and deactivating Cells at runtime in response to Cell termination events due to failure or execution completion, and the deployment process of new fresh Cells.

5.4.2.1 Simulator Design:

We devised a Cell representation to simulate the COA behavior encryption module and the multimodal recovery system, and a simple representation for the situational awareness unit. Our simulator starts by deploying Cells all over the network based on the input parameters. Each Cell should have a representation for a group of software variant sets for each possible induced change in the network. Each of these sets contains a group of similar function, different behavior variants.

At the deployment time, each Cell will have a set of numerical values representing the expected execution time for its task defined in the loaded variants. Each Cell will have a dedicated situational awareness unit monitoring incoming attacks, failure incidents, time wasted in shuffles for confusions; and diffusion, etc.

After automatically deploying these Cells, the attack, and failure event generators produces different events following the user predetermined settings. Additionally, based on the user parameters, we seamlessly replace dead or problematic Cells with new fresh Cells. All Cells that successfully complete their task are considered dead Cells; while Cells with too many failure events are considered problematic.

The variant shuffling at each Cell works seamlessly for temporal confusion induction. The set shuffling occurs only in response to an induced change in a specific network location. Set shuffling is always followed by a request to variant shuffle for a random sample of the network to induce the needed diffusion to complete the encryption process. Independent shuffling for diffusion decisions are taken based on a predetermined parameter defining the acceptance rate for the shuffle for diffusion requests.

Spatial shuffling occurs based on the input parameter to a random sample of the network. A single special shuffling event involves two Cells that will swap their location in the network. One

of these Cells is a live Cell and the other one a stem-Cell. The process starts by a request from the Cell that had the event coming to the GMS to select a migration target Cell. After migration the migrated Cell will transform to a Stem-Cell waiting for specialization.

5.4.2.2 Extracted results

In this section we present the results of multiple experiments that were performed using our MATLAB simulator. These experiments have different objectives regarding evaluating the provisioned level of security and the effect of increasing the level of security over the execution time of the application. CBE encrypt the execution behavior of the application by confusion and diffusion induction. The following experiments quantify the strength of ChameleonSoft behavior encryption mechanism in terms of confusion and diffusion induced levels. The performance aspect of the experiment is introduced through a representation of the average downtime for all the Cells in the network.

Table 5.2 shows the main parameters used in the simulation. The network parameters are mainly static parameters used to setup the experiments, except for the deployment of fresh Cells in the network. The dynamic part depends on a set of distributions mentioned in the column named “Generator “.

Through the experiment we are simulating the case that all the nodes have average capabilities and we assumed that a node would not refuse shuffling or relocation requests. With that assumption, it is closer to a population description; which makes the normal distribution a good distribution to describe the location of the next event. While the rate of change, or inter-arrival time " the time frame between consecutive events" is best represented as a Poisson distribution. Uniform distribution was selected to describe the variant selection "which variant to replace the current active variant" and independent decision making.

The shuffling event parameters represent the spatiotemporal distribution of shuffling commands to induce confusion while the attack or change in environment parameters show the spatiotemporal distribution of attack events and the event type that necessities variant set change to respond to the change. Events shuffling variants selection parameters represent the selection criteria of the next variant to be shuffled while the independent shuffling decision on each Cell parameter represents when the Cell should take shuffling decision for diffusion induction. The recovery parameter represents the initial recovery mode for each Cell, and the dynamic recovery change through the experiment life time. The "deploy new Cell" parameter represents the rate and location for the deployment of fresh Cells to replace dead or problematic Cells in the network. All experiments had the same time period of 6 hours with a sample rate of 6 mints giving us 60 samples of events of changes within the network of Cells.

Classification	Parameter		Generator	Run 1	Run2	Run3
Network	Network size		Static	10*100	10*100	10*100
	# shuffling variants in each set		Static	8	8	8
	# shuffling sets		Static	5	5	5
	Exp_Time		Static	60	60	60
	Avg_App_exe_time		normal	35	35	35
	Deploy new Cell	Period	Poisson	20	18	16
		Location	normal	8,3 98,2	8,3 98,2	8,3 98,2
Recovery	Recovery at deploy		normal	8,3	8,3	8,3
	Mode change	Period	Poisson	20	18	16
		Type	normal	8,3	8,3	8,3

Event	Temporal Shuffling event	Period	Poisson	22	18	14
		Location	Normal	8,3 98,2	8,3 98,2	8,3 98,2
	Spatial Shuffling event	Period	Poisson	22	18	14
		Location	Normal	8,3 98,2	8,3 98,2	8,3 98,2
	Failure events	Period	Poisson	18	18	18
		Location	Normal	8,3	8,3	8,3
	Attack or change in environment event	Timing	Poisson	21	20	18
		Location	Normal	11,3 99,1	9,4 99,1	10,2 99,1
		Type	Uniform	10	10	10
Software	Shuffling Variants Selection		Uniform	10	10	10
Shuffling	Independent shuffling decision on each Cell		Uniform	10	10	10

Table 5.2 CBE simulator parameters

We examined the behavior encryption module through three experiments with different settings. The experiments aimed to measure the effect of changing attack arrival rate and location with the change of shuffling event generation on the behavior output as illustrated in Figure 5.6 and Figure 5.7. The effect of continuous variant shuffling within CBE diffusion induction mechanism on the output behavior was obvious. A simple change in any of those inputs leads to significant change in the output.

Our primary goal in this study is to illustrate the effect of CBE on the overall network behavior after attack events. This study focuses on the security analysis of the system by showing the level of induced confusion and diffusion. Performance analysis will be discussed latter.

Figure 5.6 A gives a snapshot of the set and variant distribution over the Cells at bootstrap. Each column represents a Cell in the network where the value represents the current executing set index or variant index.

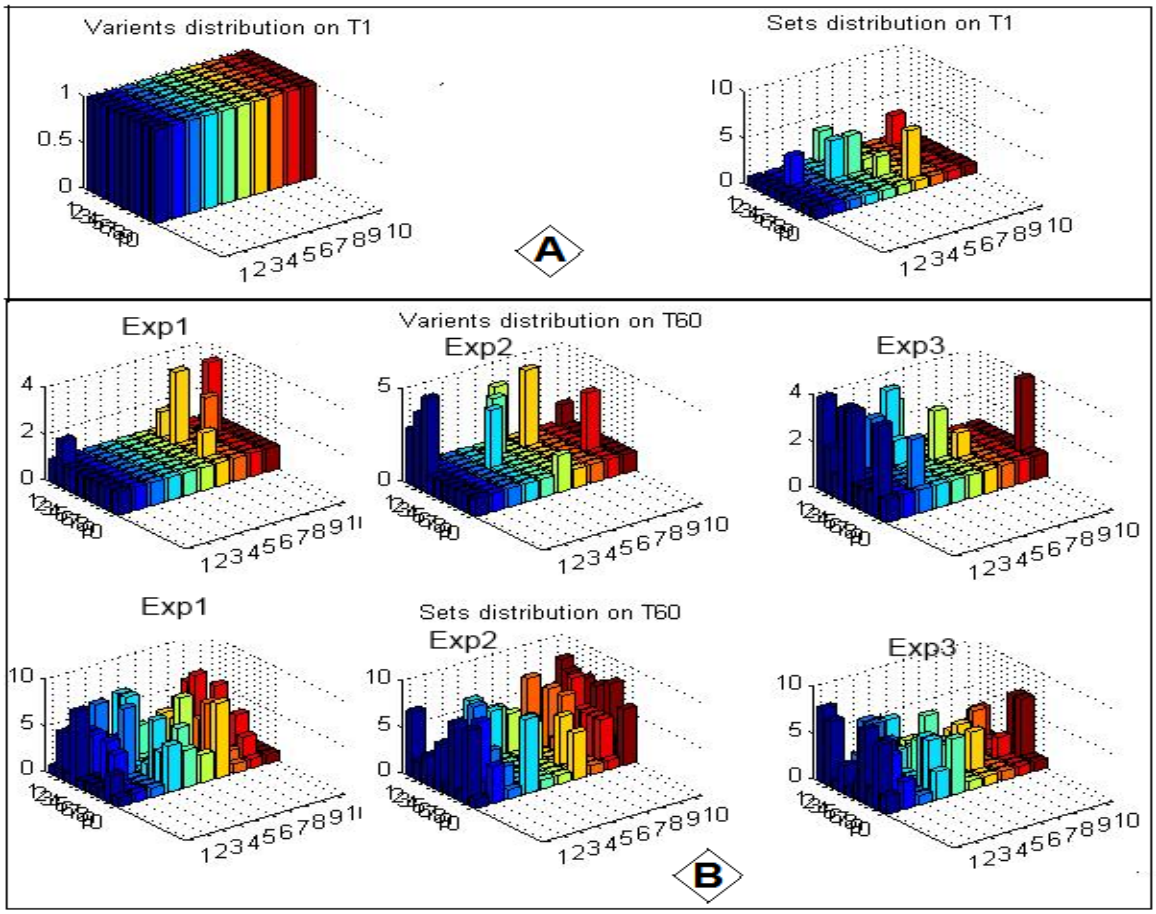


Figure 5.6 CBE Effect on the Network Behavior

In Figure 5.6 B we illustrate the behavior output after short period of continuous behavioral encryption for the three experiments. It is clear that behavior changes are diffused all over the network. This can be seen by the massive change in the behavior of the whole network by the end of the experiment.

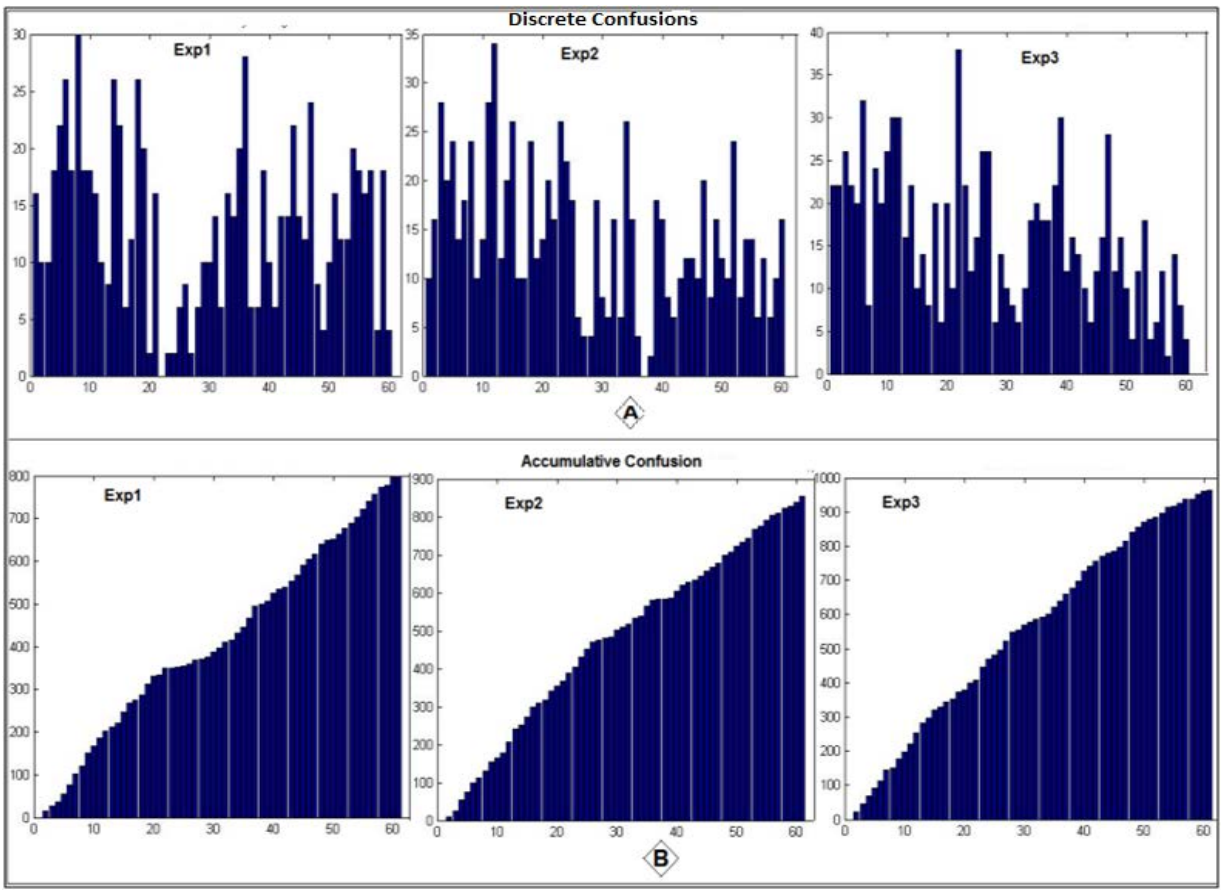


Figure 5.7 Induced Confusions and Diffusions

Figure 5.7 A reflects the total number of changes "diversity application" in the network behavior at each time tick. This is an indication for the induced confusions at each time event. Figure 5.7 B illustrates the accumulating change in the network behavior over time reflecting the effect of re-encryption and the increase in complexity of correlating the input to the output over time.

We performed multiple experiments to evaluate the performance of CBE system. Figure 5.8 illustrates four different experiments conducted to evaluate the effect of changing the recovery mode on the average downtime due to failures, and attacks using the parameters presented in Table 5.2. The first experiment conducted to evaluate the failure downtime in case of no CBE.

The remaining experiments where testing the effect of CBE with, and without recovery considering the two recovery modes hot, and cold.

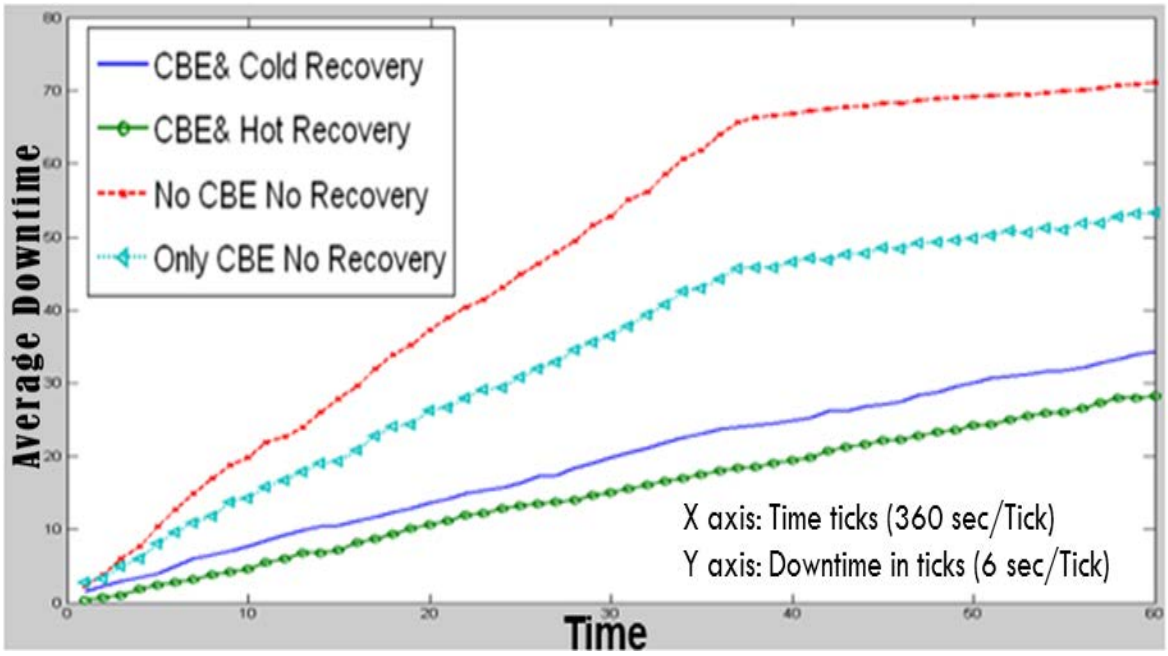


Figure 5.8 The effect of applying CBE, and the different modes of recovery on the failure downtime due to failures and attacks

The results show a noticeable improvement in the failure downtime with only CBE even without recovery. CBE saved a considerable amount of failure downtime just by mitigating wide set of the induced attacks. The situation improves when we apply our coarse and fine grained recovery that quickly resolve any coincidental or intentional failures that might result from the shuffling process itself, or from attacks.

Figure 5.9, and Figure 5.10 aim to illustrate the effect of increasing the attack arrival rate on the system downtime, and the system automated response to increase the provisioned level of security by increasing the confusion and diffusion levels to mitigate these attacks. We used same parameters of the first run in Table 5.2, and an attack generation rate range of (10,20,30).

Figure 5.9 illustrates the effect of the increase of the attack generation rate, with the presence of our CBE with no recovery, and with coarse and fine grained recovery modes. The experiment shows significant improvement in minimizing the failure downtime when the CBE is used when compared with the mono-variant mode, even without recovery. The reason behind that resides in the fact that a large portion of these attacks will fail to succeed if the targeted vulnerability was not active where, and when it was supposed to be. The average downtime significantly decreases when we activate the COA fine and coarse grained recovery mechanisms. Both recovery modes, will rabidly recover failed Cells minimizing the attack and failure downtime.

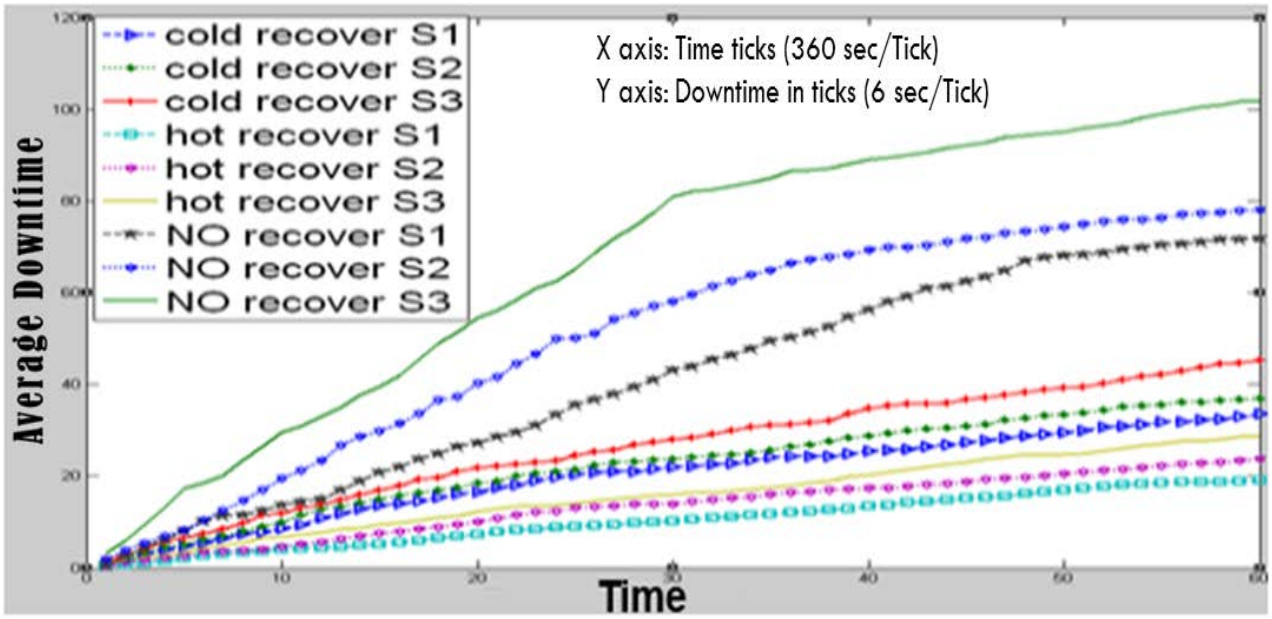


Figure 5.9 The Average downtime in response to increasing attack generation rate for, no shuffling "mono variant", and CBE with no recovery, CBE and coarse grained recovery, and CBE and fine grained recovery

Figure 5.10 shows the system automated response to the increase of incoming attack events. The system autonomously increase the level of induced confusion and diffusion levels by increasing the shuffling speed, and widening the diffusion-shuffling-requests scope to mitigate

incoming attacks, and by alerting other Cells of that event. The produced confusions and diffusions are calculated as follows:

$$Produced\ Confusions\ and\ diffusions$$

$$= \sum_{1}^{N} number\ of\ tempral\ and\ spatial\ shuffling\ of\ each\ Cell$$

These early warning alerts, will minimize the attack success ratio, minimizing the average downtime of attacks.

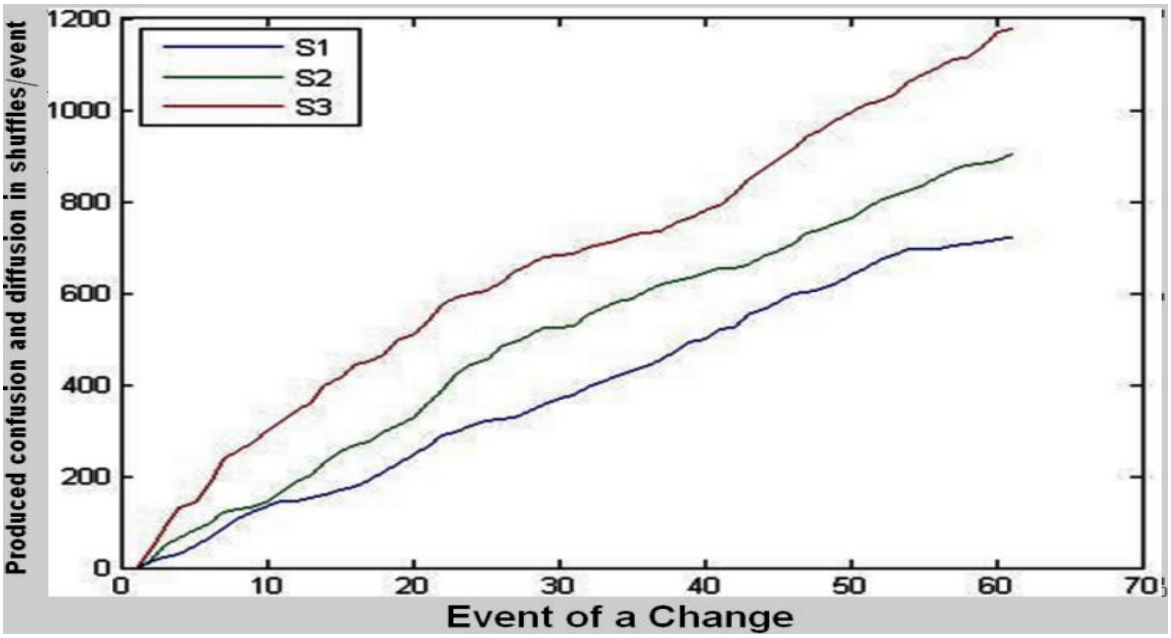


Figure 5.10 The automated system response to increase the level of provisioned security in response to an increase of attack arrival rate

We conducted more experiments to correlate the effect of enhancing the security provisioning over the system performance. We used parameters from Table 5.2, and a shuffling frequency range of (10,20,30) . Figure 5.11 and Figure 5.12, illustrates the level of induced confusion and diffusion with respect to the change in the shuffling speed over time for different behavior encryption mechanisms.

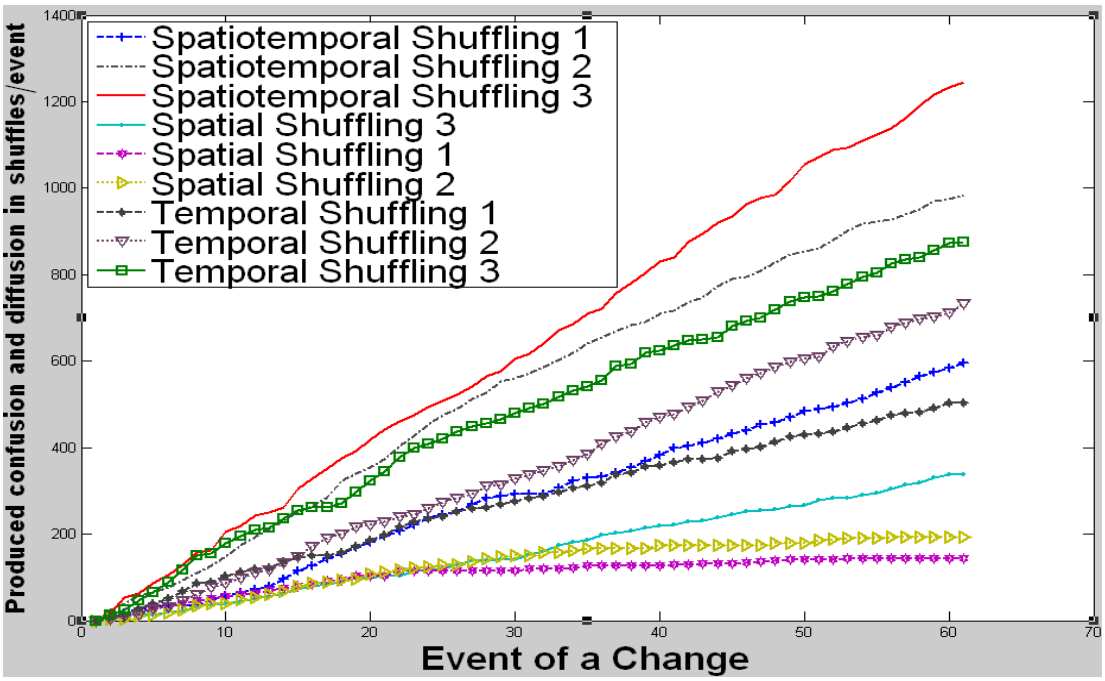


Figure 5.11 Ievel of induced confusion and diffusion with respect to a Change in the shuffling speed over time

Figure 5.12 shows the effect of shuffling frequency change on the downtime. Figure 5.11 shows the effect of changing the shuffling speed on the level of provisioned security. Figure 5.11 and Figure 5.12 illustrates that complicating the correlation between the input and output network behavior by increasing the level of induced confusion as a reflection of increasing the frequency of shuffling linearly increase the average downtime. The reason behind that comes from the increased number of preemptions prior to temporal and spatial shuffling, and the possible failures that might result from these processes.

As mentioned before ChameleonSoft is capable of changing its diversity application technique at runtime to suit changes in the surrounding environment, and application requirements. The reason behind enabling such feature is to provide some guarantees that the system will always consider using the right resources at the right time towards balancing the security and performance output of the system.

ChameleonSoft employ different recovery mechanisms with different granularity levels to suit the dynamic change in the surroundings. Fine grained recovery by Cell replication might consume more resources in order to guarantee short recovery downtime and successful restoration of all the previous states before failure. As mentioned before ChameleonSoft optimize the replication resource usage by replicating only the STM,I/O, and data store components of the Cell. The remaining components of the Cell remain in hibernation waiting for resurrection when the replica takes over. The overall recovery time depends on the time needed to resurrect the hibernated replica components and the time spent to detect failure.

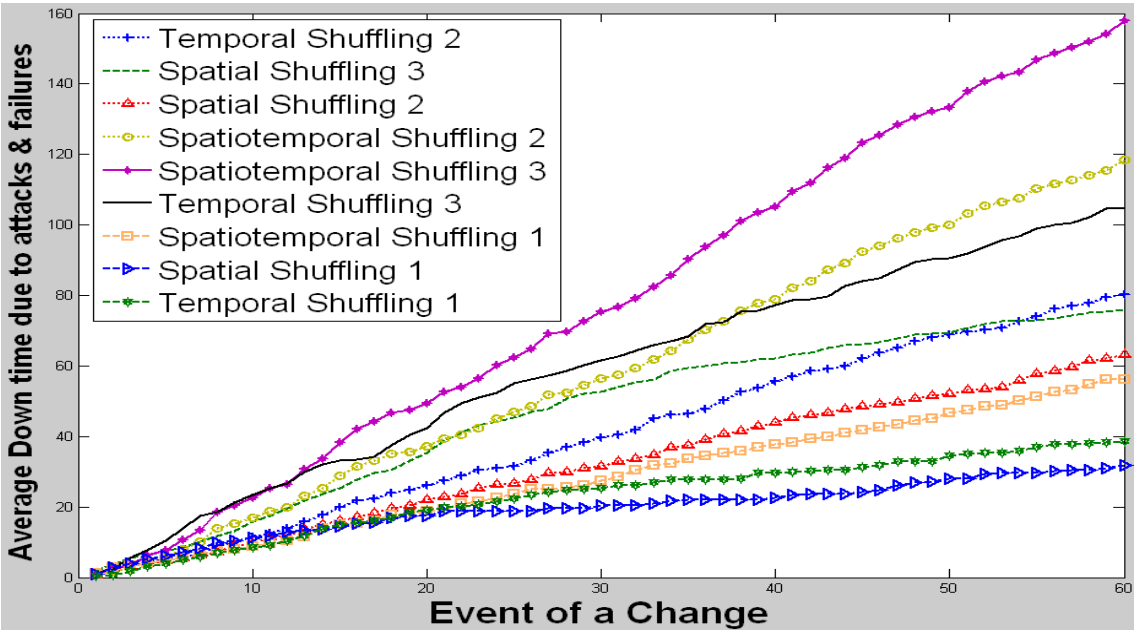


Figure 5.12 The average downtime with respect to the increase of shuffling speed increasing the level of provisioned security

ChameleonSoft usually uses coarse grained recovery mode in resource constrained environments to save the resources used by the replicated Cell components. Restoring a failed Cell with no replica might involve remote data store queries, collecting communication logs from other Cells, and analyzing these logs for unsaved lost states. This process increases the overall recovery time without any guarantee for a successful restoration for all states before failure.

5.4.3 *Observations*

From the presented results we can conclude by illustrating the following list of observations:

- ChameleonSoft software behavior encryption technique was able to induce the needed confusions and diffusions to encrypt the behavior of the running software
- ChameleonSoft dynamically responds to attack arrival rate increase by increasing the level of provisioned security
- ChameleonSoft utilized CyberX enhanced dynamic failure resilience to reduce the effect of cell failure.
  - Such failure might be the outcome of the increased attack arrival rate or the shuffling speed escalation in response to such increase.
  - This enhancement had a clear impact minimizing the average downtime of the Cell even in high attack arrival rates

5.5 **Pervasive defense provisioning, and trustworthy tipping and cueing**

In this section we study the final stage where CyPhyCARD starts the defense provisioning process through EvoSense. In this step, the defense services are hosted on the DSP side over CyPhyCARD secured platform and uses EvoSense circulation mechanism to monitor the ToD hosts and to provision the needed defense services based on the analysis of the collected data. In order to comprehensively analyze this process we start by presenting a parametric study for the different parameters controlling the various aspects of the defense provisioning techniques, followed by a quantitative study of our defense delivery mechanism using simulation.

5.5.1 *Parametric study*

Currently attack detection tools can be categorized as either locally hosted within the attack target or distributed with limited or no cooperation (between tools, or between DSPs) hosted within the attack targeted network. Table 5.3 lists estimations for the resource usage with respect to memory, storage, processing, and network bandwidth for each tools/components/ tasks in the defense provisioning process. The table presents estimated values for the purpose of illustrating the effect of enabling distributed defense provisioning, and isolating the defense provisioning process from the ToD network on the overall resource consumption. The presented estimations listed in Table 5.3 are justified by the discussion below. The table compares the resource usage of each item in case of presenting defense services through a locally hosted defense tool that use only signature based detection methods, or combines it with AI techniques to predict attacks. Additionally, we also list our estimations for the resource usage of a distributed defense provisioning platform that moves a major part of the analysis and investigation workload to dedicated servers hosted within the ToD network. The investigation included cases where AI techniques are combined with signature based techniques or not.

				EvoSense	Local & simple	Local & AI	Distributed & simple	Distributed & AI
C1	External feedback events	Memory Usage	On ToD	Negligible/none	Not Supported	Not Supported	Negligible/none	Normal
			On DSP	High	NA	NA	High	High
		Hard-disk Usage	On ToD	Negligible/none	Not Supported	Not Supported	Negligible/none	Negligible/none
			On DSP	High	NA	NA	High	High
		Processor Usage	On ToD	Negligible/none	Not Supported	Not Supported	Normal	Normal
			On DSP	High	NA	NA	High	High
		Network Bandwidth Usage	On ToD	Extremely low	Not Supported	Not Supported	Normal	Normal
			On DSP	High	NA	NA	High	High
	Scanning							
C2a	On-Demand Scanning	Memory Usage	On ToD	Low	High	Extremely High	High	High
			On DSP	Low	NA	NA	Normal	Normal

		Hard-disk Usage	On ToD	Low	High	Extremely High	High	High
			On DSP	Medium low	NA	NA	Normal	Normal
		Processor Usage	On ToD	Medium low	High	Extremely High	High	High
			On DSP	Medium low	NA	NA	Normal	Normal
		Network Bandwidth Usage	On ToD	Medium low	High	Extremely High	High	High
			On DSP	Medium low	NA	NA	Normal	Normal
C2 b	On-Access & Real-time	Memory Usage	On ToD	Low	High	Extremely High	High	High
			On DSP	Low	NA	NA	Normal	Normal
		Hard-disk Usage	On ToD	Low	High	Extremely High	High	High
			On DSP	Medium low	NA	NA	Normal	Normal
		Processor Usage	On ToD	Medium low	High	Extremely High	High	High
			On DSP	Medium	NA	NA	Normal	Normal
		Network Bandwidth Usage	On ToD	Medium	High	Extremely High	High	High
			On DSP	Medium	NA	NA	Normal	Normal
C2c	Scheduled	Memory Usage	On ToD	Low	High	Extremely High	High	High
			On DSP	Low	NA	NA	Normal	Normal
		Hard-disk Usage	On ToD	Low	High	Extremely High	High	High
			On DSP	Medium low	NA	NA	Normal	Normal
		Processor Usage	On ToD	Medium low	High	Extremely High	High	High
			On DSP	Medium low	NA	NA	Normal	Normal
		Network Bandwidth Usage	On ToD	Medium low	High	Extremely High	High	High
			On DSP	Medium low	NA	NA	Normal	Normal
D1	Static analysis and emulation	Memory Usage	On ToD	Extremely low	High	Extremely High	High	High
			On DSP	High	NA	NA	High	High
		Hard-disk Usage	On ToD	Extremely low	High	Extremely High	High	High
			On DSP	High	NA	NA	High	High
		Processor Usage	On ToD	Extremely low	High	Extremely High	High	High
			On DSP	High	NA	NA	High	High
		Network Bandwidth Usage	On ToD	low	High	Extremely High	High	High
			On DSP	High	NA	NA	High	High
D2	Heuristics	Memory Usage	On ToD	Extremely low	Extremely High	Extremely High	High	High
			On DSP	High	NA	NA	High	High
		Hard-disk Usage	On ToD	Extremely low	Extremely High	Extremely High	High	High
			On DSP	High	NA	NA	High	High
		Processor Usage	On ToD	Extremely low	Extremely High	Extremely High	High	High
			On DSP	High	NA	NA	High	High
		Network Bandwidth Usage	On ToD	low	Extremely High	Extremely High	High	High
			On DSP	High	NA	NA	High	High
D3	Tunneling signatures	Memory Usage	On ToD	low	Not Supported	Not Supported	Negligible/none	Normal
			On DSP	Medium low	NA	NA	High	High
		Hard-disk Usage	On ToD	low	Not Supported	Not Supported	Negligible/none	Negligible/none

			On DSP	Medium low	NA	NA	High	High
		Processor Usage	On ToD	Extremely low	Not Supported	Not Supported	Normal	Normal
			On DSP	low	NA	NA	High	High
		Network Bandwidth Usage	On ToD	Extremely low	Not Supported	Not Supported	Normal	Normal
			On DSP	Medium low	NA	NA	High	High
D4	Acquiring consultations	Memory Usage	On ToD	Extremely low	Not Supported	Not Supported	Normal	Normal
			On DSP	Medium low	NA	NA	High	High
		Hard-disk Usage	On ToD	Extremely low	Not Supported	Not Supported	Negligible/none	Negligible/none
			On DSP	Medium low	NA	NA	High	High
		Processor Usage	On ToD	Extremely low	Not Supported	Not Supported	Normal	Normal
			On DSP	Medium low	NA	NA	High	High
		Network Bandwidth Usage	On ToD	Extremely low	Not Supported	Not Supported	Normal	Normal
			On DSP	Medium low	NA	NA	High	High
D5	Advanced system cleaning protocol	Memory Usage	On ToD	Extremely low	Extremely High	Extremely High	Extremely High	Extremely High
			On DSP	Medium	NA	NA	Extremely High	Extremely High
		Hard-disk Usage	On ToD	Extremely low	Extremely High	Extremely High	Extremely High	Extremely High
			On DSP	Medium	NA	NA	Extremely High	Extremely High
		Processor Usage	On ToD	Extremely low	Extremely High	Extremely High	Extremely High	Extremely High
			On DSP	Medium	NA	NA	Extremely High	Extremely High
		Network Bandwidth Usage	On ToD	Extremely low	Extremely High	Extremely High	Extremely High	Extremely High
			On DSP	Medium	NA	NA	Extremely High	Extremely High
	privacey assurance							
		Memory Usage	On ToD	Extremely low	Normal	High	Normal	High
			On DSP	Medium low	NA	NA	Normal	Normal
		Hard-disk Usage	On ToD	Extremely low	Normal	High	Negligible/none	Negligible/none
			On DSP	Medium low	NA	NA	Normal	High
		Processor Usage	On ToD	Extremely low	Normal	High	Normal	Normal
			On DSP	Medium low	NA	NA	Normal	High
		Network Bandwidth Usage	On ToD	Extremely low	Normal	High	Normal	High
			On DSP	Medium low	NA	NA	Normal	Normal
E1	Searching for a resolution mechanism	Memory Usage	On ToD	Extremely low	Extremely High	Extremely High	Extremely High	Extremely High
			On DSP	Medium low	NA	NA	Extremely High	Extremely High
		Hard-disk Usage	On ToD	Extremely low	Extremely High	Extremely High	Extremely High	Extremely High
			On DSP	Medium low	NA	NA	Extremely High	Extremely High
		Processor Usage	On ToD	Extremely low	Extremely High	Extremely High	Extremely High	Extremely High
			On DSP	Medium low	NA	NA	Extremely High	Extremely High
		Network Bandwidth Usage	On ToD	Extremely low	Extremely High	Extremely High	Extremely High	Extremely High
			On DSP	Medium low	NA	NA	Extremely High	Extremely High
E2a	Quarantine	Memory Usage	On ToD	Extremely low	Extremely High	Extremely High	High	High
			On DSP	Medium low	NA	NA	High	High
		Hard-disk Usage	On ToD	Extremely low	Extremely High	Extremely High	High	High

			On DSP	Medium low	NA	NA	High	High
		Processor Usage	On ToD	Extremely low	Extremely High	Extremely High	High	High
			On DSP	Medium low	NA	NA	High	High
		Network Bandwidth Usage	On ToD	Extremely low	Extremely High	Extremely High	High	High
			On DSP	Medium low	NA	NA	High	High
E2b	Cost of dedicated full time scanners for the quarantine box	Memory Usage	On ToD	Extremely low	Normal	High	Normal	High
			On DSP	Extremely low	NA	NA	Normal	Normal
		Hard-disk Usage	On ToD	Extremely low	Normal	High	Negligible/none	Negligible/none
			On DSP	Extremely low	NA	NA	Normal	High
		Processor Usage	On ToD	Extremely low	Normal	High	Normal	Normal
			On DSP	Extremely low	NA	NA	Normal	High
		Network Bandwidth Usage	On ToD	Extremely low	Normal	High	Normal	High
			On DSP	Extremely low	NA	NA	Normal	Normal
E2c	Cost of prioritized reports							
		Memory Usage	On ToD	Extremely low	Normal	High	Normal	High
			On DSP	Extremely low	NA	NA	Normal	Normal
		Hard-disk Usage	On ToD	Extremely low	Normal	High	Negligible/none	Negligible/none
			On DSP	Extremely low	NA	NA	Normal	High
		Processor Usage	On ToD	Extremely low	Normal	High	Normal	Normal
			On DSP	Extremely low	NA	NA	Normal	High
		Network Bandwidth Usage	On ToD	Extremely low	Normal	High	Normal	High
			On DSP	Extremely low	NA	NA	Normal	Normal
E3	Process resolution protocol							
		Memory Usage	On ToD	Medium	Normal	High	Normal	High
			On DSP	Medium low	NA	NA	Normal	Normal
		Hard-disk Usage	On ToD	Medium	Normal	High	Negligible/none	Negligible/none
			On DSP	Medium low	NA	NA	Normal	High
		Processor Usage	On ToD	Medium low	Normal	High	Normal	Normal
			On DSP	Medium low	NA	NA	Normal	High
		Network Bandwidth Usage	On ToD	Medium	Normal	High	Normal	High
			On DSP	Medium low	NA	NA	Normal	Normal
E4	Reporting events							
		Memory Usage	On ToD	Negligible/none	Not Supported	Not Supported	Normal	Normal
			On DSP	Medium low	NA	NA	High	High
		Hard-disk Usage	On ToD	Negligible/none	Not Supported	Not Supported	Negligible/none	Negligible/none
			On DSP	Medium low	NA	NA	High	High
		Processor Usage	On ToD	Negligible/none	Not Supported	Not Supported	Normal	Normal
			On DSP	Medium low	NA	NA	High	High
		Network Bandwidth Usage	On ToD	Negligible/none	Not Supported	Not Supported	Normal	Normal
			On DSP	Medium low	NA	NA	High	High
E5	Updating signature database							

		Memory Usage	On ToD	Extremely low	Not Supported	Not Supported	Normal	Normal
			On DSP	Medium low	NA	NA	High	High
		Hard-disk Usage	On ToD	Extremely low	Not Supported	Not Supported	Negligible/none	Negligible/none
			On DSP	Medium low	NA	NA	High	High
		Processor Usage	On ToD	Negligible/none	Not Supported	Not Supported	Normal	Normal
			On DSP	Medium low	NA	NA	High	High
		Network Bandwidth Usage	On ToD	Extremely low	Not Supported	Not Supported	Normal	Normal
			On DSP		NA	NA	High	High
	sharing and exchange of information	Memory Usage	On ToD	Negligible/none	Not Supported	Not Supported	Negligible/none	Normal
			On DSP	Medium	NA	NA	High	High
		Hard-disk Usage	On ToD	Negligible/none	Not Supported	Not Supported	Negligible/none	Negligible/none
			On DSP	Medium	NA	NA	High	High
S		Processor Usage	On ToD	Negligible/none	Not Supported	Not Supported	Normal	Normal
			On DSP	Medium	NA	NA	High	High
		Network Bandwidth Usage	On ToD	Negligible/none	Not Supported	Not Supported	Normal	Normal
			On DSP	Medium	NA	NA	High	High

Table 5.3 Comparisons between different detection mechanisms

The following parametric study focuses mainly on the parameters that have great impact on the aspects under evaluation. Figure 5.13 presents an anatomy of the typical defense provisioning platform illustrating the main components of defense provisioning process. These components are used to construct the equation that we used as a key tool to analyze the cost of enabling EvoSense evolutionary sensing and effecting.

The total cost of executing a defense system locally within the host can be estimated as described in the following equation (1),

Equation 1, The total cost of executing a defense system locally within the host

$$A + B + (C1 * EXFB) + \left(C2a\text{evants} * \frac{S}{E1}\right) + \left(C2b\text{events} * \frac{S}{E2}\right) + \left(C2c\text{events} * \frac{S}{E3}\right) + (D1 + D2 + D3 + D4 + D5 + D6) + (E1 * SE) + \left((E2a * BB) + (E2B * SC) + (E2c * NR) + E2d\right) + E3 + E4 + E5$$

Where, *EXFB is the number of feedback sources , S/E1 is the number of sensor / on demand event , S/E2 is the number of sensor / on access event, S/E3 is the number of sensor / scheduled event, SE is*

the search elements, BB is the number of boxes, "E2a = cost of a single box", SC is the number of scanners/box, and NR is the number of reports

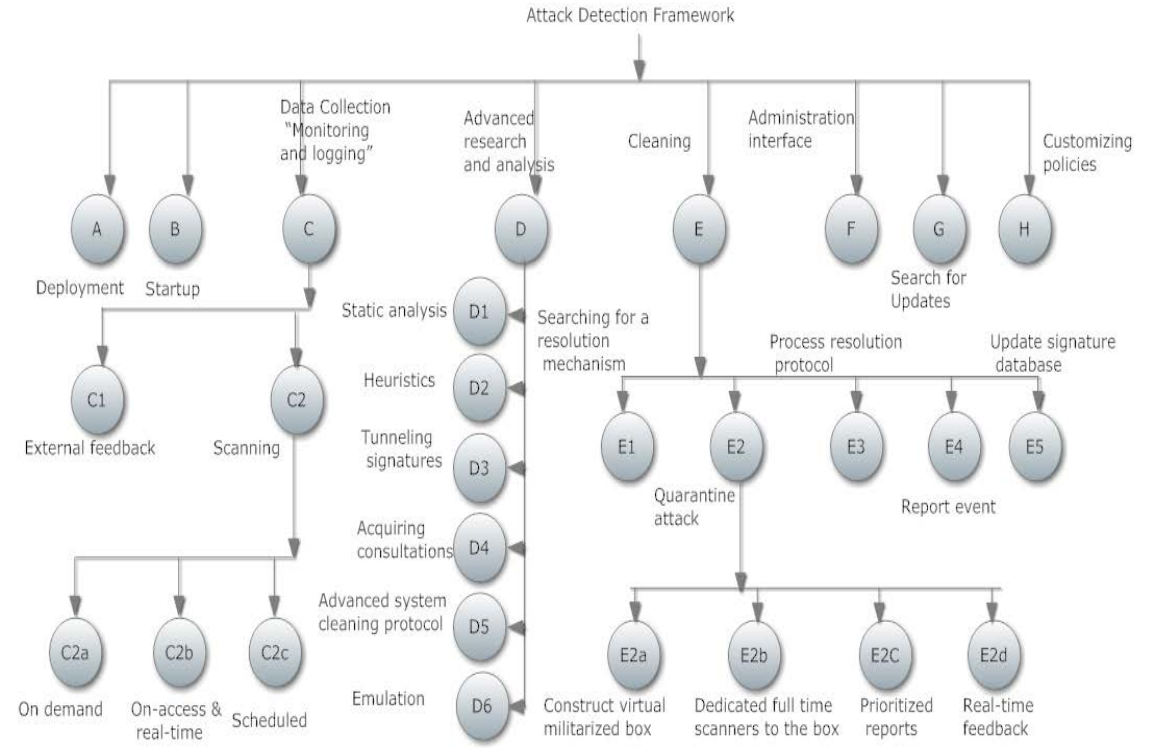


Figure 5.13 The anatomy of attack detection tools

When we used equation (1) to estimate the effect of the aforementioned parameters on the total cost vary by the variation of the target of calculation. For example some of the parameters would have significant effect when used to calculate the effect of defense provisioning on the memory usage, while same parameters might not have same effect when used to calculate the network bandwidth usage.

Equation (1) presents the total cost of the detection process as an accumulation of the cost of activating the different components of the detection tool. By analyzing the list of components we can notice that some of these components add only static cost that neither adds a specific workload once nor over time. Such parameters will not be affected by changing the detection tool

or mechanism. In this study we are going to focus only on the dynamic cost components, where the added workload changes massively by changing the defense provisioning tool or technique.

The following study aims to analyze the total cost of defense provisioning in terms of resource usage and time to detect attack. The study always refer to five main defense provisioning mechanisms , four of them represents an abstract classification of the conventional defense provisioning techniques (Local, Local with AI, distributed, and distributed with AI) based on their working environment distance from the ToD, and the fifth is our EvoSense.

The study aims to illustrate the effect of enabling trustworthy information sharing and exchange, and the effect of autonomous sharing defense tools between different components of the DSP on the studied aspects. Additionally we also aim to illustrate the effect of enabling pervasive sensing and effecting on the attack dispersion, and the time needed to immune the ToD hosts against it.

5.5.1.1 Total consumed resources

The most resource consuming components in any defense provisioning tool based on the anatomy presented in Figure 5.13 is presented in branch C, D, and E of the tree. These branches represent, the data collection by monitoring, scanning and logging; the research and analysis either by static analysis, emulation environment monitoring, heuristics, acquiring for consultations from cooperating units if supported by the defense provisioning tool, and composing the cleaning protocol.

The 3rd branch is the cleaning phase that involves executing the resolution protocol, processing of the infected items to restore it to original condition, and quarantine attacks in controlled sandboxes while closely monitoring the attack interactions within the box.

The main cost of the C branch “data collection by monitoring, scanning and logging” resides in the resources consumed by the sensing and logging elements utilized by the data collection tools. These tools are either acquired when needed or stored locally at each host. Also these tools are either generic that can be composed at runtime to collect specific data, or specific where each component has its own list of tools.

Tool reuse is expected to have a huge impact on the overall consumed resources by the data collection unit. For example, if we have N component using M sensing element/component then without sharing we have a total of N*M sensors. If on average each sensor consumes k memory or storage space then we have N*M*K total consumed resources. In case of sharing with average of h% shares then the total will be N*M*K*h% where H<100. The higher the value of (H) is, the lower resource usage.

Conventional defense tools are mostly hardwired with a set of components that has its own sensing elements integrated with the control logic. To our best of knowledge, the concept of abstract sensing and effecting was not presented by any other technique rather than EvoSense. Abstract sensing and effecting is a key enabler for resource sharing and reuse.

Distributed defense provisioning mechanisms utilize some sort of resource sharing by enabling a single remote node to analyze the feedback from multiple hosts. The effect of that sharing is expected to be clear with respect to parameters presented in the D branch of the tree. EvoSense is designed to maximize resource sharing increasing the value of H. a single sensors feedback can be utilized by multiple analysis and control techniques on EvoSense side. Additionally, the utilizing the idea of programmable generic sensors, enable EvoSense Sensors to be used for collecting information regarding different aspects of the system. Further, EvoSense sensors are designed to crewel thought the ToD network and reside back on EvoSense side. EvoSense sensors are not

supposed to consume much storage or memory resources of the host, as they are all on demand sensors that gets loaded only when needed then disposed automatically afterwards.

For the D branch “advanced research and analysis” we should mainly worry about the consumed processing and memory usage. Most of the available defense tools carry partial or full analysis of the senor feedback locally within the host. Distributed defense provisioning tools, tries to minimize the processing workload on the local host by moving the high workload part to remote servers [79,72]. However, they do not save much as the feedback sent from the local agent to the remote server will be limited not to violate the privacy policy of the ToD. Additionally, these tools were not designed to enable abstract sensing enabling automated sensor feedback sharing saving huge network bandwidth in case of transferring the feedback to remote servers for analysis.

Based on the anatomy presented in Figure 5.13 D2 and D6 are expected to be the most resource consuming among the D branch. Using heuristics techniques to predict or reveal unknown threats is known to noticeably increase the defense provisioning workload. For that locally hosted defense tools tries to limit the investigation depth or the utilized techniques when heuristics is used. Doing so limits the prediction capability and the accuracy of the detection tool and increases the chance of false positives.

One of the most effective, and accurate way in predicting unknown threats is the environment emulation presented by D6 , where attacks gets quarantined in a controlled virtual environment for close monitoring and investigation. Some of the locally hosted tools can do that [69], while this process involves huge resource consumption needed to create the virtual environment and to keep the created virtual box and application executing within it under full time supervision. Distributed defense systems solutions can do that remotely on dedicated servers saving a

considerable amount of the ToD resources. However, given the current virtualization technology, the DSP invest a considerable amount of resources to apply such technique. Additionally, with the limited cooperation nature of the current defense provisioning tools, multiple virtual environments might be created to investigate same events.

With EvoSense intrinsic cooperative feature, such unnecessary duplication will be limited or omitted saving the DSP network a considerable amount of resources. Further, EvoSense utilizing the COA infrastructure, can create a virtualization environment for suspicious application with one of the COA Cells, saving much of the resources wasted in creating a fully virtualized environment utilizing one of the currently available techniques. COA Cells are Nano virtual machines that can create a fully/Simi virtualized environment. Additionally, COA Cells are designed to be in full time monitoring and supervision making it easier for EvoSense to monitor the execution of the enclosed suspicious application.

The E branch is mainly concerned about after attack, and attack immunization process. Usually this process involves cleaning or quarantine the infected application. In case of simple cleaning the cost is relatively similar regardless of the technique used to provision defense services, while in some cases cleaning becomes impossible and the application would revert to one of two options, either delete the file, or to quarantine it. The quarantine is usually located within the host, and the infected application is always under full time supervision by dedicated monitors. The process is computationally expensive especially when the number of infected files passes certain threshold.

One of the advantages that EvoSense grants to the defense provisioning process is the ability to use the COA Cells to encapsulate suspicious applications. Doing so facilitates moving the

application outside the ToD to be executed in a more controlled environment. In this case, the Cell will be under deep surveillance without adding extra workload to the ToD hosts.

Using COA Cell as a virtual environment either for analysis or as quarantine to threats is computationally cheaper than using a fully virtualized environment utilizing any of the available virtualization techniques. Meanwhile, EvoSense copy the infected application to one of its remote serves to apply deep cleaning methods which might be effective to clean the infection. Additionally, EvoSense can acquire a clean copy of the application from one of the available backups of the host before infection and use it to replace the infected one.

Further, based on a research work of [68], a large set of threats massively increase the resource consumption of their targets after infection. Time to detect, resolve, contain attacks is an important factor in the overall estimated resource consumption of a networked system specially after being infected by any of those attacks. The following discussion will represent that EvoSense pervasiveness and autonomic defense missions and tips sharing can effectively decrease the attack dispersion by rapidly distributing detection and resolution tools. Doing so, is expected to have a noticeable effect on minimizing the resource consumption that would have been wasted by the attack.

5.5.1.2 Time needed to detect attacks

The technique used to detect attacks has a great impact on the time frame between infection and detection. Locally hosted techniques relay on signature database containing all the signature of known attacks. AI enabled techniques use heuristics to allocate suspicious applications, and report it or quarantine it. Large distributed techniques, feed their detection host side application with frequent updates about recently known attacks in the form of added signatures. However,

both of them relay mostly on central servers that distributes new signatures for the recently found threats. The time needed to device such signature is a major player in this process.

The D branch includes the key components controlling the duration between infection and detection. The components are either active or passive. For the active components like D2 and D6, the spent in collecting and analyzing the sensing elements feedback either by one/more heuristic technique has a great impact on the duration between infection and detection. Passive components like D1, D3, and D4 relay on the speed of attack event distribution, and the time needed to device valid signature for the attack.

Mostly, the conventional detection techniques do not share their attack prediction component reports due to the high risk of violating the privacy policy of the ToD, and due to incompatibility and lack of abstraction between defense tools. Different versions of the same products built by the same company might utilize different formats to represent their signature database, or detection/resolution protocols. Further, attack reports might indicate attacks that require specific sensing equipment to be detected. Conventional detection tools are not designed to share tools, as this might violate their manufacturer copy rights.

Generally speaking, sharing events without appropriate tools to verify the existence of certain attack within the ToD network can be considered useless.

EvoSense is built to support DSP cooperation locally within the DSP network, and globally between EvoSense enabled DSPs. Enabling such cooperation reduces the time frame between infection and detection, as EvoSense autonomously share detected attack events through the pervasive management and control units distributed all over the ToD network. EvoSense sensors extract privacy friendly information from infections hosts that can easily be shared with the tools

used for detection in one package “defense mission”. EvoSense share attack events easily without violating the privacy policy of the ToD.

EvoSense leverage the homogeneity feature of its COA based infrastructure to enable sharing attack detection packages “sensing missions”. The package includes set of tools and an execution protocol and attack detection assurance mechanism to be distributed within the TOD network of any EvoSense capable DSP. EvoSense sensing missions describes the behavior of an infected application, rather than attack signature. Doing so massively reduce the time wasted in manually/automatically creating signatures.

Let us assume that the average infection rate R/sec, then at time T we should have

IM=2 ^(T*R) infected machines.

If the time needed for the first attack to be detected is X, the time needed for a signature to be devised is Z, the time needed for a single machine to have a copy of the signature is Y, at time to resolve an infected computer is V

Time to safety either by immunization of by recovery and immunization is calculated as follows: X+Z+ ((Y*(N-IM))+(Y*V*IM))

The most time consuming process is the (X+Z) period. It has a huge impact on the cost of attack. The attack cost is an estimation of the losses occurred due to infection like, overloading the host, or the host network, interruption of operation, … etc.

The time V also depends on the nature and the severity of the attack, some attacks sets the detection system itself as one of its primary targets, doing so complicates the process of automatic recovery.

With conventional systems, the value of X, and Z depends mostly on manual or semi-automated analysis. The value of V depends on the cleverness of the defense tools, and the attack type. With EvoSense, the value of X is the only important factor, as by the time of detection, the system automatically responds to attacks by quarantining it either locally at the infection point, or in a remote sandbox. In EvoSense, a successful mission that detects attack gets automatically distributed to all machines connected to EvoSense enabled system minimizing the values of Z and Y to great extent. The value of V is supposed to be small, as the system responds to threats automatically by containing the attack in controlled environment until a resolution effector resolves the situation. Using such technique limits the attack dispersion (R) minimizing the number of infected machines.

From the presented study we can notice that using EvoSense to deliver defense services is expected to reduce the overall consumed resources to detect attacks, and the time between infection, detection, and resolution/immunization. Next I will present EvoSense sensing circulation protocol.

5.5.2 *Simulation results*

We use simulation to conduct four experiments for the purpose of evaluating EvoSense performance. The first experiment evaluates the overall downtime with and without EvoSense evolutionary features while increasing attack arrival rate, it also evaluates the effect of widening the defense experience sharing-scope on the system downtime. The second experiment evaluates the effect of changing the pervasiveness-density while increasing the attack diversity on the amount of consumed resources, on the attack detection promptness, and attack detection accuracy. The third experiment is to illustrate the effect of sensor circulation of the system performance in

regard to resources consumed, and time to detect attacks. Finally, the last experiment aims to illustrate the effect of using dynamic host classification using profiles on the system performance aspects.

5.5.2.1 Simulator design

We used MATLAB to build a simulation program representing a 100*10*10 network of hosts classified into different hosts in organizations and enclaves. Each node in the network holds records for attack and resolution history during the experiment time.

Attacks are spatiotemporally distributed over the hosts based on a set of random distributions as illustrated in Table 5.4. We designed defense mission generation module that mimics the defense service provisioning in the real systems applying EvoSense. This mechanism uses containment organisms to resolve un-resolvable attacks. We assume that containment organisms will resolve the situation locally. Deploying containment organisms increases the downtime at the deployment point by an estimated predetermined value reflecting the time needed to contain the problem and restore the host to its original state. We used a predetermined threshold representing the risk-factor, it is the time needed to authorize containment organisms deployment.

Defense missions are rewarded at each successful resolution attempt. Rewarded missions are shared and applied to other nodes within the sharing scope of the experiment. We tested the sharing effect on the overall downtime using three scopes, single enclave, single organization, and all community scopes.

The simulator was built assuming that attack activity is always detectable and it is either resolvable or containable, sharing is only for successful attacks while containment events is not for sharing, and all missions has only one active role.

Classification	Parameter		P_Type	Low	Normal	High	Very High
Network	Number of hosts		Static	100	100	100	100
	Number of enclaves		Static	10	10	10	10
	Number of hosts/management unit		Static	5	5	5	5
	Number of organizations		Static	10	10	10	10
	Number of ToD hosts in each enclave		Static	3	5	5	7
	Host participation hosting sensors		Static	30%	30%	30%	30%
	Number of profiles		Static	3	3	3	3
	Number of Hosts/Profile		Uniform	3,3	3,3	3,3	3,3
Event	Profile change	Timing	Poisson	10	10	10	10
		Type	Uniform	3,3	3,3	3,3	3,3
	Number of active defense missions	Timing	Poisson	10	20	25	30
		Locations	Normal	10,10, 100	10,10, 100	10,10, 100	10,10, 100
		How many hosts	Uniform	100	100	100	100
		Expiration date	Uniform	10	10	10	10
		Number of sensors/effectors	Uniform	5	5	5	5
		Resource usage on ToD	Uniform	12	12	12	12
		Resource usage on DSP	Uniform	40	40	40	40
		Resolution time	Uniform	5	5	5	5
		Type	Uniform	10000	10000	10000	10000
	Attack dispersion	Locations	Normal	10,10, 100	10,10, 100	10,10, 100	10,10, 100
		Scope	Uniform	5	15	35	65

		Rate	Poisson	20	50	70	100
	Sharing	Sharing scope	Uniform	15	15	15	15
		Sharing location	Normal	10,10	10,10	10,10	10,10
	Attack event generation	Timing	Poisson	10	20	30	50
		Added downtime on ToD	Uniform	10	10	10	10
		Resource usage on ToD	Uniform	10	10	10	10
		Location	Normal	10,10, 100	10,10, 100	10,10, 100	10,10, 100
		Type	Uniform	10000	10000	10000	10000
Conventional Deployment Settings	Sensor Set Size		Uniform	5	5	5	5
	Sensor Set Locations		Normal	10,10	10,10	10,10	10,10
	Sensor Set scope		Uniform	5	5	5	5
	Sensor Set Deployment rate		Poisson	15	15	15	15
Evolution	Sensor Circulation	Timing	Poisson	15	30	50	80
		Deploy location	Normal	10,10, 100	10,10, 100	10,10, 100	10,10, 100
		Scope	Uniform	5	15	25	40
		Package size	Uniform	5	5	5	5
		Type	Uniform	10000	10000	10000	10000
	Containment	Contain after	Uniform	15	15	15	15
		Containment penalty "time"	Uniform	100	100	100	100
		Containment penalty "Resources"	Uniform	100	100	100	100
	Defense missions generation rate		Poisson	10	10	10	10

Table 5.4 EvoSense simulation parameters

In the presented results, we are simulating the case that all hosts have same chance in getting infected. Further, we assumed that the senor unguided circulation for the sake of detecting unknown attacks uses balanced host selection scheme. With that assumption, it is closer to a

population description; which makes the normal distribution a good distribution to describe it. The rate of change or inter-arrival time is best represented as a Poisson distribution.

We used three factors to evaluate EvoSense performance:

- **Pervasiveness density** reflects the amount of detection sensors deployed on each node.
- **Attack diversity** is the total number of successful attack types with respect to the overall number of attacks per node.
- **Detection accuracy** is the number of successfully detected attacks with respect to the overall number of attacks. Promptness is the time between infection and resolution.

During the experiments, at each sensor deployment, resource usage calculated for the targeted node is incremented by the estimated value of resources to be used by this sensor. Additionally, at each attack event, the amount of consumed resources at the attack-targeted node is increased by a predetermined value relevant to the type of attack.

The simulation experiments were built based on the following assumptions:

1) No immunization effect to save the computational power

2) Same number of attack types for each run (1, 2 , 3 ,4) and same # of machines

3) Use-everything and Commonly-used modes use the DSP resources for sharing only

4) All attacks are detectable there is a sensor matching all available attacks.

The commonly used mode “conventional tools”, is a random set of the sensors deployed on all hosts at all times, new sensors are added with a slow rate to mimic the normal update process of conventional tools. The update settings and the initial set size is illustrated on section “Conventional Deployment Settings” in the table.

The use-everything mode, use all the sensors and pre-deploy them on the host at all times, no expiration, no generation. This mode represents the most effective way to detect attacks.

The evolution section represent EvoSense crawlers “circulation and containment” , and the frequent deployment of sensors and effectors that represent the normal remote sensing, analysis , and resolution.

Average time to detect attacks = Total time spent to detect the attack on all infected machines for all attacks divided by the (total number of attacks + total number of machines)

Now we will discuss the simulation parameters presented in Table 5.4.

The network parameters:

Number of hosts, enclaves, organization, Number of ToD hosts in each enclave, and Number of hosts/management unit represents the network construction and the host distribution among enclaves and organizations. We use static values for this construction through the presented experiment.

The “Host participation in hosting sensors “parameter represents the participation ratio of the host in hosting sensing and effecting tools permanently in the ToD local platform. Increasing this value to 100% means I am using only conventional defense tools with full defense package hosted in the host. EvoSense rule in this case is just a sharing platform. In the case of using 0% participation means I am moving all defense services to the remote DSP platform at all times, no locally hosted sensing or effecting tools are allowed to be stored in the ToD platform.

Events:

Active defense missions: these set of parameters describes the configuration related to generating defense missions “sensors, and effectors”. The timing sub parameter presents the time slot and the rate of defense mission deployment on each host. This parameter is initialized and adjusted based on Poisson distribution. The value of (λ) controls the generated result for the next round determining the frequency of defense mission deployment. The Location sub parameter

represents the deployment locations for the next deployment event. We use normal distribution to generate a set of random locations for the next deployment package. the number of hosts is determined by the" how many hosts parameter". The sub parameter Number of sensors/effectors determines the size of each mission in terms of sensors and effectors. We use uniform distribution to generate this number to stabilize the sensor load among deployed missions.

The parameters resource usage on ToD and on the DSP determines the amount of consume resources by the deployed sensors on both platforms for each time event spend on the deployed sensors while being active. Resolution time parameter determines the penalty applied to the execution downtime as the time needed to clear the threat. The type parameter determines the mission type to be deployed on the ToD. These parameters are initialized using uniform random number generator.

Attack dispersion rate and location parameters determine the attack dispersion locations for each time event identified by the Rate value. After infection, each attack spreads to a set of locations with size initialized by the Scope parameter at the frequency determined by the rate parameter.

Sharing scope and location parameters determine the scope and the locations of sharing successful sensors with other hosts. We use normal and uniform distribution to initialize locations and scope parameters respectively.

Attack event generation parameters is described by five sub parameters, timing to determine the generation and deployment frequency and is initialized by Poisson distribution, the Added downtime on ToD parameter determines the effect of infecting a host on the execution downtime and is initialized using uniform distribution. The Resource usage on ToD Parameter determines resource load increase on the ToD due to attack and is initialized by a uniform distribution.

The following sub parameters. The location and type parameter describes the location and type of the deployed attack and is described by a normal and uniform distribution respectively.

Evolution :

The Sensor circulation parameter describes the configuration of the sensor circulation mechanism in EvoSense. The timing describes the rate of circulation and is initialized using a Poisson distribution, deploy location is the locations; scope is the list of hosts that will receive the sensor package. The package size and type, describes the number and types of sensors in each deployment package.

The containment parameter represents the containment process of deployed attacks. When the value identified by the “Contain after” parameter expires, the containment effectors is deployed and the process starts. The containment process has a penalty on the resources and on the downtime identified by the parameters listed.

The Defense mission generation rate describes the frequency of adding new missions to the list of available missions for deployment.

5.5.2.2 Extracted results and discussion

The simulator is designed to illustrate the effectiveness and efficiency of EvoSense. The results were all generated using random distributions that were configured based on the parameters in Table 5.4. The main objective of the experiments presented in this section is to illustrate the effectiveness and efficiency of EvoSense when compared with “Use everything approach” that simulate the case of deploying all the available attack detection tools on all machines at all times. This mode guarantees 100% effective attack detection.

In real world this mode is impractical as it simulates using multiple tools working together on same machine. This setup has been proven to be bad due to the uncooperative, and unawareness

nature of the available tools that might lead to multiple conflicts [1,2,3]. The commonly used mode simulates the normal case of using a set of the available defense tools packaged in one defense solution. This mode represents the most commonly used case in terms of efficiency as it provides acceptable levels of guarantee that it will detect most of the attacks, while using a reasonable amount of resources. We used this mode to replace the optimal detection mode, as it was proved multiple times by researchers that the problem of attack detection is an NP-Complete problem [96,97,98]. We used the most commonly used mode because it is a practical solution closer to what is being used in the real world. Additionally it is a solution that balances attack detection accuracy with reasonable resource usage.

The EvoSense crawlers mode describes EvoSense pervasive sensing, with intelligent circulatory sensing mechanism, and trustworthy sharing.

The following illustrates the evaluation matrix we used to extract the presented results:

- Total consumed resources=

$$\sum_1^R \sum_1^E \sum_1^N \sum_1^n \big((the\ cost\ in\ \$\ of\ the\ usage\ of\ each\ peripheral) * (usage\ of\ this\ prefral)\big)$$

*Where **N** is the number of hosts in each enclave, E is the number of enclaves in each organization, R is the number of organization, and **n** is the number of peripherals/ host*

- Total time to detect attacks

$$=\sum_1^R \sum_1^E \sum_1^N \sum_1^n (time\ difernce\ scince\ infection\ to\ dection\ for\ that\ attack)$$

*Where **N** is the number of hosts in each enclave, E is the number of enclaves in each organization, R is the number of organization, and **n** is number of attacks infected that host*

- Attack density: refers to the concentration of attacks/ hosts which is equal to the total number of (active attacks for experiment time ***T***) / (total number of hosts ***N***)
- Average circulation frequency: refers to the concentration of sensors/ hosts which is

equal to $(\sum_1^N(deployed\ sensors\ /host\ \ for\ time\ t) * T\ ))$/N

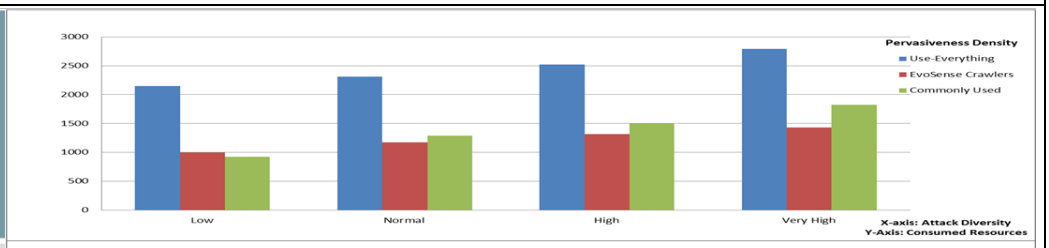
*Where **N** is the number of hosts*

- Attack diversity: is the total number of successful attack types with respect to the overall number of attacks per host

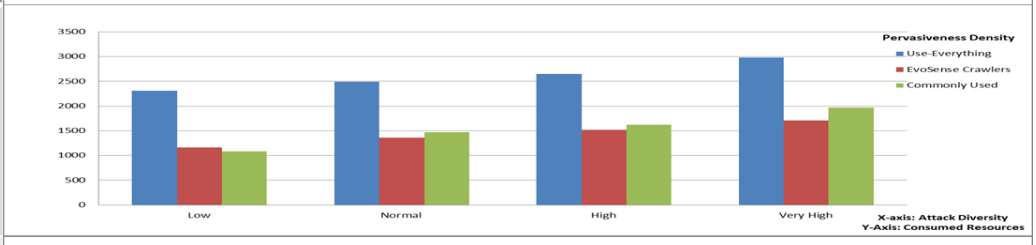
The experiment is two parts, one focuses on evaluating the efficiency of detection in terms of resource usage, and the other one focuses on the effectiveness of the detection in terms of time to detect attacks.

Resources, ToD side

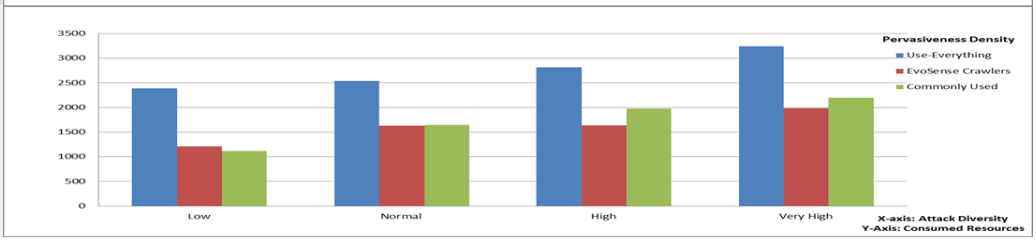
Inter-Organization [A1]



Organization [A2]



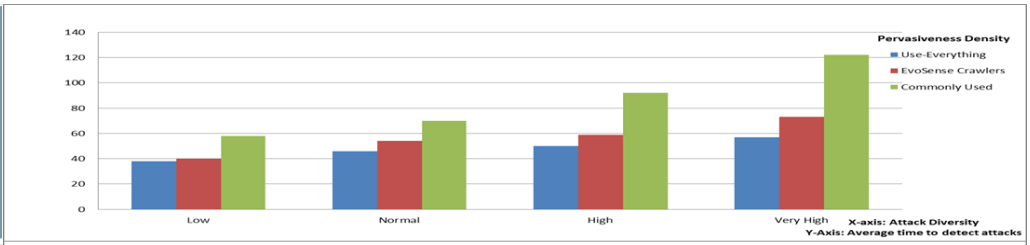
Enclave [A3]



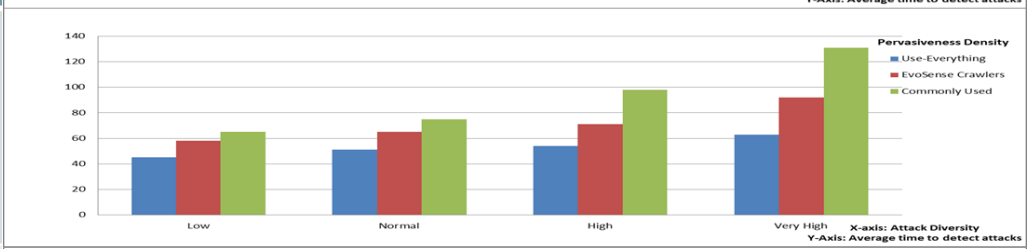
Average time to detect attacks in (ticks)

Y axis: ticks (6 sec/tick)

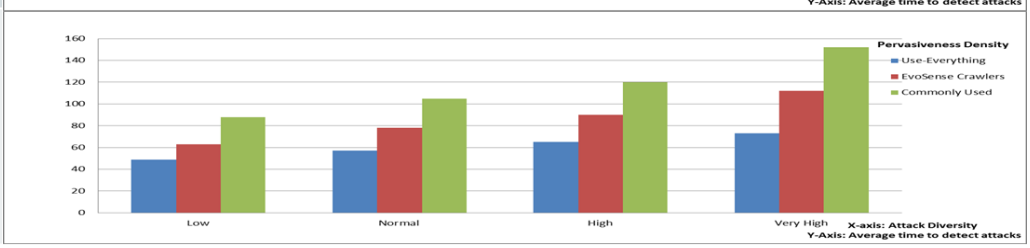
Inter-Organization [B1]



Organization [B2]



Enclave [B3]



Resources on DSP

Figure 5.14 Evaluating EvoSense effectiveness and efficiency

5.5.2.3 Efficiency

Through the presented experiments we illustrate the effectiveness of our approach by comparing the measured resource consumption by our approach with the two other approaches known to have an acceptable defense provisioning levels “the use everything” having almost 100% detection accuracy given our simulator configuration, and the conventional mode mimicking real life defense systems .

Figure 5.14 A1,A2, and A3 show that EvoSense resource consumption is slightly more than the most commonly used solution and much less than the use everything in low attack rates mode regardless of the sharing scope used.

The effect of sharing is obvious, when we expand the sharing scope of defense missions, we managed to detect attacks much faster minimizing attack desperation and excessive resource

consumption. However, with EvoSense, expanding the sharing scope with low attack density might consume more resources than needed due to the consumed resource invested in sharing defense missions that exceeds the wasted resources by low rate attacks.

In the higher attack arrival rate cases, EvoSense was more successful than the other two modes in saving ToD resources. EvoSense use of effective sensing martials lead to more efficient utilization of resources and fast detection of attacks. The reason behind that is minimizing the active period of attacks saves considerable amount of resources too. The effect of sharing is clear in enhancing attack detection promptness and minimizing resource waste, with and we expect that by enabling the immunization effect we can see that EvoSense can even save more resource than the most commonly used solution. Sharing defense missions with non-infected hosts immunize these hosts against future attacks saving unneeded future resource waste.

Figure 5.14 (C1, C2, C3) presents the resource consumption of the three modes on the DSP side. On the DSP side, EvoSense is expected to consume much more resources when compared to the other two modes. The simulator did not consider the cost of analysis that supposed to be added to the ToD when using the “Use everything mode, and the most commonly used solution “ to make the comparison more focused. Adding this cost to the ToD will definitely make EvoSense much more resource efficient than the other two modes. The reason behind that is that EvoSense wave this cost from the ToD to the DSP side. This is why we see that EvoSense is consuming much more resources on the DSP side than the other two modes.

The other two modes resource consumption is mainly the cost of sharing defense missions. That is why we see a sharp decrease in the DSP resource consumption when we minimize the sharing scope with the “Use everything mode, and the most commonly used solution “modes.

On the contrary, EvoSense consume more DSP resources when we reduce the sharing scope

due to the added cost of analysis of attacks that had better chance to spread all over the ToD.

5.5.2.4 Effectiveness

Figure 5.14 (B1, B2, and B3) illustrate the effectiveness of our approach by a comparison between EvoSense and use-everything and the most commonly-used modes regarding “time to detect attacks”.

We can notice that the use everything mode is always better that the two other modes because giving the list of assumptions we build our simulator on, having all sensors deployed on all machines gives the detection tool a 100% success chance in detecting attacks as soon as it hits the host. While based on our discussion before, practically this mode is inappropriate.

EvoSense circulatory defense performs slightly less than the use everything mode, and much better than the most commonly used solution in terms of time to detect attacks. The most commonly used solution use random set of defense missions deployed on all hosts. Unknown attacks would not be detected until the next update round carrying the detection tools. Attacks will have good chance of spreading all over the system.

Expanding the sharing scope massively enhance EvoSense performance in detecting attacks, with a slight enhancement on the other two modes, as this sharing comes in manual exchange of signature update from central servers. EvoSense automated trustworthy sensing and effecting tool sharing acts in much faster way than the signature database update message. EvoSense share executable packages ready to surgically detect and remove attacks quickly and accurately from infected hosts. Sharing such materials instead of sharing signatures enhances the detection accuracy and promptness as shown in the figures.

1) ***The effect of circulation***

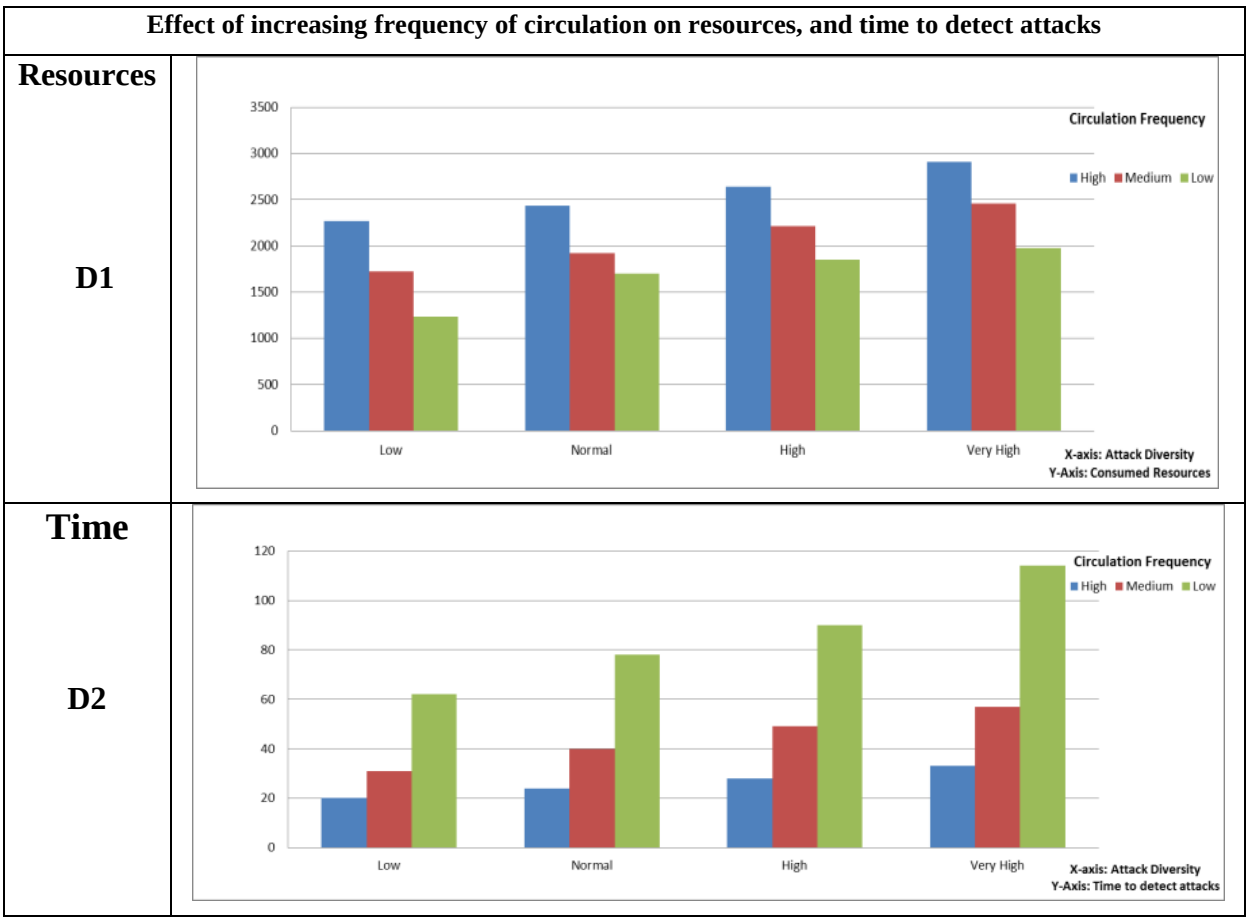


Figure 5.15 The effect of circulation

We carried out another experiment to determine the effectiveness and the efficiency of EvoSense circulatory mechanism, with different sensor/effectors circulation frequency. The experiment evaluates both aspect based on the time to detect attacks, and resource consumption respectively as shown in Figure 5.15 (D1, and D2).

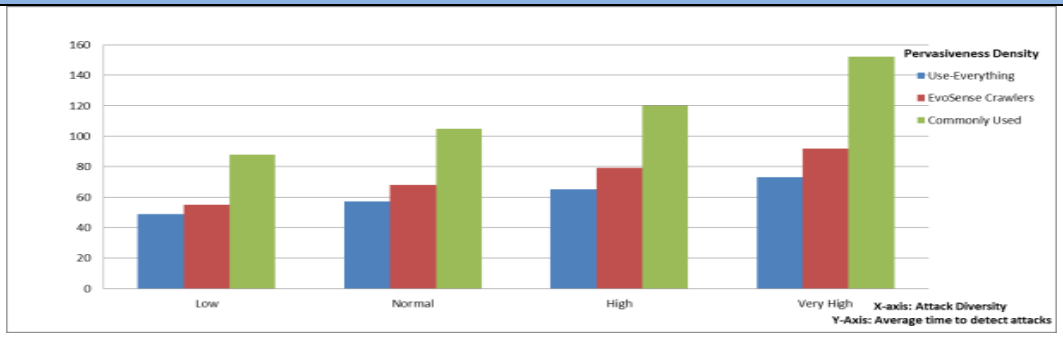
We used three circulation frequency modes high, normal, and low circulation frequencies as shown and highlighted in blue in the parameter Table 5.4. Lowering the sensor/effectors circulation frequency enhances the system resource consumption on the account of increasing the time to detect attacks. In this mode EvoSense is much like the most commonly used mode, as the

sensor change slowly over time mimicking the speed of adding new signatures to the signature database of such mode.

Increasing the sensor circulation frequency, do minimize attack detection time on the account of increasing the ToD resources. This mode is much closer to the Use everything mode as the number of sensors hosted on the host over time is closer to the total number of sensors available. Additionally, the resources saved from not hosting the sensors all the time, is wasted by the load of sensor circulation mechanism if the frequency is too high.

We can notice that EvoSense sensing and effecting circulation and sharing scope parameters can control the spectrum of defense provisioning quality and cost. The parameters controlling these aspects are usually adjusted by the heuristic mechanism in use at the time of deployment and dynamically at runtime. Also enabling the immunization effect and the full-fledged EvoSense with sensor reuse and estimation mechanisms is expected to even save more resources and enhance detection time.

2) ***The effect of distributing defense missions and directing sensor circulation based on matched profiles.***

Average time to detect attacks	
With profiling **E1**	

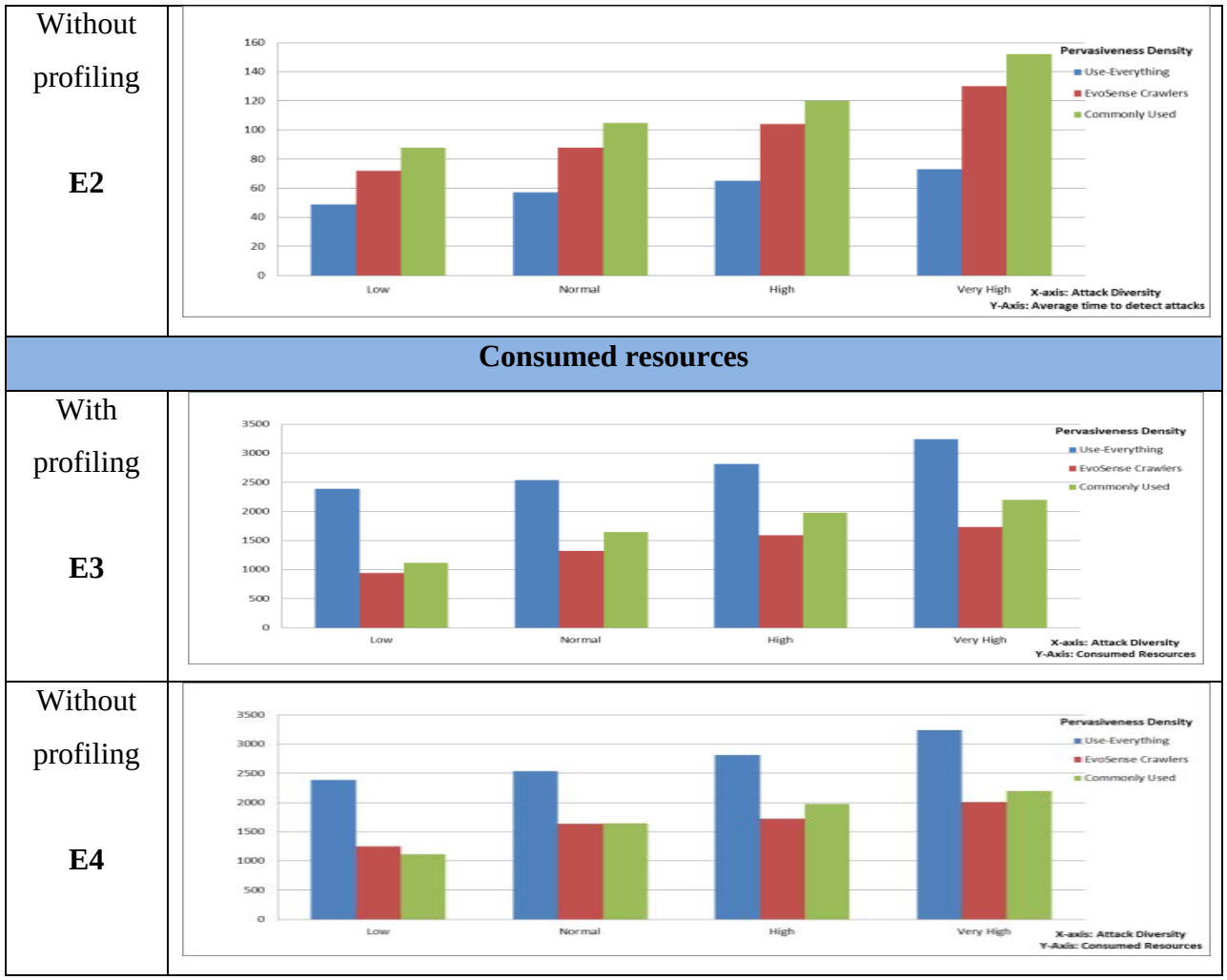


Figure 5.16 The effect of distributing defense missions and directing sensor circulation based on matched profiles

In this experiment we mimicked the configuration of one of the cases presented in Table 5.4, Case (3), while enabling the profiling mechanism of EvoSense. The simulator was modified to create a set of static profiles that describes different behavior patterns for the attached hosts. The profiles were classified into three sup categories, Organization level profiles, Enclave level profiles, and host level profiles. The Organization level profile describes the regular behavior of all the hosts within this organization. The Enclave level profiles describe the behavior of all the hosts working under certain enclave. The host level profile sub group the hosts under certain enclave based on different profiles.

The attacks on the attack pool were classified in categories matching the generated set of profiles. The attack generator was adjusted to direct only attacks that match the profile of the targeted host.

The profiling mechanism used in this experiment presented simple classification profiles that mimics the real system classification based on platform and application configuration. Classifying attacks based on that profiles, was logic given that attacks are mostly targeted targeting specific vulnerability in a certain application running on a specific platform configuration. For example, it is highly unlikely to expect that a windows based attack would infect a network where all hosts operate under Unix OS.

The main objectives of this experiment is to show that EvoSense capability to profile hosts into set of classes, and direct the defense provisioning process based on that profile can enhance the defense provisioning effectiveness quality and efficiency.

The use of such profiles minimized the search space of the investigation elements focusing on a subset of tools matching the subset of the possible attacks. As presented in Figure 5.16 (E1, E2, E3, and E4), doing so minimized the time needed to detect attacks, and the resources consumed by the sensing and effecting elements. The set of resident sensors on each host, is cleverly selected based on such profiles maximizing the success rate of such sensors and minimize the resources wasted by useful sensors.

5.5.3 *Observations*

From the presented results we can conclude by illustrating the following list of observations:

- Using smart sensor deployment increases the attack detection promptness with a reasonable overhead mostly on the DSP side.
  - Most of the workload waived from the ToD to DSP enabling EvoSense to work in

networks with resource-constrained devices

- Increasing sensor circulation frequency and level of pervasiveness has positive impact on minimizing time to detect attacks with a moderate overhead.
  - Resources saved by limiting the attack activity, compensates in part the resource consumption increase due to circulation frequency increase
- Smart sensor circulation based on dynamically changing profiles enhances the detection promptness and accuracy and decrease the ToD resource consumption
- EvoSense detection accuracy is close to the optimal case with a much less overhead that comes closely (on the ToD side) to the resource consumption of the practical case
- EvoSense is a complex defense delivery mechanism with a costly service on DSP side
  - High DSP resource usage invested in;
    - maintaining staple defense provisioning by;
      - enabling defense resilience against attacks
      - Isolating defense provisioning from ToD
    - Global and deep analysis of sensor feedback to detect unknown attacks autonomously
    - Dynamic defense mission composition based on that analysis
    - Constructing global real time view of the entire network to ease management process
    - Inspecting shared material against local privacy rules
- EvoSense was designed to serve large scale applications that desperately needs EvoSense unique features

5.6 **Conclusion**

In this chapter we presented a quantitative study that included a set of models, experiments, and analytical studies to evaluate the efficiency and effectiveness of CyPhyCARD and its pillars.

The conducted experiments and studies illustrated the capabilities of CyPhyCARD pillars to successfully accomplish their design goals with a reasonable overhead. The presented study and results illustrated that:

- The intrinsic adaptive and elastic nature of the basic building blocks enabled each pillar to adapt its resource needs towards efficient utilization of the available resources while maximizing the system performance;
- The effect of the successful and prompt detection and/or mitigation of attacks and threats with ChameleonSoft and EvoSense have a clear impact on minimizing the failure downtime;
- CyberX, ChameleonSoft and EvoSense were able to minimize the attack success minimizing the impact of attacks on resources consumed; and
- Isolating defense provisioning, and moving heavyweight tasks on DSP side, waived most of the workload from the ToD
    - giving it more space to invest such resources on serving the running applications,
    - expand the system compatibility to legacy components,
    - Minimize failures due to resource starvation, and
    - Limit the attacker ability to utilize the defense provisioning workload to interrupt the operation on the ToD hosts in attempt to lunch a DOS attacks.

CyPhyCARD and its pillars were evaluated qualitatively through the dissertation chapters by illustrating their effectiveness in mitigating our synthetic multi-threaded CPS attack, the BlackWidow. The qualitative study demonstrated that CyPhyCARD and its pillars are capable of mitigating such sophisticated attack and invalidating the attacker assumptions and the attack design invariants.

Chapter 6

Related Work

> "Good judgment comes from experience; experience comes from bad judgment." Jim Horning

6.1 **Overview**

CyPhyCARD main objective is to enable efficient, resilient pervasive and prompt attack detection and resolution for heterogeneously composed targets. CyPhyCARD achieves its design objectives by successful employment of its constructing pillars as described in the previous chapters.

CyPhyCARD was founded over a COA based foundation managed by CyberX. CyberX-managed Cell is the basic building block of the entire defense platform. CyberX works on enabling efficient and failure resilient, adaptive application execution by means of application modularization into fine grained components and smart employment of runtime diversity.

Attack resilience is granted by the second pillar, ChameleonSoft that enables runtime software behavior encryption and trace-resistant moving-target defense via complex and smart employment of diversity across time, space, and platform heterogeneity.

EvoSense uses this resilient platform to host the DSP defense services, to ensure resilience of defense service provisioning and to isolate the defense provisioning work-load from the ToD. Further, EvoSense pervasively deliver prompt and precise defense service to the ToD scattered components regardless of its platform or software composition heterogeneity. In the following

subsections we will list the variant efforts that were presented by the current literature to enable CyPhyCARD design objectives.

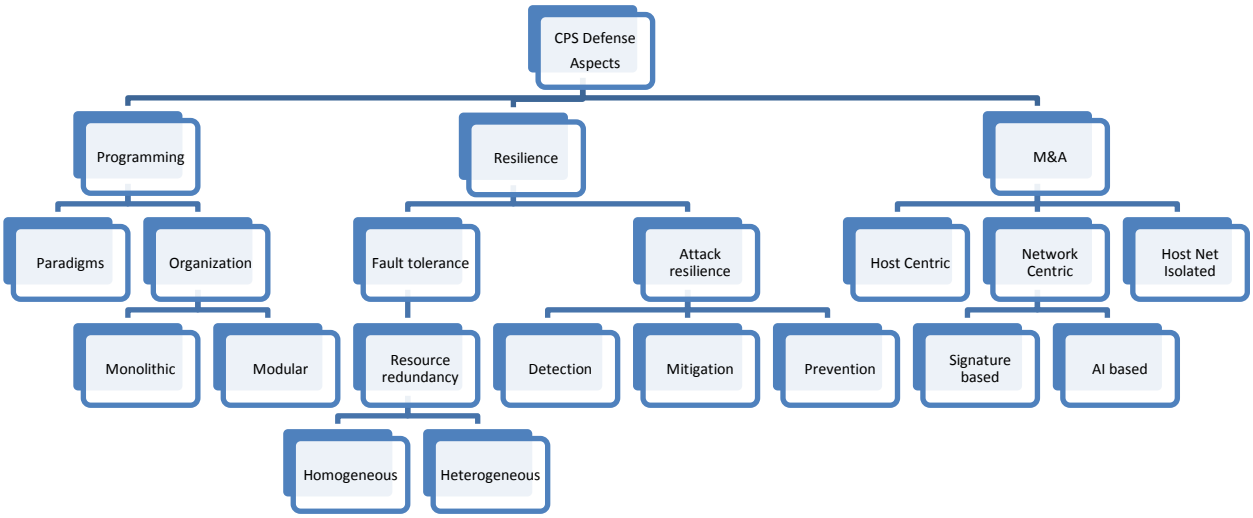


Figure 6.1The Taxonomy

6.2 Taxonomy

We presented taxonomy to lay down the foundation for the review of the literature review. The taxonomy provided focuses on three fundamentals in CPS defense domain, the programing, resilience, and monitoring and analysis domains. We will briefly explore each aspect moving from general concepts to more solid concepts.

6.2.1 *Programming landscape*

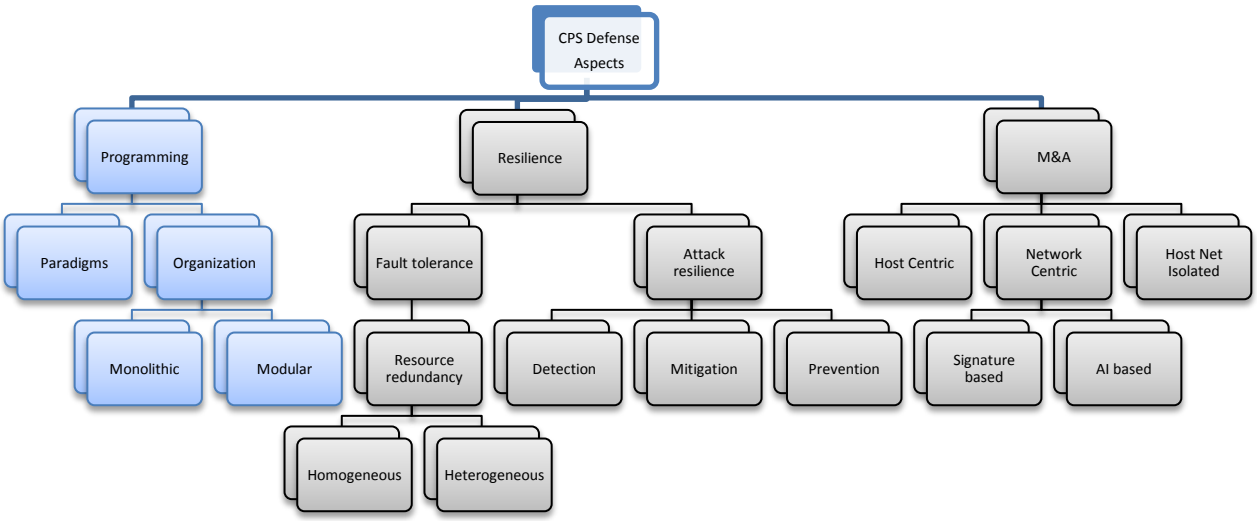


Figure 6.2 The Taxonomy: Programming Landscape

We see the programing landscape should be divided into two aspects, paradigms and organization. Programming paradigms describes style and methodologies used to solve software engineering problems. Programing paradigms vary in the ideas and thoughts used to represent the components of the programs. New technologies work on evolving platform, and applications. However, new paradigms may be needed to enhance the efficiency and the quality of the software development process.

Program organization represents the arrangement used to organize various sections of the program. In this taxonomy we see the program organization as two main classes monolithic, and modular.

Monolithic organization is the conventional technique used to present the programs as a single module. We use this term to describe programs with single image, where software production is a simple cloning process of the exact same module. Monolithic programs or mono culture programing orientation builds programs as single, unstructured, self-contained software units. These programs have serious security and performance limitations as illustrated before.

The modular programing organization describes programs that are constructed from small structures and can be composed to construct the full application. This elastic software design by fractionizing large programs into modules has been used to enhance the software flexibility, reusability, and maintainability. In section 5.3 we will go deeply through the various techniques available to realize such programing organization paradigm.

6.2.2 *Resilience landscape*

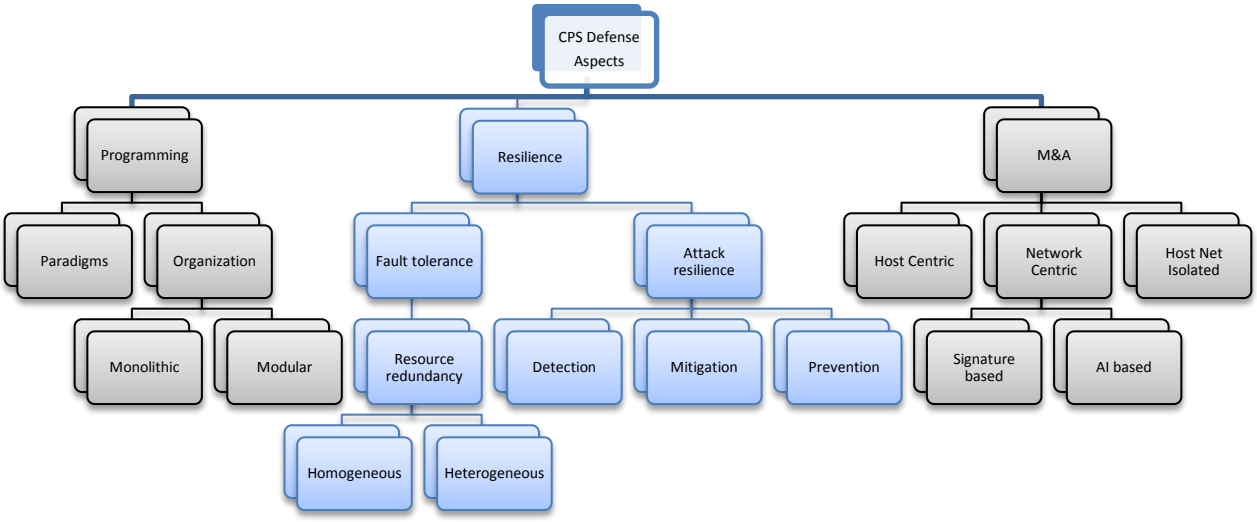


Figure 6.3 The Taxonomy: Resilience Landscape

We define resilience as the ability of systems to autonomously maintain operational stability and integrity in case of attacks, or intentional/coincidental failures. A resilient software product is a product that can autonomously mitigate or block attacks, or a product with the ability to autonomously heal from the effect of an attack or failure with minimal operation-interruption. We see two main classes under this category, attack resilience, and fault tolerances.

Fault tolerance, is the ability of software or hardware to autonomously handle intentional or coincidental failures and autonomously restore operation with minimal downtime. Redundancy and replication are the main techniques being used to provide fault tolerance. Redundancy

includes duplications of data, logic or physical resources. The key element is how to detect failure quickly, and how to minimize the time needed to restore operation with minimal resource waste due to duplications. Section 5.4 illustrates the successful attempts of employing diversity for fault tolerance and attack resilience. Attack resilient products are products that can detect and autonomously mitigate attacks, or products with mechanisms to prevent certain attack classes. Attack detection techniques vary by the variation of the application and the type of attacks that they were design to detect, and the detection mechanism. Section 5.4 and 5.5 gives a deep illustration of the different mechanisms available to provide attack resilience.

6.2.3 *Monitoring and Analysis (M&A) landscape*

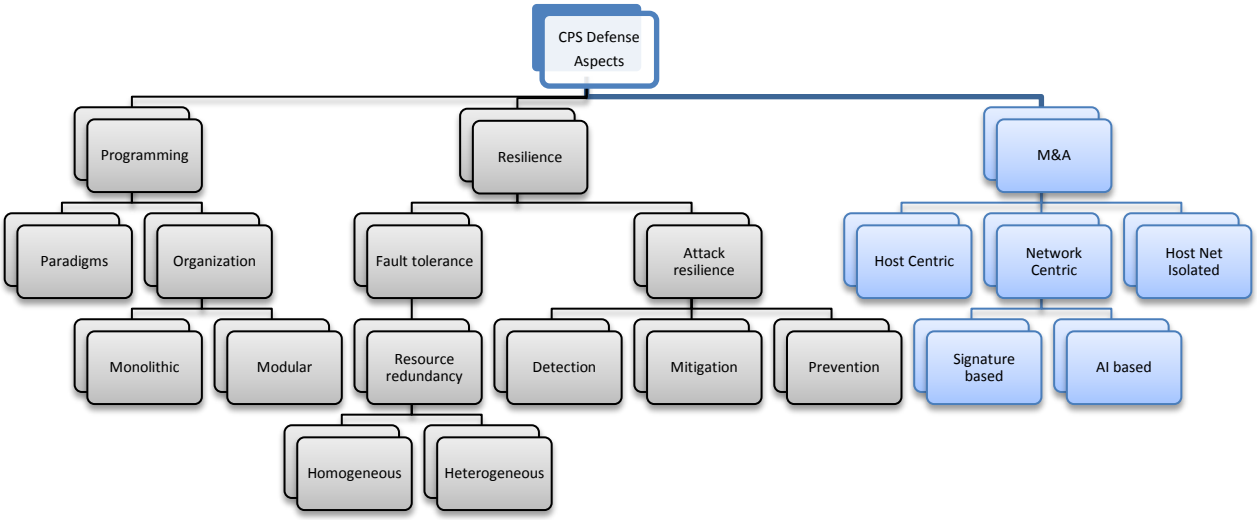


Figure 6.4 The Taxonomy : M&A Landscape

M&A landscape focuses on the mechanism provided to facilitate host monitoring and analysis for security objectives mainly. We classify it based on three main classes. Host centric, network centric, and host-network isolated. The host centric defines mechanisms designed to share the same host that it was design to monitor and analyze. These techniques use the same host resources to provision its services. Network centric approaches are approaches that were designed to provide monitoring and or analysis services via remote nodes that share the same network with the host. These mechanisms usually rely on fully or partially host-resident

applications to execute its tasks. These applications share the same resources with the host, and execute certain missions provided by remote nodes sharing the same network with the host. All the elements have to operate within the same perimeter to protect the host privacy. Mobile agents have long been used to establish such M&A mechanism as presented in section 5.5. The last class is a host-network isolated mechanism that is designed to provide M&A services in total isolation from the host and the host-network. Section 5.5 provides deep illustration of the different mechanisms available to serve under these classes.

6.3 **Elastic software design**

6.3.1 ***Software modularization***

CyberX is designed to manage COA-based systems to enable constructing elastic, dynamic, and adaptable software products with intrinsic support for situation and context aware fault tolerance. Currently software products depend mostly on static or partially dynamic architectures where data, logic, and/or physical resources are primarily tightly coupled. Multiple attempts have been presented in the literature to partially decouple these design concerns through what is termed as application modularization.

COA separates the main design concerns through an intelligent modularization of the application into a set of Cells. The application represented by an Organism is modularized into set of Cells. There are different techniques along literature that worked on application modularization for different objectives. In this section, we will illustrate the main approaches working in the field of modularizing the application into compostable components that can adapt to certain aspects.

COA modularizes the application in terms of Cells, Service Oriented Architecture (SOA) modularizes the application in terms of services, Object Oriented Architecture (OOA) modularizes the application in terms of Objects, and Aspect Oriented Architecture (AsOA) modularizes the application in terms of Aspects “quality attributes”. [4, 5, 6, 18]. An application module is sometimes called components. OOA or AsOA, SOA modularize software systems into set of components [16].

Component Oriented design was introduced to create independent entities for different modules in a software application. [16] “Define a software component as unit of composition with contractually specified interfaces and explicit context dependencies only”.

Generally speaking, a component can be represented as a closed composable box reflecting certain functionality, and behavior at runtime and with interfacing capability through a clearly defined inputs and outputs [18]. The component can communicate with other components and the surrounding environment through such interface. A clear characterization for the component was defined in [16].

Several versions of the component modularization were presented industry wise and as a research work. For example, the COM [21] from Microsoft, the EJB specification from SUN [22], CORBA [23] from the OMG, etc. Additionally the work presented like (Fractal [24], SOFA [25], etc. is a good research work related to software modularization. Fractal was one of the approaches that enable the component to modify its internal structure during the execution. The program architecture can be modified at runtime enabling the application to dynamically change at runtime.

Unfortunately none of these solutions considered the real meaning of adaptation to changes at the application or the infrastructure level. Enabling the application to communicate with the

infrastructure to support its dynamic needs and to inform the infrastructure about the internals of the execution process was not presented before CyberX COA. Additionally, the presented approaches did not realize a full separation between design concerns. The best available solution managed to partially isolate Data from Logic, while program components were always resource oriented. However, none of these approaches investigated the concept of intrinsically resilient component. CyberX Cell is a system by itself, a complex component with the ability of self-adaptation and decision making, fully situational aware, smart, and resilient. The following subsection provides more details about the main modularized software architectures.

6.3.2 *Modularized software architectures*

Aspect Oriented Software Architecture (AsOA) is one of the well-known software modularization architectures. AsOS refers to a set of emerging mechanism that defines methods of modularizing software systems [18]. The concept of modularization started with Parnas in the seventies [16]. Parnas defined modularization as the process of isolating and localization of quality attribute objectives. A quality attribute objective can represent any interest that the developers might care for about a system. Quality attribute objectives can include high-level objective, like security, robustness, or reliability. Low-level quality attribute objectives represent technical aspects like caching and synchronization [24].

Separating such quality attribute objectives enabled programmers to focus on small modules, which improved the overall application quality and minimize the chance of failure due to attacks or design faults.

Separation of quality attribute objectives is an efficient way for software designers to effectively split the application objective or the problem that the application is designed to solve into multiple isolated modules that targets specific quality attribute objectives.

Object Oriented Programming (OOP), for example, is one of the techniques that works on the concept of quality attribute objectives separation, by fractionizing the entire application into a set of objects that targets specific functional quality attribute [25].

Aspect oriented programing was defined in 1996 by Kickzales and his group at the Xerox PARC research center [24]. It was an enhanced version of the OOP to complement it in order to obtain applications that are clearer and better structured [26].

Service Oriented Architecture (SOA) is a standard to design software applications based on services that interact with each other. In [27], the authors define SOA as "a paradigm for dealing with business processes distributed over a large landscape of existing and new heterogeneous systems that are under the control of different owners." SOA aims at facing several challenges; like interoperability and heterogeneity. Heterogeneity refers to variation of resources, geographical location of service provider, consumer, system developers, and owners.

SOA as a standard does not apply to a specific technology. The most mutual application example of SOA is Web Services [18,28]. Web Services are a way to establish a SOA solution by using a specific implementation strategy.

The Service Component Architecture (SCA) was developed as a more established version of the SOA. SCA provides platform to achieve delivery, support, and management of distributed applications compliant with the rules of SOA [29]. SCA utilize software components to device services.

SCA is a set of specifications defining a formal method for developing application using SOA. It is endorsed by many well-known software manufacturers including, IBM, Oracle IONA, BEA, SAP, TIBCO and Sun.

SCA highlights the decoupling of service employment and of service assembly from the details of infrastructure abilities and from the details of the access methods used to invoke services [18].

The SCA specification supports service implementations designed via many programming languages, including declarative languages such as XQuery and SQL. SCA also supports a many programming styles, including asynchronous and message-oriented styles, in addition to the synchronous call-and-return style [29]. Also it includes object-oriented and procedural languages such as Java, PHP, C++, COBOL; XML-centric languages such as BPEL and XSLT.

Up to our knowledge our COA is the only architecture that comprehensively supports intrinsic separation of design concerns needed for runtime re-programmability, intrinsic autonomic online composability, and dynamic software adaptation and elasticity.

Attempts were presented towards enabling some of these features separately. Agent Oriented Architecture (AOA) utilized autonomic building blocks while SOA and OOA used non-autonomic components. Using autonomic building blocks facilitated supporting non-deterministic behavior change in AOA by explicit use of soft computing as presented in [8]. However, supporting online composability is not clear in AOA, while in OAA and SOA it is enabled either by aggregation [9] or by service composition [10].

The COA Cell separates logic from physical resource management by constructing an intelligently-managed elastic thin virtualization layer between the application and the underlying physical resources. Such construction facilitates unifying the execution platform for distributed applications regardless of the configuration of the host platform. Unifying the execution

environment waives the load of building platform/OS specific application for each targeted platform. In addition, the maintainability issues are divided between the developer and the technology owner. Software developers are concerned with maintaining the application itself, while the technology owner is responsible for maintaining the execution platform. Partially elastic virtualization approaches were presented for loosening the bond between physical and logical resources; where applications are partially compiled at the production phase to be executed over virtual machine host [4,16]. These techniques can be used to build a uniform execution environment for distributed applications. However, these approaches presented static elasticity and partial separation of design concerns. They did not separate data from logic and physical resources. Such separation is a key enabler for supporting intrinsic fault-tolerance, live-mobilization, and runtime adaptation to frequently changing execution environment. Our approach provides an intelligent elastic virtualization utilizing mobile software capsules (Cells) that gets specialized at runtime facilitating online re-programmability. This feature when managed by CyberX enables COA Cells to seamlessly move between heterogeneous hosts, while autonomously adapting to any resulted changes. Additionally, CyberX-managed COA Cell can encapsulate different code variants and switch between them at runtime. CyberX utilized this unique feature to enable runtime manipulation of targeted quality attributes. Doing so, facilitates real-time adaptation to execution environment changes optimizing the application performance, resource-utilization, and enhancing its reliability, survivability, and compatibility. Based on our knowledge utilizing any of the available virtualization techniques to enable such features were not possible prior to our work. The next subsection focuses on approaches employing component diversity techniques for quality attribute manipulation.

6.4 **Diversity employment for security, performance, and adaptability**

Component diversity was investigated in Genesis [11], were the idea of providing both design diversity in the form of multiple variants representing different designs of the same specification as well as data diversity were proposed. Compiler guided code variance approach [12] aimed to present automated massive-scale software diversity by the help of automated variant generation and utilizing multi-core platforms. More advanced diversity employment approaches with the objective of anomaly detection through detecting flow deviation but with fewer constraints were presented in [13, 14]. A major drawback of such solutions is the need for virtualizing every input to the whole set of executing variants at the same logical point to be able to detect the abnormal deviation of the execution flow.

Based on our knowledge utilizing runtime hot shuffling of software variants for quality attribute hot manipulation was not previously investigated. Additionally, failure recovery mechanisms were not investigated as most of these solutions presented static diversity with low probability of failure. None of them investigated the idea of a comprehensive solution that provides elastic, autonomous, resilient, situation-aware platform targeting different quality attributes, while dynamically shuffling its software components to suit changes in the surroundings. Another drawback of these solutions is the massive use of resources to realize diversity using heavy virtualization techniques and multicore or multiprocessor platforms. The following subsections introduce variant techniques for diversity employment for different objectives.

6.4.1 ***Design time diversity***

Software diversity has a long history of research work in the field of software security and fault tolerance dated back to the 70's [4]. Basically software diversity was presented as multiple independent solutions for the same problem. The realization of that is to develop multiple independent versions of a program with different teams using different languages. The main goal of this approach was to increase the attacker confusion by changing the behavior of the software, which will make system exploitation harder. They expected that at any given time the majority of these versions will be working correctly [4, 5].

Some research work showed that there is a high probability that a multi-variant software approach might face many coincidental failures [6, 7]. On the contrary other research work suggested that from the cost and the reliability point of view, the multi-variant approach is much better than the one "good" version, especially in mission critical applications where the cost of failure could be very high [30].

Design time diversity aims to device the same software in multiple designs to diversify the software product [31, 32] the objective was to defeat the mono culture of software development, and to increase the attacker search space for vulnerabilities. Different techniques[34,35] were designed to automate inducing light changes in the software product at development time. The basic idea is that the diverse software replicas maintain the same functionality, but differ only in their implementation details.

The main problem facing this approach was the fact that it is static, and it can easily be predicted by runtime analyzers working on the attacker targeted field of operations.

6.4.2 *Load time diversity*

System call randomization is a good mitigation mechanism against a wide set of code injection attacks; it aims to randomize the mapping of system calls [12]. The attacker mission to counterfeit such defense relies on guessing the system call numbers. The main issue is that the realization of this technique requires kernel recompilation with the new randomized system call mapping, and it necessitates that the binaries are rewritten to reflect the new system calls. These requirements invalidate such approach, in addition to the fact that static redesign of the kernel is a very complicated task [18]. Even with dynamic instrumentation [24] it still considered impractical due to the excessive overhead. Additionally, it is a static solution that works only against one class of attacks and is not valid for other classes of attacks.

Pointer randomization, this approach works on randomizing the stored pointer representation values. The work presented in[12] is a good example for such mechanism. The authors perform an XOR operation on the pointer values with a random integer mask the gets generated at the bootstrap time. This mechanism works on mitigating attacks targeting corruption of pointer values. Attackers trying to mitigate such attack have to guess the value of the random integer mask used at bootstrap time to device the desired pointer value for corruption. The main disadvantage of such solution is it is static randomization. The values remain the same after bootstrapping and can be analyzed or guessed by the attacker with tools working on the same host especially for long lived applications. Additionally, it works only with one class of attacks. Attacks like buffer overflow for example cannot be mitigated with such mechanism. Further, it is useless with languages that does not provide accurate type information, or languages working with un-typed buffers. With such languages the corresponding pointer value(s) cannot be protected.

Address space layout randomization is one of the most successful and most commonly used mechanisms in many operating systems. Multiple implementations were presented to realize address space layout randomization [37,39,38]. These approaches focused on randomizing the base address of memory sections. It works fine with some attack classes like buffer overflow attacks, while it share the same problem of static diversity approaches. These mechanisms provide static diversity that can be detected by resourceful attacker with tools executing on the same machine running the targeted software.

6.4.3 *Runtime diversity*

Diversity has been realized in various ways. Some work presented it in the form of confusion induction paradigm [41,40] where diversity is used to confuse the attack in order to complicate the attack process. An example for leveraging diversity for confusion induction is presented in the form of a load-time binary transformation as the one mentioned before and the one presented in [42]. Others presented different solution for diversity realization based on virtual machines called "private machine architecture" [43]. They used randomization to promote heterogeneity at the machine level aiming to increase the cost of broad-based binary attacks. Moreover, some commercial operating systems realized the ideas of operating system randomization [44, 45]. System call mappings, global library entry point, and stack placement randomization were used to induce diversity as mitigation for buffer overflow attacks.

Component diversity was investigated in Genesis [35], were the idea of providing both design diversity in the form of multiple variants representing different designs of the same specification as well as data diversity were proposed. Data diversity uses multiple copies of a single implementation operating on different data inputs but yielding the same desired results.

Massive-scale software diversity was presented by the help of automated variant generation and utilizing multicourse platforms. Compiler guided code variance approach aims to present such automation [41]. A realization of this massive-scale software diversity approach for the purpose of detecting anomalies by replicated execution was first presented by [11, 12, 50] they mixed diversity with parallelism and check pointing. They execute different variants of a program in a muti-core environment while monitoring any deviation in the program flow to issue an intrusion alert.

A major drawback of existing solutions is the need for virtualizing every input to the whole set of executing variants at the same logical point to be able to detect the abnormal deviation of the execution flow. More advanced approaches with the objective of anomaly detection through detecting flow deviation but with fewer constraints were presented in [51,52,13,54].

These approaches generally apply different types of diversity mainly for reliability by replication or for intrusion detection by program flow deviation detection at runtime. Based on our knowledge utilizing runtime hot shuffling of software variants for behavior encryption was not previously investigated. Further, existing solutions used diversity to target specific quality attribute. Failure recovery mechanisms were not investigated as most of these solutions presented static diversity with low probability of failure. None of them investigated the idea of a comprehensive solution that provides elastic, autonomous, resilient, situation-aware platform targeting different quality attributes, while dynamically shuffling its software components to suit changes in the surroundings. Another drawback of these solutions is the massive use of resources to realize diversity using heavy virtualization techniques and multicore or multiprocessor platforms. ChameleonSoft is designed to support legacy systems with limited resources. It can dynamically tailor its tasks to suit the dynamic change in resource availability.

6.5 Attack detection and resolution

6.5.1 *Malware detection*

A malware is malicious software designed to infiltrate or damage a Cyber system or Cyber Physical System (CPS) without the owner's informed consent [107]. There are many malware types with different shapes and entry points. Most of these software objects share similar purposes while they are expected to behave differently at time of infection. **Viruses**, worms, **botnets**, wabbits, Trojan-horses, exploits "backdoors", spyware "scumware, stealware, parasiteware, adware", **rootkits**, **blended threats**, **evolving threats**, keyloggers, hoaxes are examples of the different malware types. Figure 6.5 lists the different types of attacks and the usability ration of each one of them [33].

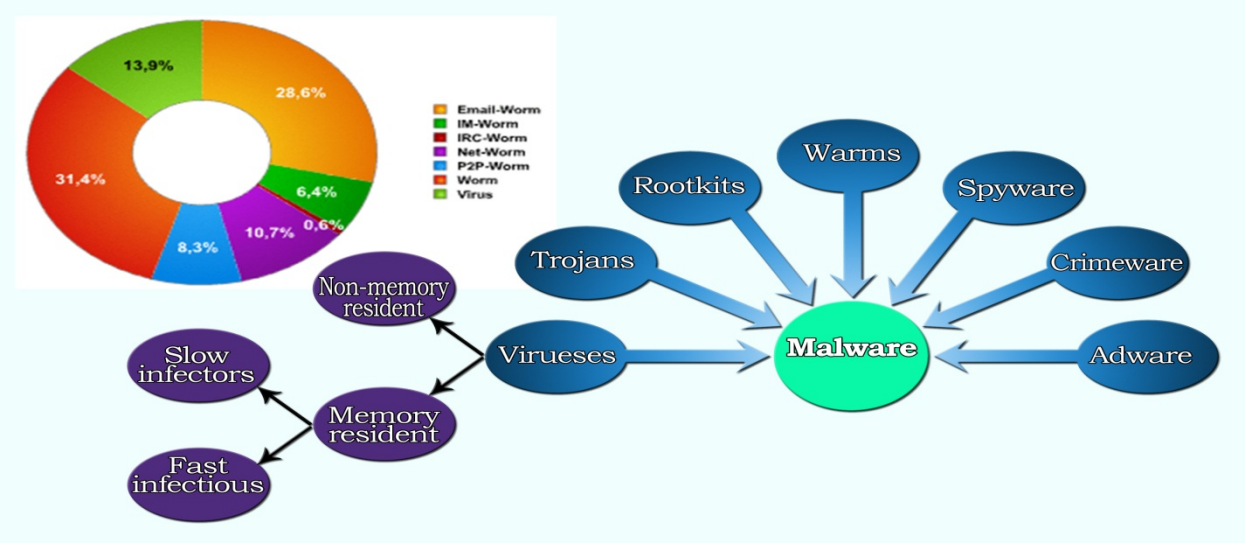


Figure 6.5 classification of malware related attacks

Each malware group has its own way of being undetected. Modern malware detection tools utilize multiple detection mechanisms to be able to detect multiple malware categories as presented in Figure 6.5. Malware especially viruses are either memory resident or non-memory

resident. Non memory resident are simple attacks that can easily be detected an entry point with a cleaver detection tool.

The memory resident attacks are more complex and efficient that stays in memory and hides their presence from detection tools. These attacks are either fast infectious aiming to infect as much files as possible locally within the infected host or remotely through the host network, and network shares. The second category of memory resident attacks is the slow infectors. Slow infectors are the most dangerous type of malware as it uses stealth and encryption techniques to stay undetected as long as it can. They are powerful attacks that can be a combination of multiple processes working together towards certain objective.

Malware detectors use signature based detection techniques to detect known attacks. Signature based detection became very efficient way of detecting known threats [49]. Finding a specific signature in one of the executable codes can accurately identify any enclosed threats within such code. Attack signatures are frequently updated and stored on the local anti-malware database. Unfortunately this technique is inefficient if the attack has a malformed signature either by the programmer or by a mutation engine.

Heuristic techniques are one the most efficient ways to detect such mutated attacks. Heuristic and metaheuristic techniques are used to spot unknown or known attacks with polymorphic behavior.

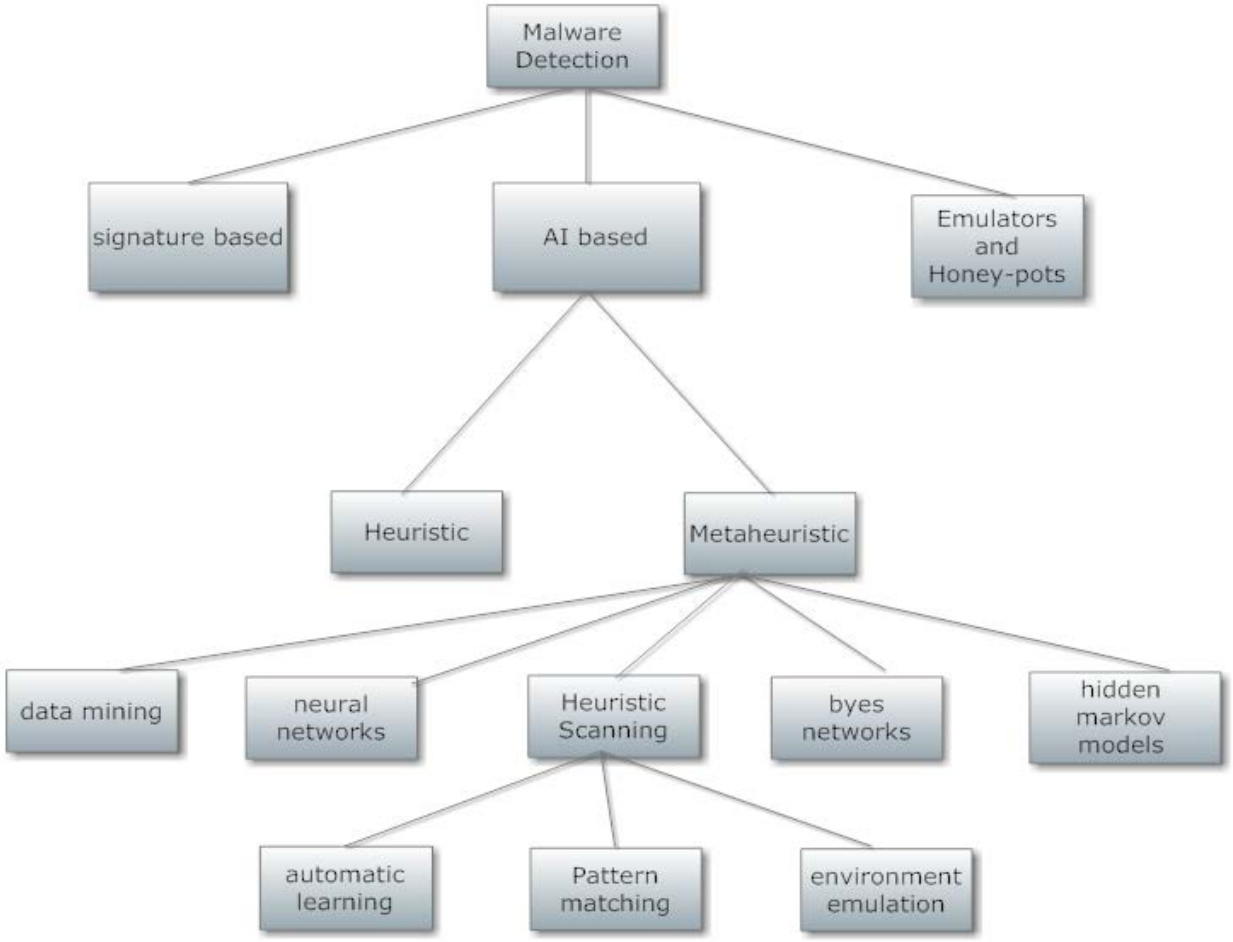


Figure 6.6 Classification of malware detection mechanisms

By definition, heuristic technique is an informal technique to solve problems efficiently and in a way close to the optimal path [49]. Heuristic techniques are commonly used to rapidly reach a solution that is somehow close to the best possible solution. The metaheuristic technique is a heuristic method for solving many of the computational problems by combining user-given black-box procedures in a hopefully efficient way [49].

Most of the modern malware detection techniques that use metaheuristics to detect attacks utilize a set of isolated tools utilizing different techniques hoping in detecting one of the attacks that

there is no specific way to detect it. Most of these tools utilize one of the following mechanisms, Pattern matching, automatic learning, environment emulation, neural networks, data mining, byes networks, and hidden markov models. There are other metaheuristics techniques but most of them are built based on one or more of the aforementioned mechanisms.

The main concept of heuristic based detection techniques is to detect attacks without knowing too much about its internal structure. Heuristic techniques mainly focus on examining the behavior and the characteristics of the executing software to anticipate whether it is acting maliciously or not. The most successful heuristic based detection technique named as The Heuristic Scanning Technique utilizes a mixture of multiple metaheuristic techniques such as pattern matching, automatic learning, and environment emulation.

Heuristic scanning in the common sense uses pattern matching to examine the assembly language instruction execution sequence, and qualifies them by their potential dangerousness. Heuristic scanning usually follows a set of built-in rules with pre-assigned weight on each rule. In case of violation of any of any of the rules the weight of the violated rule is added to the total violated rule by the same program or process. The program is flagged as malicious only if the total sum of added weights exceeds certain threshold. Figure 6.7 illustrated the idea of a single layer classifier with predetermined threshold.

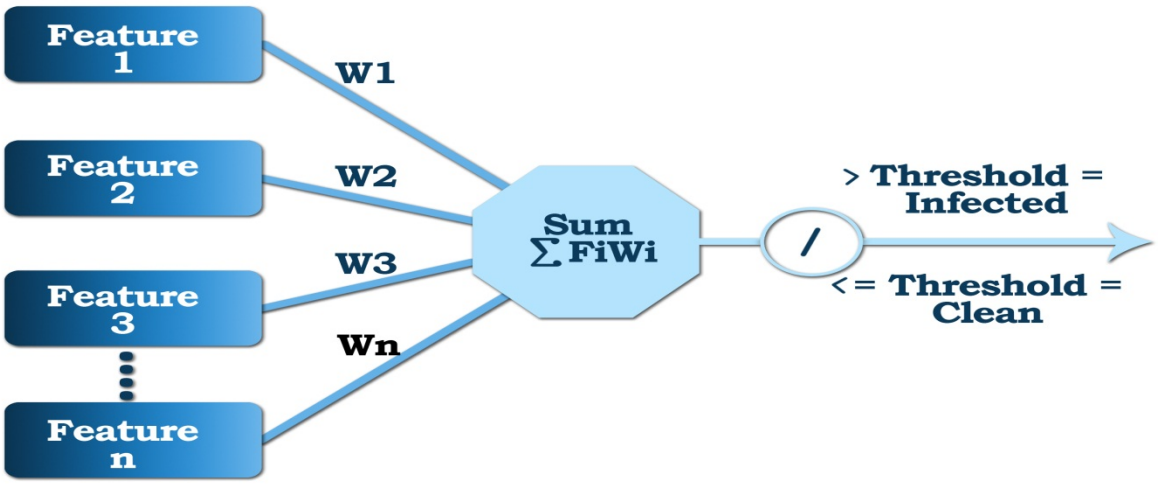


Figure 6.7 Single layer classifier

The feedbacks from the different scanners are fed into global summarizing point that follows a certain metaheuristic mechanism as illustrated in Figure 6.7. The overall result will decide whether to flag the scanned object or not.

As the detection techniques gets more cleaver, the modern attacks or malware also emerge to more complicated attacks utilizing more sophisticated stealth techniques. Such techniques give them the advantage of being invisible to traditional scanners. Moreover the use of real-time encryption, and anti-heuristic sequences made them looks totally harmless to traditional malware scanners.

Heuristic scanners that use single metaheuristic mechanism that focuses only on monitoring the execution flow of the instructions of a certain program are deceivable by code obfuscation. Code obfuscation occurs by embedding some meaningless instructions within a malicious code. The same technique deceives detectors utilizing heuristic and signature scanning combined together.

One of the successful mechanisms to resolve the aforementioned problem is the use of artificial runtime environment emulation. However, it is not a light weight detection mechanism, but it has

high success rates in detecting unknown attacks. Environment emulation utilizes the idea of virtual machines; the malware detection tool provides a virtual machine with independent and isolated operating system and allows malware to perform its routines freely within the virtual environment. The execution behavior of the suspicious application is being continuously examined while the malware is not aware. Most of the stealth and anti-heuristic techniques are irrelevant in this case, as the detection tools scan the behavior from outside the box with a clear vision of what is really happening inside.

The main problem facing such technique is the massive resource consumption and the expected delay needed to construct the virtualization environment, and infiltrate the harmful instructions from being executed on the real machine.

Another problem that arises with using heuristic methods for detecting malwares is the possibility of false positives. A false positive event occurs when a benign program gets flagged as malicious by the heuristic scanner. The problem occurs frequently specially with noncommercial programs having suspicious routines through their encryption functionalities.

The use of automatic learning is a good resolution of such problem, where the detector learns from its mistakes. The main issue with this technique is it requires an advanced user. In order to resolve such problem autonomically, detection scanners have to increase their scanning depth, and combine feedback from multiple heuristic mechanisms. Also external consultation is one of the most efficient techniques, where an external resourceful node gets consulted for guidance related to suspicious programs with weights that parley cross the threshold line. The only issue with that solution is the possibility of privacy violation due to sending specifics about the suspicious events.

Recently more complicated attacks were introduced that depends on infecting and controlling multiple hosts creating an automated taskforce targeting multiple objectives. Such attacks usually have dynamic objectives, and construction components. Additionally, they are frequently and autonomically get updated using a dynamic up/down link between the attacker and the malware itself. Detecting such attacks is a very complicated task given the uncooperative nature of the conventional modern detection tools, and the fact that they share the same host, or host network with their ToD.

Sharing the same network or host with the ToD makes them an easy target for attackers to deceive, or destroy [46,71]. Additionally, the successfulness of the malware detector depends mostly on the fast real-time, and deep analysis of the scanners feedback. Such process, especially when it involved creating a runtime emulated execution environment is a computationally costly process for a tool that shares the ToD resources.

6.5.2 *Standalone and distributed monitoring and evaluation solutions*

Defense services for CPS are highly dependent on the promptness and accuracy of the Monitoring and Analysis (M&A) mechanisms employed. Traditional M&A approaches do not treat sensing and effecting for cyber components and physical components seamlessly. The current M&A mechanisms were designed based on a set of assumptions that unintentionally neglect the real-time interaction and the tight coupling between these converging components. The **assumption** was that physical components were protected by isolation and parameter defense while real-time response was not a primary factor for cyber components. Further, they assumed that there is no need to employ privacy preservation techniques as the Target of Defense (ToD) privacy is implicitly protected by cyber and physical parameter defense. Additionally, they

assumed that resource heterogeneity and scale could still be resolved by a distributed set of heterogeneous, pre-deployed platform-dependent defense tools with fixed resource profiles.

Research works in [83,84] as well as our own have disputed the validity and correctness of such assumptions as they lead to drastic **problems and limitations** negatively impacting the quality and promptness of the CPS defense service provisioning. Current CPS Defense Service Providers (CPS-DSPs) fail to provision trustworthy robust and reliable **monitoring and evaluation** of the ToD components due to the use of scattered, uncoordinated, uncooperative, unaware, isolated and heterogeneous monitoring tools, and reporting mechanisms. Such limitations increase the use of resources due to redundancy, increase the risk of conflicts, and failures due to limited awareness and coordination, lower the defense quality due to the poor, and boundary limited feedback, increase the latency in defense provisioning and in detecting attacks giving the attacker the advantage to spread the attacks through multiple networks, the tool heterogeneity and uncooperative nature massively complicates automating its management, the static nature of such tools complicates attempts to autonomously adapting to changes in the surroundings.

Research presented in [87,88,89,90] attempted to resolve some of the problems resulting from such assumptions using more flexible sensing and control elements. They devised a mobile multi-agent based attack detection system. The presented solutions were situation unaware and offered limited defense-tools pervasiveness and coordination. Generally speaking, provisioning defense services while sharing the same host with the ToD exposes the ToD to DoS attacks, and limit the system's scalability and interoperability.

Works in [89, 90] utilized a multidisciplinary approach to intelligently resolve some of the presented limitations. They combined multiple artificial intelligence techniques to build a

complex smart attack detection system. Unfortunately, these techniques were bounded by the available technology constraints; they were designed to provision dedicated defense service while sharing the ToD host or host network. They were unable to overcome the curse of complex systems dimensionality. With the increase of system complexity and numerousness of input features, the processing time involved with clustering system events might badly affect system, and attack detection timeliness. Time constraints may sometimes force the system to prune less important features (dimensionality reduction) to maintain system timelines. However, the pruning approach is not always possible as it might compromise the detection accuracy.

All the above mentioned approaches were mainly concerned with defense service provisioning for cyber components. The work presented in [91, 92] is a hardware based static detection system capable of supporting the requirements of both cyber and physical components. Using hardware based detection and analysis techniques guarantee prompt, and resource efficient response for quickly spreading attacks. A major disadvantage of technology is its limited flexibility, adaptability, interoperability, and maintainability. These systems are designed to work for specific target and cannot seamlessly adapt to match different targets.

Multiple attack detection solutions were presented utilizing mixtures of the abovementioned methodologies employing different M&A techniques [93, 94], Unfortunately, none of these systems where capable of presenting a comprehensive, autonomous, interoperable, globally situational aware and scalable solution that can guarantee adequate defense provisioning quality and promptness while maintain the ToD survivability, operability, and privacy. Up to our knowledge EvoSense is the first solution that can provide such features comprehensively and pervasively with low overhead.

6.5.3 *CPS related control solutions*

In addition to the limitations presented in the previous two subsections, in regards to monitoring and evaluation, and analysis of feedback, the control phase; where the defense system takes actions regarding detected threats face a serious set of limitations [71]. The limitations are mainly due to the lack of cooperation and awareness that limit the defense tools capability to resolve or even contain persistent fast spreading attacks.

For example, it is too hard for such uncoordinated, scattered tools to marshal and coordinate task force to hunt down the attacks spreading all over the network or a set of interconnected networks as it is hard to control the DSP, and the ToD tools and equipment to block attack access to the shared network. Further, without appropriate global control, and situational awareness too hard to block the source of dynamic remote attacks. Such limitations can be utilized to cause DoS attack by keeping the DSP busy treating infected files and strike more and more files.

Research work has been focusing on presenting a resolution for some of the control problems in CPS environments. Researchers in [75] presented what is called Autonomous Multi-agent Cooperative Problem Solving (TEAM-CPS), and successfully applied it on one of the critical CPS, the public telephone networks. They used multi intelligent agents that were designed to work together to provide distributed control for such system. Unfortunately, the system was not scalable enough to suit large scale systems. The limitations against this approach and other agent passed approaches like the work presented in [53,72] is the high resource consumption nature of the agents, and the fact that they are designed to share the host resources. These limitations limit the approach capability to scale.

From another perspective, the use of intelligent agents lacks the support of the physical part of the network. The used agents are not aware of the interactions between the cyber and the physical parts of the system. Such unawareness increases the chance of conflicts, errors, and failures.

A more advanced version of this line of research was resented by the work of [71,53] as they used multiple AI techniques to control a pool of mobile agents performing control tasks. The use of AI guided the management platform towards smarter decisions. Unfortunately, they shared the same problem of their insisters, the lack of situational awareness, and the inconsideration of isolating the control platform from the host under control. Such limitations limited the scalability of such systems, and their ability to suit CPS applications.

6.6 **Conclusion**

In this chapter we presented an overview of the latest efforts that were presented by the current literature that can be utilized towards the realization of CyPhyCARD design objectives. We illustrated the various techniques available to enable software elasticity needed to facilitate efficient and dynamic adaptation to changes within CPS domain. Additionally, we presented the various techniques available to enable software diversity that can be utilized to realize moving-target defense for platform security. Finally, we presented the different attack detection and resolution mechanisms being used within the cyber and CPS domains. We observed that despite the existence of solid and concrete research base addressing these various design aspects, these solution fall-short to realize the needed level of quality, efficiency, and effectiveness to support the CPS defense cloud presented here. In addition to the efficiency and effectiveness limitations of the presented solutions, these solutions were not designed to be composeable or cooperative facilitating the construction such large defense platform like CyPhyCARD. Additionally, these solutions were not designed to satisfy CyPhyCARD's targeted field-of-operation needs and characteristics. Based on our best of knowledge, the presented pillars independently or combined together under CyPhyCARD umbrella, they present unique efficient and effective solution to a set

of CPS security challenges that were not previously addressed, inadequately-addressed, or addressed-with-serious-limitations by other solutions.

Chapter 7

Conclusion and Future Work

> "I came to the conclusion that I am not a fiction writer." Tim LaHaye

7.1 **Conclusion**

In this dissertation we presented CyPhyCARD platform that provides the means to guarantee continuity of operations as well as deter attacks and prohibitively increase the cost on potential attackers. CyPhyCARD efficiently coordinates defense missions and tools in real-time to accomplish the following objectives:

- Achieve asymmetric advantage to CPS defenders, prohibitively increasing the cost for attackers;
- Ensure resilient operations in presence of persistent and evolving attacks and failures; and
- Facilitate defense alliances, effectively and efficiently diffusing defense intelligence and operations transcending organizational boundaries.

CyPhyCARD presents a unified resilient platform to monitor, manage, and control the heterogeneous composition of CPS components. Such unification of control with the help of CyPhyCARD autonomous management capability expands the applicability of such system in multiple domains related to cyber and CPS.

CyPhyCARD uses its resilient cloud-like infrastructure to host defense services and to perform all the heavy tasks related to defense provisioning waiving a large computationally-heavy load from the ToD. Waiving that load in addition to CyPhyCARD utilization of platform independent sensing and effecting capsules for defense provisioning expands the system support to various host configuration and legacy systems.

CyPhyCARD provides the means to automate trustworthy multi-organization information sharing to enable early attack alarm and enhance the defense system global situation-awareness towards more accurate decision making. Enabling such features makes it possible to successfully provision defense services to mission-critical heterogeneously-composed systems like CPS, while maintaining the operation timeliness and stability in presence of persistent attacks.

Throughout CyPhyCARD, we presented three novel contributions addressing a list of serious security challenges facing cyber and CPS domains. These solutions were designed to be self and situation aware and can autonomously and harmonically work together to construct CyPhyCARD.

Figure 7.1 illustrates the main scientific contributions, solved hard-problems, and the major outcomes of realizing CyPhyCARD and its constructing pillars.

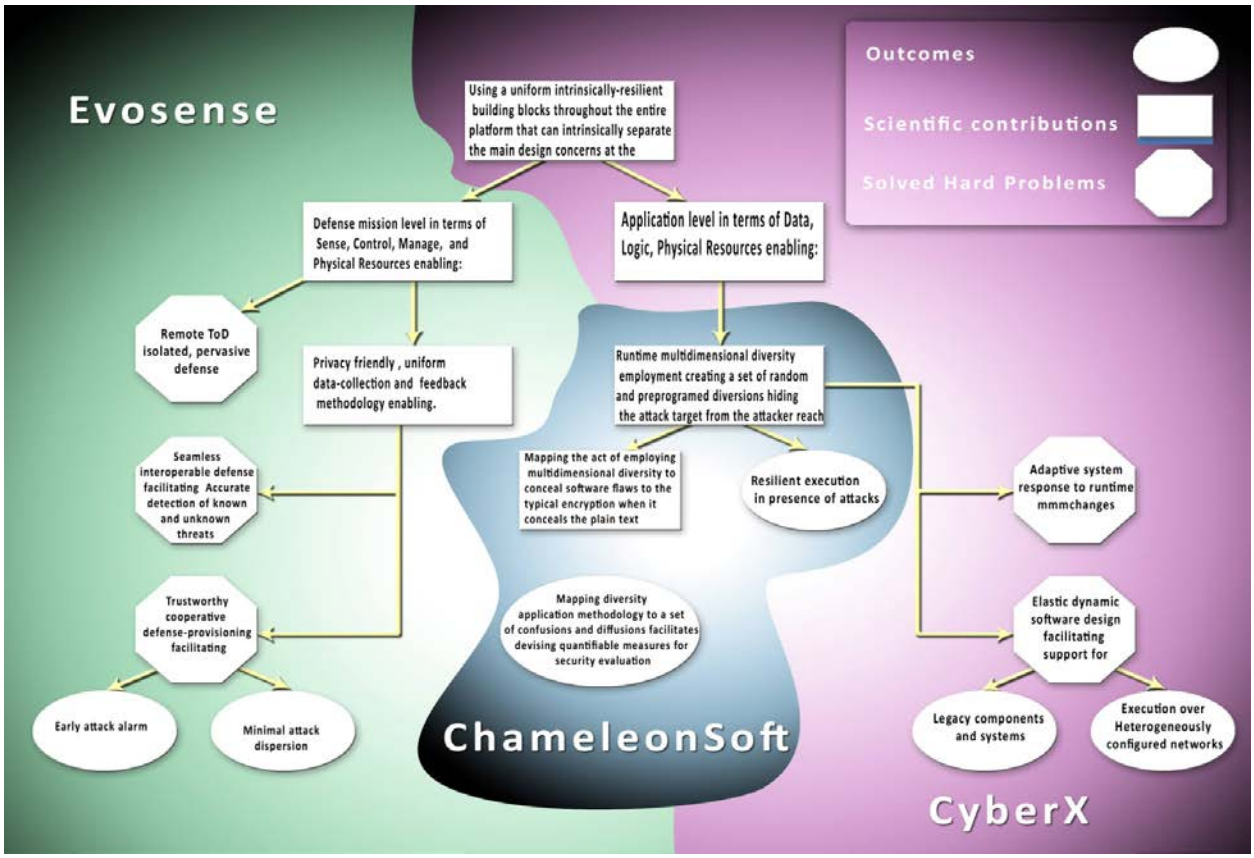


Figure 7.1 CyPhyCARD main contributions

The main contributions of this dissertation are:

- CyberX is a smart management platform that isolate the main design concerns data, logic, and physical resources. Such isolation enabled software applications to be platform-independent, elastic, dynamically adaptable to changes, resilient, and resource efficient.
- ChameleonSoft that employs multidimensional software diversity to, in effect, induce spatiotemporal software behavior encryption. ChameleonSoft utilizes the loosely coupled foundation provided by CyberX to mobiles at runtime the executable behaviorally-encrypted software components across heterogeneously-configured platforms inducing a trace resistant moving target defense.

- EvoSense realizes pervasive monitoring and analysis for heterogeneously composed targets. EvoSense is a biologically-inspired intrinsically-resilient, situation-aware sense and response system to seamlessly effect biological-immune-system-like defense. EvoSense acts as a middle layer between the defense service provider(s) and the Target of Defense creating a uniform defense interface that hides ToD's scale and heterogeneity concerns from the defense-provisioning control and management. EvoSense is elastic where solutions are dispatched through a dynamic set of sensors and effectors to the ToD rather than using pre-deployed M&A components. EvoSense circulates context-driven, online customizable sensing and effecting capsules into the ToD body to pervasively monitor, analyze and control ToD components. The key design principles for EvoSense are:
    - ToD-independent defense service provisioning;
    - Decoupling sensing and effecting tools from the control and management logic towards enabling interoperable and dynamic defense; and
    - Intrinsically supporting trustworthy scalable cooperative defense with shared indicators.

The presented qualitative and quantitative study illustrated the capability of CyPhyCARD and its pillars to effectively and efficiently achieve their design goals. Further, the study illustrated that CyPhyCARD and its pillars can adjust their resource-needs and operational-characteristics to support defense provisioning for large-scale mission critical heterogeneously-composed platforms like CPS.

7.2 **Future Work**

Future work will focus on the following directions:

- The Cell: we realized the simple and fast version of the Cell with partial isolation between the logic and the targeted execution platform. We will devise a lightweight, complex version of the Cell with a fully virtualized environment. Devising such version of the Cell will enable us to construct a full test bed of ChameleonSoft trace-resistant moving-target defense where Cells can migrate between heterogeneous platforms seamlessly with no need to change the current active variant to another variant matching the targeted platform. Enabling such feature will expand the migration landscape increasing the complexity for the attacker to trace its target.
- Variant generation: we will devise techniques for behavior computation to support the design of an automated similar function different behavior variant generation framework capable of generating variants based on specific requirements and behavior deviation distances.
- Intelligence: the system currently uses a generic model for the smart processors controlling all the decisions being taken within the entire platform. Further study will be conducted to determine the best artificial intelligence technique suitable for each location. The system should be able to switch between these techniques at runtime based on the changes of the situation in hand.
- Quality of Service (QoS): we will devise models for the instrumentation and control of various QoS parameters in CyPhyCARD, and develop corresponding control mechanism to adjust the aspects of the working components to maintain the design targeted quality of service levels.

- Sensing and effecting: Formal description of CyPhyCARD ready defense missions for the DSPs to follow. In addition to developing sensor, effector, and control logic extraction framework that can autonomously transform conventional tools to CyPhyCARD ready sensing and effecting APIs.
- Test bed: we will complete the implementation of CyPhyCARD test bed and conduct extensive evaluation of different classes for real life applications under various scales and workload patterns. CyPhyCARD test bed integrates the C# implementation of the Cell, CyberX, ChameleonSoft, EvoSense, and multiple simulation packages that were built using MATLAB. Future versions of the test bed will exploit new virtualization techniques to realize the Cell with integrations of such techniques within CyberX, ChameleonSoft, and EvoSense platforms. Further, the Test bed shall include a framework for building generic CyberX ready digital interface for various physical components. The test bed will be tested on large-scale heterogeneously composed networks of cyber and physical components.

Publications

- Mohamed Azab and Mohamed Eltoweissy, “ChameleonSoft: Software Behavior Encryption for Moving Target Defense,” Springer Journal on Mobile Networks and Applications (MONET), DOI: 10.1007/s11036-012-0392-0 ,2012.

- Mohamed Azab and Mohamed Eltoweissy, “Bio-inspired Evolutionary Sensory System for Cyber-Physical System Defense,” IEEE Technologies for Homeland Security, Nov 2012.

- Mohamed Azab and Mohamed Eltoweissy,”CyberX: A Biologically-inspired Platform for Cyber Trust Management,” 8th International Conference on Collaborative Computing, Oct 2012.

- Mohamed Azab, Reham Hassan and Mohamed Eltoweissy, “ChameleonSoft: A Moving Target Defense System,” 7th International Conference on Collaborative Computing, Oct 2011.

- Mohamed Azab and Mohamed Eltoweissy, “Towards A Cooperative Autonomous Resilient Defense Platform for Cyber-Physical Systems,” 7th Annual Cyber Security and Information Intelligence Research Workshop, Oct 2011.

- Mohamed Azab and Mohamed Eltoweissy,” Defense as a Service Cloud for Cyber-Physical Systems,” 7th International Conference on Collaborative Computing, Oct 2011.

Patents and awards

- **Provisional Patents**

  - ChameleonSoft: Software Behavior Encryption for Moving Target Defense [Application 61731489, EFS ID 14347009] 2012

  - CyberX: Resilient Software Management and Operation Technology [Application 61724987, EFS ID 14198975], 2012

  - Bio-inspired Evolutionary Sensory System for Cyber-Physical System Defense [In preparation]

- **Awards**

  - **“**CyPhyCARD, Smarter Cyber-Physical Security”, nominated as one of the top ten projects in the Sixth Annual National Security Innovation Competition, 2012

Bibliography

[1] A. Cardenas, S. Amin, B. Sinopoli, A. Giani, A. Perrig, and S. S.Sastry, "Challenges for securing cyber physical systems," in *Workshop on Future Directions in Cyber-physical Systems Security*, 2009.

[2] GAO, "Critical infrastructure protection: Challenges and efforts to secure control systems." United States General Accounting Office (GAO), pp. 04–354, 2004.

[3] N. Adam, "Workshop on future directions in cyber-physical systems security." Report on workshop organized by Department of Homeland Security (DHS), 2010.

[4] C. Grim, "Application virtualization," 2012. [Online]. Available: http://www.vmware.com/products/thinapp/overview.html .

[5] G. Lawler, "Distributed architecture for the object oriented methods for interoperability," NAVAL POSTGRADUATE SCHOOL, 2003.

[6] C. Hahn, C. Madrigal-Mora, and K. Fischer, "Interoperability through a platform-independent model for agents," in *3rd International Conference on Interoperability for Enterprise Software and Applications*, 2007.

[7] Zeigler, C. Seo, and B.P., "DEVS namespace for interoperable DEVS/SOA," in *2009 Winter Simulation Conference*, 2009.

[8] C. Carrascosa, A.Terrasa, A.García-Fornes, A.Espinosa, and V.Botti, "Behaviour management in real-time agents," in *Fifth Iberoamerican Workshop on Multi-Agent Systems*, 2004, pp. 1–11.

[9] A. Tolk, S. Diallo, C. Turnitsa, and L. Winters, "Composable M&S Web services for Net-centric Applications," *Journal of Defense Modeling and Simulation*, pp. 27–44, 2006.

[10] P.-O. Östberg and E. Elmroth, "GJMF - A Composable Service-Oriented Grid Job Management Framework," 2010. [Online]. Available: http://www.cs.umu.se/ds.

[11] J. C. LKnight, J. W. Davidson, D. Evans, A. Nguyen-Tuong, and C. Wang, "Genesis: A Framework for Achieving Software Component Diversity." Technical Report AFRL-IF-RS-TR-2007-9, University of Virginia, January, 2007.

[12] S. Forrest, A. Somayaji, and D. Ackley, "Building diverse computer systems," in *6th Workshop on Hot Topics in Operating Systems (HotOS-VI)*,, 1997, pp. 67–72.

[13] T. Jackson, B. Salamat, G. Wagner, C. Wimmer, and M.Franz, "On the Effectiveness of Multi-Variant Program Execution for Vulnerability Detection and Prevention," in *International Workshop on Security Measurements and Metrics (MetriSec 2010)*, 2010.

[14] M. Franz, "E unibus pluram: Massive-Scale Software Diversity as a Defense Mechanism," in *New Security Paradigms Workshop 2010 (NSPW 2010)*, 2010.

[15] NSF, "Cyber Trust program solicitation," 2012. [Online]. Available: http://www.nsf.gov/pubs/2008/nsf08521/nsf08521.htm .

[16] L.Parnas, "On the criteria to be used in decomposing systems into modules," *Communications of the ACM*, vol. 15, no. 12, pp. 1053–1058, 1972.

[17] G.Booch, "Object-oriented design," *ACM SIGAda Ada Letters*, vol. I, no. 3, pp. 64–76, 1982.

[18] C.Szyperski, *Component Software Beyond Object-Oriented Programming*, 2nd ed. Addison Wesley, 2002.

[19] H.Kopetz, "Real-Time Systems Design Principles for Distributed Embedded Applications." Norwell, MA: Kluwer, 1997.

[20] C.Parra, "Towards Dynamic Software Product Lines: Unifying Design and Runtime Adaptation," Université Lille, 2011.

[21] D.Box, *Essential COM*. Addison Wesley, 1998.

[22] B.Burke and R.Monson-Haefel, *Enterprise JavaBeans 3.0. O'Reilly*, 5th ed. Beijing: , 2006.

[23] D.Schmidt, "Tutorial on the Lightweight COBRA Component Model (CCM)," 2012. [Online]. Available: http://www.slideshare.net/jwillemsen/omg-corba-component-model-tutorial.

[24] E.Bruneton, T.Coupaye, M.Leclercq, V.Quéma, and J.Stefani, "The FRACTAL component model and its support in Java: Experiences with Auto-adaptive and Reconfigurable Systems," *Software Practice & Experience*, vol. 36, no. 11–12, pp. 1257–1284, 2006.

[25] F.Plasil, D.Balek, and R.Janecek, "Sofa/dcup: Architecture for component trading and dynamic updating," in *International Conference on Configurable Distributed Systems*, 1998.

[26] R.Filman, T.Elrad, S.Clarke, M.Ak¸, and Sit, *Aspect Oriented Software Development*. Boston: Addison-Wesley, 2005.

[27] G.Kiczales, J.Lamping, A.Mendhekar, C.Maeda, C.V.Lopes, J.Loingtier, and J.Irwin, "Aspect-oriented programming," *ECOOP*, pp. 220–242, 1997.

[28] C.Quintero and et al, "Architectural Aspects of Architectural Aspects," *Springer.EWSA 2005. LNCS*, vol. 3527, no. 247–262, 2005.

[29] N.Josuttis, "SOA in Practice: The Art of Distributed System Design," 2007.

[30] D.Gisolfi, "Web services architect: Part 1, an introduction to dynamic e-business," 2012. [Online]. Available: http://www.ibm.com/developerworks/webservices/library/ws-arc1/.

[31] R. Feldt, "Generating Multiple Diverse Software Versions with Genetic Programming," in *24th EUROMICRO Conference (EUROMICRO '98)*, 1998.

[32] L. Hatton, "N-version design versus one good version," *IEEE Software*, vol. 14, no. 6, pp. 71–76, 1997.

[33] Symantec, "W32.bugbear@mm," 2002. [Online]. Available: http://securityresponse.symantec.com/avcenter/venc/data/w32.bugbear@mm.%html.

[34] R.Ommering, "Building Product Populations with Software Components," University of Groningen, 2004.

[35] M. Chew and D. Song, “Mitigating buffer overflows by operating system randomization.” Carnegie Mellon University, pp. 02–197, 2002.

[36] S.Dai and S.Kuo, “MAPMon: A Host-Based Malware Detection Tool,” *Dependable Computing ,PRDC*, 2007.

[37] R. Pucella and F. Schneider, “Independence from obfuscation: A semantic framework for diversity,” in *IEEE Computer Security Foundations Workshop*, 2006.

[38] C. Cowan, S. Beattie, J. Johansen, and P. Wagle, “PointGuard: Protecting pointers from buffer overflow vulnerabilities,” in *USENIX Security Symposium*, 2003.

[39] PaX, “PaX,” 2001. [Online]. Available: http://pax.grsecurity.net .

[40] R. P. Wilson and M. S. Lam, “Efficient context-sensitive pointer analysis for C programs,” in *ACM SIGPLAN Conference on Programming Language Design and Implementation*, 1995.

[41] J. Xu, Z. Kalbarczyk, and R. K. Iyer, “Transparent runtime randomization for security,” in *Symposium on Reliable and Distributed Systems (SRDS)*, 2003.

[42] F. Cohen, “Operating system protection through program evolution,” *Computers and Security*, 1993.

[43] C. Pu, A. Black, C. Cowan, and J. Walpole, “A specialization toolkit to increase the diversity of operating systems,” in *ICMAS Workshop on Immunity-Based Systems*, 1996.

[44] J. E. Just and M. Cornwell, “Review and analysis of synthetic diversity for breaking monocultures,” in *ACM Workshop on Rapid Malcode (WORM ’04)*, 2004, pp. 23–32.

[45] D. A. Holland, A. T. Lim, and M. I. Seltzer, “An architecture a day keeps the hacker away,” *SIGARCH Computer Architecture News*, vol. 33, no. 1, pp. 34–41, 2005.

[46] S.Sze and W.Tiong, “A Comparison between Heuristic and MetaHeuristic Methods for Solving the Multiple Traveling Salesman Problem,” *World Academy of Science, Engineering and Technology*, 2007.

[47] R.Thomas, A.Householder, A.Manion, L.Pesanet, and G.M.Weaver, “Managing the Threat of Denial-of-Service Attacks,” *CERT Coordination Center*, vol. 10.0, 2001.

[48] Y. Bjornsson and K. Halldorsson, “Improved heuristics for optimal path-finding on game maps,” in *AIIDE*, 2006, pp. 9–14.

[49] X. Chen, J. Andersen, Z. M. Mao, M. Bailey, and J. Nazario, “Towards an understanding of anti-virtualization and anti-debugging behavior in modern malware,” in *International Conference on Dependable Systems and Networks*, 2008.

[50] B. Salamat, T. Jackson, A. Gal, and M. Franz, “Intrusion detection using parallel execution and monitoring of program variants in user-space,” in *Eurosys 2009*, 2009.

[51] B. Salamat, A. Gal, and M. Franz, “Reverse stack execution in a multi-variant execution environment,” in *Workshop on Compiler and Architectural Techniques for Application Reliability and Security (CATARS’08)*, 2008.

[52] B. Salamat, T. J. A. Gal, K. Manivannan, G. Wagner, and M. Franz, “Multi-variant program execution: Using multi-core systems to defuse buffer-overflow vulnerabilities,” in *International Workshop on Multi-Core Computing Systems (MuCoCoS 2008)*, 2008.

[53] R.Lemos, “White House Network Attack Highlights Need for Stronger Defenses,” 2012. [Online]. Available: http://www.eweek.com/security/white-house-network-attack-highlights-need-for-stronger-defenses/.

[54] M. Franz, “E unibus pluram: Massive-Scale Software Diversity as a Defense Mechanism,” in *New Security Paradigms Workshop 2010 (NSPW 2010)*, 2010.

[55] T. Jackson, C. Wimmer, and M. Franz, “Multi-Variant Program Execution for Vulnerability Detection and Analysis,” in *Sixth Annual Cyber Security and Information Intelligence Research Workshop (CSIIRW’10)*, 2010.

[56] B. Salamat, T. Jackson, G. Wagner, C. Wimmer, and M. Franz, “Run-Time Defense against Code Injection Attacks using Replicated Execution,” *IEEE Transactions on Dependable and Secure Computing. IEEE Computer Society*, 2011.

[57] F. Cohen, “Computer Viruses,” University of Southern California, 1985.

[58] E. H. Spafford, “Computer viruses as artificial life,” *Journal of Artificial Life*, 1994.

[59] A. Avizienis and L. Chen, “On the implementation of n-version programming for software fault tolerance during execution,” *IEEE COMPSAC 77*, pp. 149–155, 1977.

[60] J.Wood and K.Jackson, “How Cephalopods Change Color,” 2012. [Online]. Available: http://www.thecephalopodpage.org/cephschool/.

[61] D. E. Eckhardt, A. K. Caglayan, J. C. Knight, L. D. Lee, D. F. McAllister, M. A. Vouk, and J. J. P. Kelly, “An experimental evaluation of software redundancy as a strategy for improving reliability,” *IEEE Transactions on Software Engineering*, 1991.

[62] C.Pfleeger and S.Pfleeger, *Security in Computing*, 3rd ed. Prentice Hall, 2003.

[63] Y.Cai, “Mobile Agent Based Network Defense System in Enterprise Network,” *International Journal of Handheld Computing Research*, vol. 2, pp. 41–54, 2011.

[64] E. Barrantes, D. Ackley, T. Palmer, D. Stefanovic, and D. Zovi, “Randomized instruction set emulation to disrupt binary code injection attacks,” in *ACM Conference on Computer and Communications Security*, 2003, pp. 281–289.

[65] G. Kc, A. Keromytis, and V. Prevelakis, “Countering code-injection attacks with instruction-set randomization,” in *ACM Conference on Computer and Communications Security*, 2003.

[66] S.Musman, A.Temin, M.Tanner, D.Fox, and B.Pridemore, “Evaluating the impact of cyber attacks on missions,” in *5th International Conference on Information Warfare and Security*, 2010.

[67] B. Cox, D. Evans, A. Filipi, J. Rowanhill, W. Hu, J. Davidson, J. Knight, A. Nguyen-Tuong, and J. Hiser, “N-variant systems: A secretless framework for security through diversity,” in *the 15th USENIX Security Symposium*, 2006.

[68] M. Polychronakis, K. G. Anagnostakis, and E. P. Markatos, “Real-world polymorphic attack detection using network-level emulation,” in *CSIIRW ’08: Proceedings of the 4th annual workshop on Cyber security and information intelligence research*, 2008.

[69] S. McGrath, D. C. N, and K. Whitebread, “Intelligent Mobile Agents in Military Command and Control.” Advanced Technology Laboratories, 2000.

[70] C. Cowan, C. Pu, D. Maier, J. Walpole, P. Bakke, S. Beattie, A. Grier, P. Wagle, Q. Zhang, and H. Hinton, "Stack Guard: Automatic adaptive detection and prevention of buffer-overflow attacks," in *the 7th USENIX Security Symposium*, 1998.

[71] W.Podgórski, "Artificial Intelligence Methods in Virus Detection & Recognition - Introduction to heuristic scanning," 2012. [Online]. Available: http://podgorski.wordpress.com.

[72] S.Prayurachatuporn and L.Benedicenti, "Increasing the reliability of control systems with agent technology," *ACM SIGAPP Applied Computing*, 2001.

[73] L. Chen and A. Avizienis, "N-Version Programming: A Fault Tolerance Approach to Reliability of Software Operation," in *8th International Symposium on Fault-Tolerant Computing*, 1978.

[74] M.Joseph, "Architectural Issues in Fault- Tolerant, Secure Computing Systems," UCLA Department of Computer Science, 1988.

[75] I.Santos, Y.Penya, J.Devesa, and P.Bringas, "N-Grams-based file signatures for malware detection," in *the 11 the International Conference on Enterprise Information Systems (ICEIS)*, 2009.

[76] R.Linger, T.Daly, and M.Pleszkoch, "Function Extraction (FX) Research for Computation of Software Behavior: 2010 Development and Application of Semantic Reduction Theorems for Behavior Analysis." the Air Force Office of Scientific Research Mathematics and Information Science Directorate, 2011.

[77] R.Bartholomew, L. Burns, T. Daly, R. Linger, and S.Prowell, "Function Extraction: Automated Behavior Computation for Aerospace Software Verification and Certification," in *2007 AIAA Aerospace Conference*, 2007, pp. 2145–2153.

[78] R.Spruijt, "Application Virtualization Smack down: Head-to-head analysis of Citrix, Endeavors, Install Free, Microsoft, Spoon, Symantec and VMware," 2012. [Online]. Available: www.brianmadden.com/blogs/rubenspruijt/archive/2010/09/22/application-virtualization-smackdown-head-to-head-analysis-of-endeavors-citrix-installfree-microsoft-spoon-symantec-and-vmware.aspx .

[79] M.Schmitt, "Wired Warfare: Computer Network Attack and Jus in Bello." International Review of the Red Cross, 2002.

[80] A. Nguyen-Tuong, A.Wang, J. Hiser, J.Knight, and J. Davidson, "On the effectiveness of the metamorphic shield," in *The Fourth European Conference on Software Architecture ECSA '10*, 2010, pp. 170–174.

[81] A.Peslyak, "Return-to-libc Attack." Bugtraq Mailing List, 1997.

[82] N. Hardy, "The Confused Deputy (or why capabilities might have been invented)," *ACM SIGOPS Operating Systems Review*, vol. 22, no. 4, 1988.

[83] J. C. Knight and N. G. Leveson, "An experimental evaluation of the assumption of independence in multiversion programming," *IEEE Transactions on Software Engineering*, vol. 12, no. 1, pp. 96–109, 1986.

[84] N. Jorstad and T. S. Landgrave, "Cryptographic algorithm metrics," in *20th National Information Systems Security Conference*, 1997.

[85] P.Pal, R.Schantz, K.Rohloff, and J.Loyall, "Cyber physical Systems Security Challenges and Research Ideas," in *Workshop on Future Directions in Cyber-physical Systems Security*, 2009.

[86] C.Neuman, "Challenges in Security for Cyber-Physical Systems," in *DHS Workshop on Future Directions in Cyber-Physical Systems Security*, 2009.

[87] J. Haack., G. Fink, E. Fulp, and W. Maiden, "Cooperative Infrastructure Defense," in *Workshop on Visualization for Computer Security (VizSec)*, 2008.

[88] W. M. Maiden, "DualTrust, A Trust Management Model for Swarm-Based Autonomic Computing Systems," Washington State University, 2010.

[89] W. M. Maiden, I. Dionysiou, D. A. Frincke, G. A. Fink, and D. E. Bakken, "DualTrust: A Distributed Trust Model for Swarm-Based Autonomic Computing Systems," *Data Privacy Management and Autonomous Spontaneous Security*, 2010.

[90] Y.Lee, "A Pre-Kernel Agent Platform for security assurance," in *IEEE Symposium on Intelligent Agent (IA)*, 2011.

[91] A. Abraham, R. Jain, J. Thomas, and S. Y. Han, "D-SCIDS: distributed soft computing intrusion detection system," *Journal of Network and Computer Applications*, vol. 30, no. 1, pp. 81–98, 2007.

[92] S.Wu and W.Banzhaf, "The use of computational intelligence in intrusion detection systems: A review," *Applied Soft Computing*, vol. 10, no. 1, pp. 1–35, 2010.

[93] S.Mukherjee, "FPGA based Network Security Architecture for High Speed Networks," MTech, 2001.

[94] M. Otey, S. Parthasarathy, A. Ghoting, G. Li, S. Narravula, and D. Panda, "Towards nic based intrusion detection," in *the ninth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2003, pp. 723–728.

[95] K.Gurdip, "Intrusion detection system using honeypots and swarm intelligence," in *the International Conference on Advances in Computing and Artificial Intelligence - ACAI '11*, 2011.

[96] C.Te-Shun, F.Sharon, Z.Wei, F.Jeffrey, and D.Asad, "Intrusion aware system-on-a-chip design with uncertainty classification," in *The 2008 International Conference on Embedded Software and Systems-ICESS*, 2008.

[97] M.Azab, R.Hassan, and M.Eltoweissy, "ChameleonSoft: A Moving Target Defense System," in *7th International Conference on Collaborative Computing: Networking, Applications and Worksharing, (CollaborateCom'11)*, 2011.

[98] D. Spinellis, "Reliable identification of bounded-length viruses is NP-complete," *IEEE Transactions on Information Theory*, vol. 49, no. 1, pp. 280–284, 2003.

[99] M. Suresh, R. Stoleru, R. Denton, E. Zechman, and B. Shihada, "Towards optimal event detection and localization in acyclic flow networks," in *International Conference on Distributed Computing and Networking*, 2012.

[100] Y. Liu and K. Han, "Behavior-based Attack Detection and Reporting in Wireless Sensor Networks," in *Third International Symposium on Electronic Commerce and Security (ISECS)*, 2010, pp. 209–212.

[101] J. D.Murray, *Mathematical Biology*, 2nd ed. Springer Verlag, 1993.

[102] J. Piqueira, B.Navarro, and L.Monteiro, "Epidemiological Models Applied to Viruses in Computer Networks," *Journal of Computer Science*, vol. 1, no. 1, pp. 31–34, 2005.

[103] A. V. R.Pastor-Satorras, "Epidemic spreading in scale-free networks," *Physical Review Letters*, vol. 86, no. 14, pp. 3200–3203, 2001.

[104] H.Ebel, L.-I.Mielsch, and S.Bornholdt, “Scale free topology of email networks,” *Phys*, vol. 66, 2002.

[105] M.Williamson, “Throttling viruses: Restricting propagation to defeat malicious mobile code,” in *ACSAC Security Conference*, 2002, pp. 61–68.

[106] S. Schechter, J. Jung, and A. Berger, “Fast Detection of Scanning Worm Infections,” in *Seventh International Symposium on Recent Advances in Intrusion Detection (RAID)*, 2004.

[107] C.Zou, W.Gong, D.Towsley, and L.Gao, “The monitoring and early detection of internet worms,” *IEEE/ACM Transactions on Networking (TON)*, vol. 13, no. 5, pp. 961–974, 2005.