Securing Cognitive Radios with a Policy Enforcer and Secure Inter-component Transport Mechanisms

Jatin S. Thakkar

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

Master of Science
in
Electrical Engineering

Jung-Min "Jerry" Park, Chair
R. Michael Buehrer
Sandeep Shukla

September 03, 2010
Blacksburg, Virginia

Keywords: Policy Enforcer, Cognitive Radios, CORBA

Securing Cognitive Radios with a Policy Enforcer and Secure Inter-component Transport Mechanisms

Jatin S. Thakkar

ABSTRACT

Current wireless communications are confronted with two significant problems with regard to spectrum use — spectrum scarcity and deployment difficulties. It is widely believed that Software Defined Radios (SDRs) and Cognitive Radios (CRs) are the key enabling technologies to address these problems.

The reconfigurability of SDRs combined with the decoupling of policies and the platform in policy-based radios poses a new technical problem — viz, enforcing policy conformance. Each DARPA XG radio is equipped with a set of policy conformance components (PCCs) which are responsible for ensuring that the radio is policy-conformant and does not cause harmful interference. The Policy Reasoner (PR) is the inference component of the PCCs whereas the Policy Enforcer (PE) performs enforcement.

DARPA's XG program prescribes the Software Communications Architecture (SCA) as the model for SDR/CR architectures. Distributed processing is a fundamental aspect of the SCA, and it uses the Common Object Resource Broker Architecture (CORBA). It is reasonable to assume that some of the SDRs will be implemented as distributed systems, irrelevant of whether they are SCA compliant devices. It is thus obvious that middleware has to be secured for complete security.

This thesis enumerates the requirements of an "ideal" PE. We have described the design and implementation of two possible implementations, which can fulfill some of these requirements. The PE can function similar to a firewall, and be at the very boundary of software and hardware components. The PE can also be implemented as a "man-in-the-middle" between the System Strategy Reasoner and the transmission hardware. We further describe a novel method of providing cache coherency for a cache-based PE.

We also perform an in-depth analysis of the security requirements in a distributed implementation of a policy-based radio. To this end, we describe the design and implementation of such a system using CORBA middleware. We identify potential vulnerabilities due to the use of CORBA, and describe countermeasures for them. We compare the performance of transport and security mechanisms of two commercial, off-the-shelf (COTS) Object Request Brokers. We show that the magnitude of performance degradation can be reduced by the use of a cleverly selected combination of transport and security mechanisms.

This thesis is dedicated to my beloved mother, Late Mrs. Jyoti Thakkar

— *your memories always give me the strength and support I need,*

and to

my father and Meeta,

for their constant and undying love, support and encouragement.

Acknowledgments

I cannot find words to express gratitude to my parents and my family for their support at all stages in my life, particularly graduate studies. I could not have reached this stage without the love and support of my parents — Mr. Suresh Thakkar and Mrs. Jyoti Thakkar, who always had faith in all my endeavours. I would like to thank my sister Mrs. Meeta Ghotgalkar and my brother in law Mr. Nilesh Ghotgalkar for their support, particularly in the last few months.

I would like to thank my advisor, Dr. Jung-Min "Jerry" Park for his guidance and support from the first day I met him. Your constructive criticism and invaluable suggestions have not only helped my research, but also made me into a better person. I especially thank you for your patience and faith in me over some difficult times. I am extremely privileged to have been a student under your guidance.

I would like to thank my committee members Dr. R. Michael Buehrer and Dr. Sandeep Shukla for agreeing to be on my committee. I would like to especially thank them for their patience and support during my thesis writing and defense exam scheduling.

I had the amazing opportunity to work with some wonderful people at the ARIAS lab. I would like to thank Amol Deshpande, Swati Kanaujia, Daniel Ali, Alexandru Turcu, Kaigui Bian, Behnam Bahrak, Hao Wu, Bo Gao and Maxwell Whitaker for their help in various aspects of my research. I would like to thank Alexandru and Daniel for their immense help during implementation and coding — you guys rock! I really appreciate the insightful

solutions provided by Swati for most of my coding problems, and Amol's help in virtually all aspects of my research. I loved the time spent with you guys, in the lab and at the lab dinners.

I would like to thank all my friends who have moulded me into a better person, especially Priyam, Priyanka, Prajwal, Shreyas, Richa, Pranav, Ankit, Prashant, Abhijit, Sohan, Jai, Kanchan, Nikhil and Karl. Special thanks to Sakshi for her help with LaTeX formatting, and to Navaneeta for her company during thesis writing. I would like to thank everyone who has helped me enjoy some special moments, provided support during tough times and been there for me in my hour of need.

Finally, I thank God and my mother for their countless blessings.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Motivation

Software-defined Radios (SDRs) are emerging as an essential element of future wireless communications. SDR defines a collection of hardware and software technologies where some or all of a radio's physical layer processing are implemented through modifiable software or firmware operating on programmable processing technologies [2]. SDRs are much more capable than current handsets or "*software-controlled radios*" [3], which usually employ embedded computers only to adjust amplifier gains or select an appropriate radio frequency (RF) front end for supporting multiple frequency bands. An ideal SDR performs filtering and conversion of signals from RF to lower frequencies digitally, leaving only a low noise RF amplifier and antennas as hardware components in the entire radio architecture. This provides for unprecedented flexibility, since a single SDR can potentially be able to transmit and receive signals over different frequencies, standards and modulation techniques. Cognitive Radios (CRs) are SDRs that have intelligence to adapt to their environment and can reconfigure themselves for optimal performance (or some other predefined objective).

Current wireless communications are confronted with two significant problems with regard

to spectrum use — spectrum scarcity and deployment difficulties. The current method of fixed spectrum block allocations leads to the spectrum scarcity problem, while difficulties in configuring wireless systems to coordinate with complex frequency and operating limits at different places lead to long deployment delays. It is widely believed that SDRs and CRs are the key enabling technologies that can address these problems. DARPA's NeXt Generation (XG) communications program [4] aims to develop the enabling technologies and system concepts for policy-based CRs [5] [6] that are capable of Dynamic Spectrum Access (DSA). A policy is a guiding principle or procedure specifying spectrum usage. In other words, in the context of policy-based CRs, policies are required to govern the dynamic spectrum usage of CRs. XG radios utilize *opportunistic spectrum access* [7] to confront the problems of spectrum scarcity. The fundamental change from legacy systems is that spectrum management is now performed by each radio, by assessing the actual situation at each instant in time, rather than predefined behaviors [7]. This approach requires that in addition to *spectrum agility*, the radios have *policy agility* — wherein the policies controlling radio behavior can be dynamically changed. The XG program provides policy agility by the decoupling of policies from behaviors and behaviors from protocols in policy-based radios.

The reconfigurability of SDRs combined with the decoupling of policies and the platform in policy-based radios poses a new technical problem — viz, enforcing policy conformance. Enforcing spectrum access policies in legacy radios is relatively simple since the policies are an inseparable part of the radio's firmware and platform. Making controlled changes to these policies would require an adversary to have very specialized expertise in firmware and hardware. Unfortunately, the same cannot be said of SDRs/CRs. The reconfigurability of SDRs introduces the possibility of an adversary making unauthorized changes to the signal processing software with the intent to change the radio's behavior — e.g., transmitting in a spectrum band not approved by the spectrum regulators or transmitting over the authorized power limit. Such scenarios pose serious security concerns which can hamper the wide spread adoption of SDR and CR technology for next generation wireless communications systems. These concerns exist for simple SDRs as well as advanced policy-based CRs such as XG

radios.

Each XG radio is equipped with a set of policy conformance components (PCCs) [5]. These are responsible for ensuring that the radio's behavior conforms to the currently active policies and does not cause harmful interference. The Policy Reasoner (PR) or the policy conformance reasoner (PCR) is the inference component of the PCCs whereas the Policy Enforcer (PE) performs enforcement. A system strategy reasoner (SSR) determines the system's behavior under regulatory and system policies for opportunistic spectrum access. The PE thus takes a central role in ensuring that radio behavior is compliant with policies.

The XG program utilizes the Software Communications Architecture (SCA) [8] as the model for SDR architectures. Distributed processing is a fundamental aspect of the SCA, and it uses the Common Object Resource Broker Architecture (CORBA) [9] to achieve its portability and platform independence goals. The SCA prescribes a subset of CORBA called *minimum CORBA* [10]. CORBA is a middleware that provides a layer of abstraction between the application and platform-specific elements, such as operating system, network and transport layers. In the context of SCA-based radios, CORBA is primarily used for component location, inter-component communication, and logging [11]. Distributed systems have the advantage that components can be implemented on platforms ideal for them, and still work together with all other components. A distributed SDR system is ideal since SDRs can consist of multiple hardware components such as General Purpose Processors (GPPs), DSPs and FPGAs, each performing a specific functionality required from the SDR. It is reasonable to assume that some of the SDRs will be implemented as distributed systems, irrelevant of whether they are SCA compliant devices. It is thus obvious that securing the middleware is an important requirement for securing the radio, and for ensuring policy-compliant behavior.

The PE acts as the last line of defense to prevent radios from exhibiting rogue transmission behavior in policy-based radios. The core of the *policy enforcing* problem is to ensure that conditions for spectrum access specified by the regulators are met by the radio at all times. A PE thus has to ensure that the transmission hardware does not allow illegal

transmissions at all times. A PE has to perform this functionality for any policy-based radio, including distributed systems. For distributed systems, the PE has to be assisted by secure transport mechanisms between the components of the SDR for enforcing policies. Previous research has focused on the performance and viability of CORBA within a real-time distributed system such as an SDR [11, 12]. Research has also focused on securing CORBA implementations [13] [14] and SDRs [15]. However, there has not been a focus on the performance effects of providing security to a distributed SDR system such as a distributed policy-based CR. In light of the importance of enforcing policies, it is evident that more research is needed in terms of defining the requirements of a PE, as well as its design and development in simple and distributed systems. Research is also required to understand the trade-off relationship between performance and security in distributed policy-based CR systems.

1.2 Contribution

We have determined the requirements of an "ideal" PE. We have proposed two possible implementations that can fulfill some of these requirements for current policy-based radio architectures. We determined that the placement of the PE in the policy-based radio architecture influences its core functions and implementation. There are two possible locations where a PE can be located in the policy-based radio architecture. The PE can function similar to a firewall, and be at the very boundary of software and hardware components — employing a "monitoring" approach to policy enforcement. We call this the *RF Firewall* approach. The PE can also be implemented as a "*man-in-the-middle*" between the System Strategy Reasoner and the transmission modem — intercepting messages and enforcing policies "pro-actively". Apart from the proposed PE implementations; we also perform an analysis of the requirements of a PE and security in a distributed system implementation. To this end, we describe the design and implementation of a distributed policy-based cognitive radio system using CORBA middleware.

To evince the necessity of the PE, we have implemented an attack against a SDR as a proof of concept. Specifically, the attack was launched against a SDR that employs a laptop for running GNU Radio [16] software and a USRP [17] board for processing RF signals. This attack makes controlled changes to a SDR's transmission frequency without modifying the radio's signal processing software and circumvents any tamper resistance scheme that may be in place to protect it. The attack is thus aimed at the signal generating component (generally the SSR) in a policy-based radio. The attack exploits hardware features common to most modern GPPs, such as floating point registers (FPRs) and debug registers (also called Hardware Breakpoints (HWBPs)). Existing integrity checking-based tamper resistance schemes would be unable to detect the attack because the attack does not make any changes to the signal processing software itself. Instead, the attack modifies the contents of an FPR.

For our first proposed implementation of a PE, we describe the architecture of a PE analogous to network firewalls, called the *RF Firewall*. As the name implies, the proposed PE applies the principles of network firewalls to enforce policy conformance in CRs. Analogous to network firewalls that filter packets that do not conform to network security policies, the *RF Firewall* blocks transmissions that are non-conforming to the active set of spectrum access policies. The *RF Firewall* blocks non-conforming transmissions by monitoring the passage of the appropriate control signals from the signal processing software to the RF hardware. We demonstrate that the *RF Firewall* is adept at countering attacks such as the one we have implemented.

For our second implementation, the PE acts as a "man-in-the-middle" between the SSR and the transmission modem. Any communication from the SSR to the modem has to go through the PE, which ensures that transmission commands non-conforming to active policies never reach the modem. This prevents the radio from exhibiting rogue transmission behavior, even if an adversary manages to manipulate the SSR or if the SSR is not functioning properly. Our implementation of the *MITM* PE uses a cache mechanism to improve performance. The cache-based approach allows the PE to alleviate the processing load of the PR by making repetitive policy reasoning unnecessary. Our novel approach of attaching the *meta-policy*

obtained from the PR to the cached values provides cache coherency for updates in policy without the need for periodic or unnecessary flushing of the cache.

To experimentally evaluate performance penalties of incorporating security into the middleware, we have implemented a prototype distributed policy-based CR. The system consists of a simple SSR, our cache-based PE, a PR and a transmission modem implemented as a distributed system using CORBA. This system utilizes *BRESAP* (Binary decision diagram-based REasoner for Spectrum Access Policies) [18], which is a peer-reviewed implementation of a PR. The implemented system employs GNU Radio software to create the waveforms and the USRP board to generate the RF signals as part of the transmission modem component. Further, we analyze the use of different transport and security mechanisms used in commercial, off-the-shelf (COTS) CORBA Object Request Brokers (ORBs).

The major research contributions of this thesis can be summarized as follows:

1. We have implemented a "proof of concept" attack against a SDR that demonstrates that it is possible for an adversary to make controlled changes to a SDR's transmission frequency, even if the radio's signal processing software is protected with a tamper resistance scheme. The attack bypasses any *checksumming-based* tamper resistance scheme that may be protecting the signal processing software. This attack underscores the importance of a PE.

2. We enumerate the requirements of an "ideal" PE. We describe the design of a PE, called the *RF Firewall* — that enforces policy conformance of the cognitive radio's transmissions. The working of the *RF Firewall* is analogous to traditional network firewalls. To validate the principles of the *RF Firewall*, we have implemented a proof of concept implementation of it on the GNU Radio-USRP platform.

3. We design and implement a cache-based PE which acts as a "man-in-the-middle" between the SSR and the transmission modem to pro-actively enforce policies. We propose a novel method of maintaining cache coherency in cache-based PEs that utilizes

meta-policies obtained from the PR. We analyze the benefits of using a cache-based approach vs. a non-cache approach and provide results for the same.

4. We identify potential vulnerabilities due to the use of CORBA in a distributed CR system and describe countermeasures for the same. We compare the performance of transport and security mechanisms of two COTS ORBs in providing these countermeasures. To achieve this, we have implemented an actual policy-based CR system utilizing *BRESAP*, GNU Radio, USRP and our cache-based PE as components in a distributed system utilizing CORBA. We analyze the performance penalty of providing security in a distributed CR system using an actual implementation.

1.3 Thesis Organization

Chapter 2 provides a brief overview of current research on Dynamic Spectrum Access systems. It is followed by introduction to current policy-based architectures, the SCA and distributed systems in SDRs. A brief description of CORBA and other middleware as well as their security issues is also provided. The chapter concludes with a detailed analysis of current research and related work in the areas described in the chapter.

Chapter 3 describes a "proof of concept" attack on an SDR system. The system architecture used for the attack is described. The assumptions for the attack are enumerated, and the attack mechanism is explained in detail using flowcharts.

Chapter 4 provides an in-depth analysis of the role of a Policy Enforcer in a policy-based CR. The requirements of an ideal PE are described. It also explains two possible approaches for implementing a PE. The design and implementation of the two approaches is then described and they are compared.

Chapter 5 describes the implementation of a prototype distributed policy-based radio CR. It highlights potential vulnerabilities in a distributed policy-based CR system using CORBA.

Possible countermeasures for these vulnerabilities and their implementations are also described.

Chapter 6 provides results of our experimental evaluation for the attack, *RF Firewall* and cache-based PE implementations. An analysis of the trade-offs between performance and security for transport mechanisms used in CORBA middleware is also provided.

Chapter 7 concludes the research work, and provides suggestions for future work in securing distributed CR systems.

Chapter 2

Technical Background

2.1 Dynamic Spectrum Access

Dynamic Spectrum Access (DSA) is one of the most promising technologies available to revolutionize current and future spectrum dependent communication services. DSA systems identify unused spectrum and manage their users' operation within the identified spectrum. DSA systems also ensure that no "harmful" interference is caused to other users either by sensing the environment (as described in [19]) or by using pre-existing knowledge and real-time databases (as proposed for *TV Whitespace* systems), or a combination of both [20]. DSA thus increases spectrum efficiency by enabling spectrum allocation in a dynamic manner to utilize vacant spectrum over space and time. DSA systems achieve their functionality using DSA-enabled devices, which are defined as smart devices that can sense spectrum opportunities and correspondingly adapt their transmission parameters. Cognitive radios (CRs) [21] and software defined radios (SDRs) are ideal choices for DSA systems.

DSA technologies have increased importance today due to the huge strain on spectrum resources caused by current licensing based spectrum allocation. Current spectrum allocation is based on static licenses in which large spectrum spaces are licenses to entities for a price.

However, these large spectrum blocks are monopolized by the high spectrum consuming technologies utilized in them [22]. Further, studies show that some capacity in a cellular carriers' allocated spectrum remains idle for more than 98% of the time when the number of calls underway require less than full capacity [23]. Meanwhile, new entrants who may only want small amounts of bandwidth for sporadic periods of time must wait for large swaths to become available and then pay billions for an allocation they cannot completely fill at all times and all places [24]. This has led to an apparent scarcity of spectrum. Studies today indicate that smart-phones will outnumber ordinary voice phones by 2015 [25]. Also, the number of end-user devices such as cameras, video recorders, eBook readers, etc. that incorporate some form of wireless technology is increasing along with a new class of sensors, monitors and surveillance equipment that can connect to the internet. The combined effect of these trends is that mobile data usage and broadband traffic are expected to increase exponentially in the next few years [25]. These estimates call for an increase in the capacity of current wireless networks and the need for evolution of wireless networks to satisfy growing demands.

Finding new spectrum, auctioning it and actually using it takes 6-13 years in the current licensing long-term model [26] and it is increasingly hard to find new spectrum. Thus, traditional methods of increasing capacity for wireless networks may not be effective for the impending needs of mobile and broadband data traffic. A new model of spectrum usage is thus required, and DSA systems aim to fulfill this need.

2.2 Policy-based Radios

One of the primary requirements from a radio for *opportunistic spectrum access* is enforcing behaviors consistent with applicable policies while using identified opportunities. The policy-based approach to radio operation involves decoupling of the policy definition, loading and enforcement from device specific implementations and optimizations as it is done in

traditional radios. The XG approach for policy-based radios requires that the radio is *policy controlled*.

2.2.1 Advantages of the Policy-based Approach

The decoupling of policies from behaviors, and behaviors from protocols as described in the XG vision [7] provides for distinct advantages in policy-based radios. Some advantages of a policy-based architecture are (from [27]):

- Radio behavior can quickly adapt to changing situations since policies can be dynamically loaded.
- Since policies are platform-independent, they can be loaded on different types of radios.
- Certification efforts are reduced since only the radio components that process policies need to be accredited to ensure policy conformant behavior.
- Radio devices and policies can evolve independently over time, thus breaking the cyclic dependency that exists between the regulatory bodies and wireless technologies. Today, regulatory bodies wait for technology to mature and technology needs to wait to see how the regulatory bodies specify usage for the policies.
- The policy-based approach is extensible with respect to the kinds of policies that can be expressed.
- An unprecedented amount of freedom and control of spectrum is possible as stakeholders can formulate spectrum policies (allowed by regulations) to best fit their needs.

2.2.2 Policy-based Radio Architectures

Under the auspices of the DARPA-XG program, several architectures for policy-based radios have been suggested. Figure 2.1 and Figure 2.2 show two popular architectures for policy-

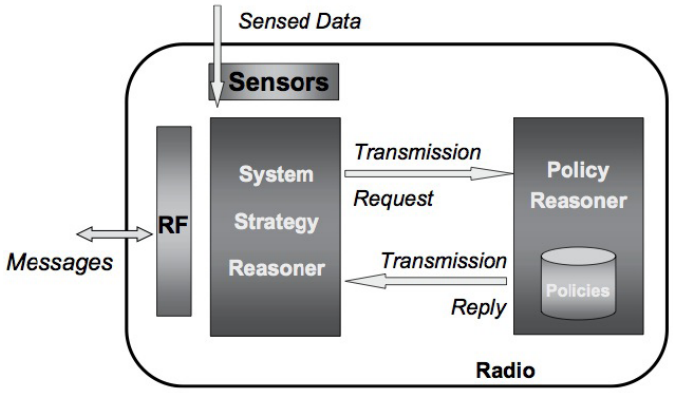

Figure 2.1: A policy-based Radio architecture with PR as the only PCC (from [1])

based radios. Authors in [6] describe an architecture that utilizes the PR for reasoning as well as enforcement. In such a case, the PR includes a module similar to the PE which performs the task of ensuring policy conformant behavior by monitoring changes requested by the SSR to the transmission modem. This architecture is similar to the one shown in Figure 2.1. Authors in [5] have split the policy reasoner described in [6] into a PR and a PE, similar to the architecture in Figure 2.2. As per [6], parts of the system that lie within the accreditation boundary are both necessary and sufficient to ensure policy conformance. As described in the XG vision [7], system and protocol innovations including the SSR are outside the accreditation boundary. The role of individual components for the architecture shown in Figure 2.2 is described in detail below.

- **System Strategy Reasoner**: The SSR determines the system's strategy for opportunistic spectrum sharing under regulatory and system policy constraints; this reasoner is aware of system specific optimizations and trade-offs and has control over the radio platform [6]. In keeping with the XG vision, the SSR is independent of the PCCs and may or may not be accredited. This allows for implementing latest cognitive technologies in the SSR without waiting for certification, with the assurance that the PCCs will ensure policy compliant behavior.

- **Policy Reasoner**: The PR works as the local policy decision unit for every policy-

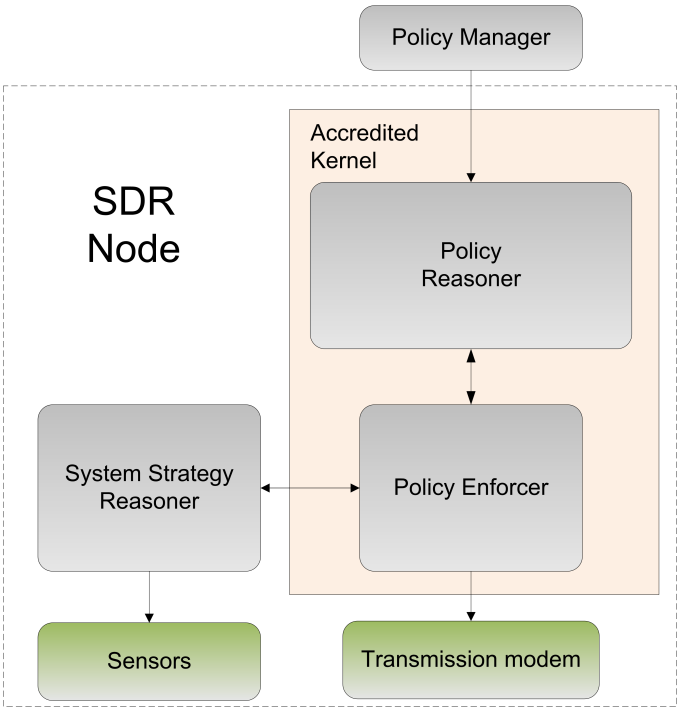

Figure 2.2: A policy-based architecture having separate PR and PE

based radio node [5]. The PR obtains the current policies and updates from a Policy Manager or over the network. The PR receives *transmission requests* from the SSR (through the PE or directly). When it receives a transmission request, the PR evaluates it against the current list of active policies, which it stores in the local policy database. A transmission request is permitted only if requirements of all currently active policies are fully met [28]. Once a decision is reached to allow or disallow transmission, the PR notifies the PE and the SSR.

- **Policy Enforcer**: The PE monitors transmission requests sent to the transmission modem by the SSR to ensure that transmission commands sent to the modem conform to the set of currently active policies. The PE has complete control over the communication hardware and can identify invalid requests from the SSR and block transmissions if needed. The PE continuously evaluates the current state and channels in use by the SSR and ensures transmissions fully satisfy policy requirements specified

by the PR.

- **Sensors**: A sensor provides situational information to the radio about the spectral environment at a given location and time. This information is key for the radio being situationally aware of spectrum opportunities enabled by policy [28].

- **Transmission modem**: The Radio hardware provides the basic hardware and primitives of a host radio system that enables opportunistic use of spectrum [28]. This is referred to as the transmission modem.

- **Policy Manager**: The policy manager (PM) stores policies authored by the regulators and maintains the database of policies. The PM acts as the gateway to on-node policy components for the regulators. The PM is responsible for activating the right set of policies in the PR for changes in radio functionality requested by the regulator, or on policy updates.

2.2.3 Operation of a Policy-based Radio

For the architecture in Figure 2.2, the interactions between components for a valid transmission, invalid transmission and incomplete transmission are described below. These interactions have been defined based on their descriptions in [5], [6] and [18].

- *Valid Transmission*: This is the case in which the optimum parameters decided by the SSR to transmit adhere to the currently active policies, and transmission can be allowed.

    1. The PM loads the currently active policy(s) specified for the device by the regulators into the PR.
    2. The SSR senses conditions enabling transmission at some calculated parameters, and decides to transmit by asking the transmission hardware to transmit.

3. The PE monitors these calculated parameters sent from the SSR to the modem, and verifies if they follow currently active policies. It forwards the parameters to the PR if it cannot verify the parameters itself (based on a cache or some other mechanism).
4. The PR gets the parameters from the PE, and evaluates it with reference to currently active policies to verify if the request satisfies all active policies.
5. The PE gets a positive reply from the PR, it allows the transmission hardware to transmit and transmission takes place. It also informs the SSR that the transmission is taking place.

- *Invalid Transmission*: In this case, the optimum parameters decided by the SSR to transmit do not adhere to the currently active policies. This could be due to an error in the SSR calculations, or a possible malicious modification of the SSR as assumed in the attacks described in Section 2.6.4. The transmission cannot be allowed in this case.

  1. Steps 1 to 4 are same as the case for valid transmission.
  2. The PE gets a negative reply from the PR, and it does not allow the modem to transmit, so no transmission takes place. The SSR is informed that the parameters do not adhere to currently active policies.

- *Incomplete transmission*: There could be a situation in which the SSR cannot determine some transmission parameters and needs to know the allowed values from the PR based on currently active policies. Assuming the currently active policy has been loaded by the PM into the PR already, the events for this case are as follows.

  1. The SSR sends missing values in the parameter list to the transmission hardware for transmission.
  2. The PE determines that the required parameters are incomplete, makes a transmission request and sends it to the PR.

3. The PR recognizes the incomplete transmission request and applies algorithms to obtain optimum values (if any) for the missing parameter values, and returns these values to the PE.
4. The PE forwards optimum parameters to the SSR which can then attempt to transmit and result in any of the three possible cases.

2.3 SCA and Distributed Systems in SDRs

The XG program uses the Software Communications Architecture (SCA) [8] as the model for SDR architectures. The SCA specification is published by the Joint Tactical Radio System (JTRS) Joint Program Office (JPO). SCA aims to provide a common software infrastructure for managing radio systems [29], with a fundamental objective of improving the business case for evolving and enhancing communications systems and their procurement. The SCA is one realization of the *Software Infrastructure* aspect of SDRs, with some parts of the *Applications* and *Services* aspect. The SCA defines a logical infrastructure for management and abstraction of physical hardware components. It also defines a standard set of abstractions for software components that form the digital processing portion of a waveform implementation and a general set of services for use by the system. It provides a set of common interfaces for managing, deploying and configuring waveform applications within the system [29]. However, the SCA does not mandate any specific architecture, design or implementation for the radio system hardware or waveform implementation. The SCA structure is intended to provide for portability of applications software by leveraging commercial standards and support the reuse of waveform design modules by building on evolving commercial frameworks. The SCA is therefore based on several related and evolved commercial technologies such as Object-Oriented (OO) techniques in software engineering, CORBA and the CORBA Components Model (CCM).

Distributed processing is a fundamental aspect of the SCA. It uses CORBA to achieve its

portability and platform independence goals. In the context of SCA-based radios, CORBA is primarily used for component location, inter-component communication, and logging [11]. A distributed system has the advantages that components may be on different platforms or/and different locations and still work together. Even if the SCA is not used as the base architecture for a SDR, making it a distributed system allows any component to be implemented on a platform ideal for it. Thus, a PE may be implemented on a separate tamper resistant processor along with the PR whereas an SSR may be implemented on a separate processor, and the transmission hardware may utilize an FPGA, all connected by middleware. This is especially true for current and future SDRs, which could contain a combination of GPPs, DSPs and FPGAs to perform specific functions in a single or across distributed nodes. The use of distributed system concepts in an SDR becomes a necessity in such cases, rather than an optional aspect.

2.4 CORBA and Middleware

Middleware is the generic term for software that resides between an application and the system hosting the application. It is more generally described as software that provides a link between separate software applications. Middleware insulates applications from the software's lower level details so that the developer only has to deal with a single programming interface, letting the middleware handle details such as mediating communication to remote objects [14]. The Common Object Request Broker Architecture (CORBA) is a communications middleware because it insulates an application from details of communication between the objects.

CORBA provides a vendor independent, platform-neutral structure that allows applications to communicate with each other [9]. It is an industry standard middleware that has been widely used in distributed applications. The availability of many COTS CORBA products and its inherent advantages is the reason that the SCA relies on CORBA as the middleware

for implementing SDRs as distributed systems. The major advantages of CORBA are:

- *Location Transparency* - The objects do not need to know where the other objects are located. Most function calls appear as local function calls to the application developer.
- *Language Transparency* - The distributed objects are not required to be written in the same language. Current COTS CORBA products support C, C++, Java, Python, Perl and most high level languages [13].
- *Architecture Transparency* - The properties and idiosyncrasies of CPU architectures (e.g. endianness) are hidden from objects. Today, CORBA can be used for most GPPs, DSPs as well as FPGAs [12].
- *Operating System Transparency* - The objects may be implemented on different operating systems.
- *Protocol Transparency* - The objects do not need to know the network and transport layers being used.

CORBA achieves this by using Object Request Brokers (ORBs) for every request being sent by a client to an object implementation. An ORB is responsible for all of the mechanisms required to find the object implementation for the request, to prepare the object implementation to receive the request, and to communicate the data making up the request. A distributed system can have multiple ORBs for each system, so CORBA specifies the General Inter-ORB Protocol (GIOP) protocol. The GIOP specifies standard transfer syntax (low-level data representation) and a set of message formats for communications between ORBs. The Internet Inter-ORB Protocol (IIOP) specifies how GIOP messages are exchanged using TCP/IP connections and is the most popular GIOP implementation. However, the GIOP is specifically built for ORB to ORB interactions and is designed to work directly over any connection-oriented transport protocol that meets a minimal set of assumptions. Thus, other inter-ORB transport mechanisms are possible by changing the transport layer used by the implementation of the GIOP, based on transport and security requirements.

2.5 SDR and Middleware Security

2.5.1 SDR Security

Due to their distinguishing characteristics, SDRs face unique security threats that are different from those of conventional wireless devices. According to a document recently authored by the SDRForum , SDRs face the following security threats [15]:

1. Installation of Rogue Software
2. Misuse of Software
3. Unauthorized Modification of Data or Software
4. Unauthorized Reading (Accessing private information of the user from the SDR)
5. Masquerade or Spoofing
6. Misuse of Spectrum

Apart from these threats, there are other potential threats such as variants of DoS attacks that exploit the advanced features of SDRs/CRs. The attacks described in [30], [31] and [32] are based on the premise that either the adversaries have the capability to build customized "attack SDRs" or they can modify existing SDRs to suit the requirements of the attack. Thus, security of SDR/CR networks thus also depends on the security of individual SDRs.

Security of SDRs based on the SCA is further complicated by inherent characteristics of a distributed environment and the SCA specification. The SCA Security Supplement [8] describes the characteristics that make implementing security for SCA radios more difficult than traditional radios. These characteristics include the requirements of (from [8]):

- Maintaining assured separation of data and user paths in a common processing environment that involves shared elements.

- Controlling the information bypassed from the RED to the BLACK side of the system.
- Maintaining the integrity of software during transport to the radio, installation into the radio, and during operation of the radio.

2.5.2 Middleware Security

Distributed systems are more vulnerable to security breaches than traditional host-centric systems, because there are more places where the system can be attacked. Distributed systems therefore need to have specific security requirements that take into account the inherent complexities that result from their distributed nature [14]. Authors in [14] note that the primary point of attack is the network, which is subject to passive attacks such as eavesdropping and active attacks such as masquerading and data manipulation. It also has to be noted that the systems are often large and geographically distributed and can have several domains with different security policies [14]. The security paradigm is further complicated due to issues such as mutual distrust between objects, complexities of inherent unpredictable and dynamic interactions, adding and removing of components dynamically and use of OO techniques such as polymorphism and inheritance [14]. And the fact that middleware is implemented to provide multiple layers of abstraction add to the complexity of providing security.

Keeping in mind the above difficulties as well as the fact that distributed systems are increasingly used for critical applications, security issues for middleware such as CORBA have been researched on. The CORBA specification [9] defines the CORBA Security Attribute Service (SAS) protocol and its use within the CSIv2 (Common Secure Interoperability Protocol Version 2) architecture. This protocol addresses the requirements of CORBA security for interoperable authentication, delegation, and privileges. The protocol is intended to be used in environments where transport layer security, such as that available via SSL [33], is used to provide message protection (i.e. integrity and or confidentiality) and server-to-client authentication. The protocol provides client authentication, delegation, and privilege functionality

that may be applied to overcome corresponding deficiencies in an underlying transport. The SAS protocol facilitates interoperability by serving as the higher-level protocol under which secure transports may be unified [9].

2.6 Related Work

With the problem of spectrum scarcity becoming acute due to increasing number of wireless devices, significant research is being performed in the fields of DSA, SDRs/CRs as well as their security. We discuss current research related to different technologies relevant to our thesis in the following subsections.

2.6.1 Dynamic Spectrum Access

DSA technologies today consist of an array of spectrum sharing techniques, driven by digital signal processing as described in [22]. Technical obstacles facing DSA systems such as hardware limitations, coordination and management issues and quality of service issues are discussed in [34] and [24]. Woyach *et al.* [24] also highlight obstacles such as the disruption of current business models, need for regulation and redistribution and pricing which affect the business aspect of DSA systems. Authors in [34] and [35] describe possible solutions to the business and enforcement obstacles for DSA.

2.6.2 Policy-based Radios

Authors in [6] evince that policy-based spectrum sharing has numerous advantages that enable CRs/SDRs to exercise their full capabilities for opportunistic spectrum access. F. Perich [5] describes the design and implementation of a policy-based spectrum access control framework. The framework includes a Policy Enforcer. Perich and McHenry [36] provide details of the architecture used for the implementation in [5]. Denker *et al.* [6] describe the

advantages of the policy-based approach and policies based on the Cognitive Radio (Policy) Language (CoRAL).

2.6.3 Policy Reasoners

Wilkins *et al.* [27] describe a PR based on CoRAL that ensures radio behavior compliant with policies by either approving or disapproving every transmission candidate proposed by the radio based on compliance with current active policies. Thus, the PR also performs the work of a PE in their proposed implementation. Moskal *et al.* [37] describe the interfacing of a PR with traditional SDR architecture and have designed an API for connecting a PR to a SDR. Bahrak *et al.* [18] describe the design and implementation of a PR which we use in our proof of concept implementation. It is described in detail below.

The BRESAP Policy Reasoner

The authors in [18] have developed a policy reasoner named *BRESAP* (Binary decision diagram-based REasoner for Spectrum Access Policies) that utilizes Multi Terminal Binary Decision Diagrams (MTBDDs) to carry out policy reasoning. It uses a set of efficient graph-theoretic algorithms to translate policies into MTBDDs, merge policies into a single *meta-policy*, and compute *opportunity constraints*. Apart from replying with approvals for valid transmission requests, this PR has the capability to respond to either under-specified or invalid transmission requests by returning a set of opportunity constraints. These constraints prescribe how the transmission parameters should be modified in order to make them conform to the policies. *BRESAP* merges all currently active policies into a single *meta-policy*, defined by the authors as "*A meta-policy is the integration of all currently active policies*" [18]. The transmission requests are reasoned against this *meta-policy*. The authors have implemented three different graph-based algorithms for computing responses and opportunity constraints for a transmission request. The Reasoner gives one of the following responses [18]:

- **Yes**: This response is given to a valid transmission request. The transmission is allowed.
- **No, invalid TX request**: This response is given to an invalid transmission request. The transmission is not allowed, but if a set of appropriate changes (opportunity constraints) are applied to the transmission request, the transmission will be allowed.
- **No, under-specified TX request**: This response is given to an under-specified transmission request, as described in the incomplete transmission case in Section 2.2.3. The transmission request is incomplete, but will be allowed if appropriate opportunity constraints are applied to the transmission request.

2.6.4 SDRs/CRs and Security

Rondeau *et al.* [38] describe the design and implementation of a cognitive engine that modifies the transmission behavior of the attached SDR based on sensed information from the environment. The SCA specification [8] defines the requirements for the architecture to be used for SDRs, including security requirements.

Challenges in validation of SDRs are described in [39]. High level security considerations for SDR technology are enumerated in [15]. Kurdziel *et al.* [40] and Davidson *et al.* [41] describe architectures and methods for high assurance implementations of SDR technology. Secure download of policies and software updates to an SDR node from the network are discussed in [42] and [43]. Uchikawa *et al.* [42] describe using SDR technology for providing secure downloads using FPGAs; similar technology may be used to download critical software and updates for the SDR itself. Considerations and requirements for software execution, download, storage, instantiation and installation in an SDR are described in [43]. The authors describe potential attacks due to the unique nature of SDR technology in [32], [30] and [31]. In [32], Brown *et al.* describe denial of service attacks carried out by adversaries equipped with CRs. In [30], Chen *et al.* describe primary user emulation (PUE) attacks.

Clancy and Goergen [31] describe a number of attacks that exploit unique attributes of CRs.

2.6.5 CORBA Security

Use of CORBA for SDRs has been examined in [11], [12], and [44]. Balister *et al.* [11] analyze the performance of an SDR using CORBA and conclude that the use of CORBA is compatible with overall performance needs of an SDR. Casalino *et al.* [12] describe the implementation of CORBA with a customized transport for use with FPGAs in an SDR. Tsou *et al.* [44] observe the latency issues of the transport used with CORBA and show that the use of UNIX domain sockets in place of IIOP can improve performance for a distributed system on the same node.

Security issues and a complete description of the CORBA security services is provided by authors in [14]. Becker *et al.* [13] describe vulnerabilities due to the use of implicit authorization and compare several COTS CORBA ORBS to identify the severity of this vulnerability among them.

Chapter 3

A "proof of concept" Attack on a SDR

We have implemented an attack against a SDR as a proof of concept. Currently, a number of tools exist that enable researchers to build and test SDR/CR technology — e.g., GNU Radio [16], OSSIE [45], and WARP [46]. We decided to use GNU Radio as the signal processing software because it is readily available and it enables us to utilize readily-available RF hardware (e.g., USRP board [17]) to observe changes in operating characteristics of the radio caused by our attack. Specifically, the attack was launched against a SDR that employs a laptop for running GNU Radio and a USRP board for processing RF signals.

This attack makes controlled changes to a SDR's transmission frequency without modifying the radio's signal processing software and circumvents any *checksumming-based* tamper resistance scheme that may be in place to protect it. Checksumming-based methods typically come under integrity-checking methods for providing tamper resistance, and are also called *self-checking* methods. Static self-checking methods check the program code for modifications once before execution, whereas dynamic self-checking methods check the integrity of the program repeatedly as the program is running. We use the features of hardware breakpoints to bypass most checksumming-based tamper resistance mechanisms, and our attack

does not modify the code being executed in any way. This attack demonstrates that the use of tamper resistance techniques for protecting CR software is not sufficient to ensure policy-compliant CR transmissions and that a Policy Enforcer is required to provide this assurance. Sections below describe the attack and the system on which the attack was performed in detail.

3.1 System Architecture for the Attack

For the attack, we considered a simple SDR node. GNU Radio was used as the signal processing software for this SDR node, and the USRP board was the hardware for RF generation. The point of attack was the signal processing software, which was running on a Linux machine. The signal processing software was thus running on a GPP. A simple representation of our system is shown in Figure 3.1, with the point of attack highlighted. Note that the system is not a CR; it is a simple SDR. However, the attack performed by us may be used to perform malicious modification on the Cognitive Engine component in traditional CRs, or the SSR component in policy-based CRs, since they may be implemented on GPPs with similar tamper-resistance mechanisms in place.

3.2 Attack Model

Checksumming methods such as the ones described in [47] and [48] would typically need to be bypassed in order for an adversary to modify the SDR software. One of the preliminary steps in reverse engineering any target software is to perform a debugging-like procedure, i.e., run it step by step or in parts using breakpoints, to discern its control flow. We cannot use software breakpoints, since these use instructions that are put into the code to provide breakpoints. These instructions will be detected by the checksumming mechanism.

Wurster *et al.* [49] describe a generic attack against checksumming based tamper resistance

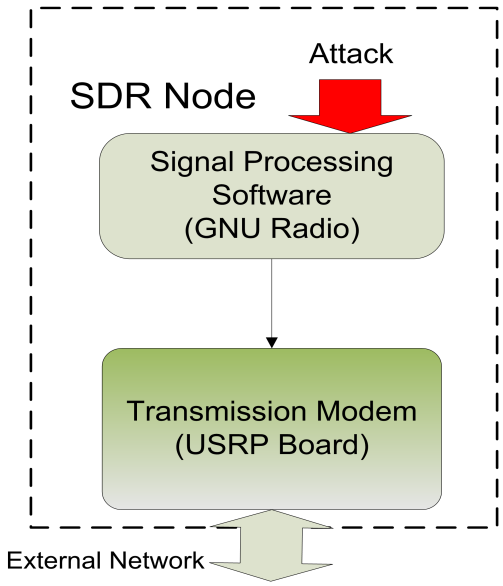

Figure 3.1: High level view of system under attack

methods that uses facilities in memory management units of modern microprocessors. Another approach for circumventing such tamper resistance schemes is to use debug registers, also referred to as hardware breakpoints (HWBPs). Since the exceptions due to HWBPs are caused by hardware and do not involve any modification of code, checksumming cannot detect them [50]. Our attack uses the latter approach because the memory management units exploited in [49] may not be a common feature of all GPPs used for SDRs. In contrast, HWBPs is a feature common to all modern GPPs. Another advantage of using HWBPs is that it does not need extensive kernel modifications as required by [49].

Current processors only support a very limited number of HWBPs (typically 2 to 4 locations), and thus reverse engineering of the target code can be very painstaking if we rely exclusively on HWBPs only. Therefore, our attack also utilizes the contents of floating point registers (FPRs). In the attack, we track the content of a FPR using a single step execution of the code. When we observe that a value that could be related to the transmission frequency is written in the FPR, we trace back to the instruction that was responsible for this write operation. This approach enables us to easily locate the point of interest in the target code

while using only one HWBP.

3.2.1 Assumptions for the Attack

The attack assumes that a SDR employs a GPP or an advanced embedded processor with HWBPs and FPRs. This assumption is valid for GNU Radio, since it uses a Linux machine as its host. GNU Radio is a signal processing package that enables programmers to transmit/receive signals using a generic radio platform by writing code in Python and C++. Thus, GNU Radio fully supports radio reconfigurability. However, for the attack implementation, we assume that only the compiled binary image of the GNU Radio code is available to the adversary.

We assume the "*malicious host*" model for our attack. In this model, once the client code resides on the host machine, the host can make use of any conceivable technique to extract sensitive data from the client code or violate its integrity. The only limiting factor is the computational resources the host can expend on analyzing the code [50]. We assert that the attack mechanism discussed in this chapter would be applicable to any system which satisfies the assumptions given below. Again, we emphasize that the attack assumes that the adversary has access to only the binary code of the signal processing software.

Our attack technique is feasible on any SDR if the following assumptions hold true.

1. *Availability of debug registers*: We use the CPU debug registers to insert HWBPs which enable us to make specific changes to the signal processing procedure without being detected by checksumming based integrity checks that may protect the SDR software. Thus, we assume that debug registers are present in the processor used by a SDR. It is envisioned that most commercial SDRs will employ a combination of a GPP and a digital signal processor (DSP) [3]. Most modern GPPs have debug registers.

2. *Presence of an operating system and access to it*: Our attack uses operating system features for handling HWBPs. Also, in order for the attack to be feasible, an adversary

needs to have access to operating system tasks such as creating new processes. In practice, such access might require changes in the configuration files (or the binary kernel in specific cases) of the operating system, although no changes are required in the case of a general purpose computer. Most existing SDR architecture frameworks, such as the SCA [8] require an operating system.

3. *Tamper resistance scheme*: As mentioned previously, a checksumming-based tamper resistance scheme would be unable to protect the target code against our attack. Obfuscation techniques may be used in combination with checksumming to provide more robust protection against tampering. We believe that obfuscation techniques such as junk code insertion and execution flow mangling would not be a significant obstacle to our attack, due to the way we reverse engineer the target code. Therefore, our attack is expected to be feasible if simple obfuscation mechanisms protect the target code. However, the attack may not be feasible if more robust obfuscation or tamper resistance schemes are in place, such as hardware-based schemes.

Although the platform under consideration (GNU Radio) is open source, we can assert that the attack mechanism discussed in this chapter would hold true for any system which satisfies the above assumptions, or one which can be modified to make these conditions true.

3.3 Attack Mechanism

There are two hardware components in the USRP board that are configured separately to select the transmission frequency. The *AD9862 CODEC* chip on the USRP board shifts up the frequency of the baseband signal using a Numerically Controlled Oscillator (NCO) with a maximum frequency of 32 MHz [51], while the *daughterboard* converts this intermediary frequency up to the final radio frequency. The CODEC chip is configured based on the ratio of the desired frequency shift to the frequency of the NCO. It is this ratio that GNU Radio computes using the FPRs, and the attack's main target is these registers.

3.3.1 Overview of HWBPs and FPRs

HWBPs [52] are designed to provide programmers an easy means of debugging or profiling their code. Most microprocessors have support for such breakpoints using debug registers. As an example, processors based on Intel's IA32 architecture support 4 HWBPs whereas ARM 11 processors have 6 [53]. We will focus our discussions to the IA32 architecture, but similar principles apply to other processors as well. A breakpoint can be set on instruction or data. It can be set up to be triggered on either *read, write or execute*.

When a processor's execution sequence reaches an instruction that has a breakpoint on it, it will generate a debug exception known as a *fault*. After the fault gets generated, the processor returns to the state previous to the execution of the breakpointed instruction. The processor will then pass execution to an interrupt handler, which usually belongs to the operating system. The Linux kernel handles breakpoints by sending a *TRAP* signal to the original process.

FPRs are used to perform floating point number operations. A floating point unit (FPU) is employed and usually operates on a special set of registers that are normally used only for this purpose. Intel processors use the x87 FPU [54]. This FPU provides eight registers that are treated as a stack. Many operations are performed at the top of the stack. At least one operand or the result is written on the top of the stack.

3.3.2 Pinpointing the Code of Interest using FPRs

Normally, before trying to make modifications to a binary image, an attacker would spend an extensive amount of time analyzing the disassembled code in order to understand the execution path and the data flow. However, we tried to find a faster way of discovering a possible attack point. We exploited the fact that GNU Radio's code for dealing with the transmission frequency uses floating point numbers.

Our starting assumption was that if we obtained a log of all floating point operations per-

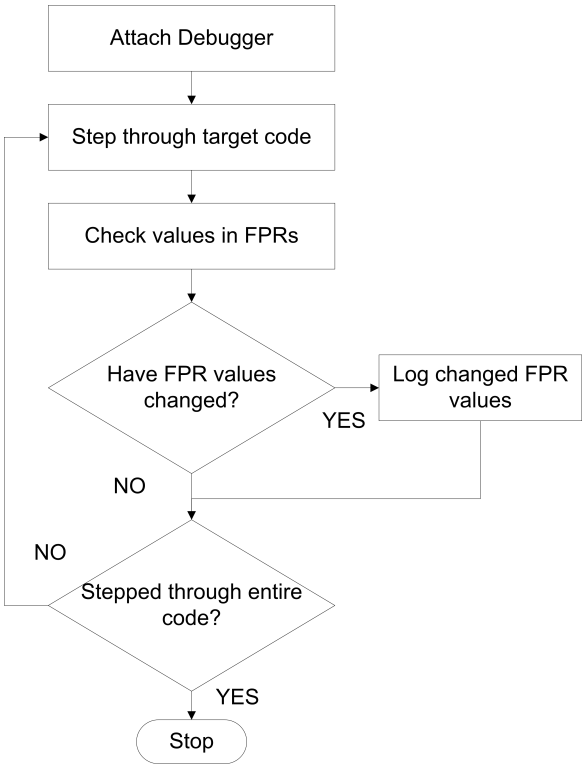

Figure 3.2: Flowchart for logging FPR values

formed by the processor, we should be able to identify those operations that handle the initialization of the transmitting frequency in the hardware. Routines that perform floating point operations belong either to the initialization phase or the data processing phase; in the former the operations occur only a few times, whereas in the latter they are repeated a large number of times. The process that we used to log FPR values is illustrated in Figure 3.2. To obtain this log we stepped through the process instruction by instruction. A second process was used to supervise the target process using the *Ptrace* API [55]. *Ptrace* API is Linux's method of providing debugging capabilities. The parent process forks itself, thus creating a child process. The child calls *Ptrace* to request a debugger's intervention. The parent then calls *Ptrace* to get control over the execution of the child process (this is called "attaching"). The parent can send commands like continue, step, get/set the contents of registers, stop, detach, kill, etc.

We used the `step` command in our attack. After each instruction, the FPRs were inspected

to determine whether a change has occurred or not. In the event of a change, the program counter, instruction data and the value on top of the stack were logged. The log was then analyzed to find which module (executable or dynamic library) each instruction came from. Also, we identified individual repeating blocks and the number of occurrences. We then ruled out the code that originates in libraries with many floating point operations. From this point forward, the code that handles the transmission frequency was easy to identify.

3.3.3 Inserting and Handling Breakpoints

We proceeded to place a breakpoint on the vulnerable instruction. However, since the particular library that contains it is not loaded at the time the execution starts, we had to delay placing the breakpoint until the library is available. To do this, we placed a breakpoint temporarily in the dynamic library loader's code and checked for our desired library. When the library is available, the final breakpoint is placed on the target code. The Linux kernel only allows setting HWBPs from outside the process, using the *Ptrace* API. The procedure is the same as the one outlined for running a process step by step. The *Ptrace* command used is the `poke user` command. This changes the information inside the kernel's user structure for the target process. Among others, this structure contains the values of the debug registers.

When the breakpoint is hit, the kernel sends a *TRAP* signal to the process. Because the process is being *Ptraced*, the debugger gets to handle the signal. After control is handed over to the debugger, it performs the following tasks:

- Change the radio's transmission frequency as explained in the Section 3.3.4.
- Set the RF flag in the *EFLAGS* register in order to avoid triggering the same breakpoint again, thus avoiding an infinite loop [52].
- Resume execution of the child process.

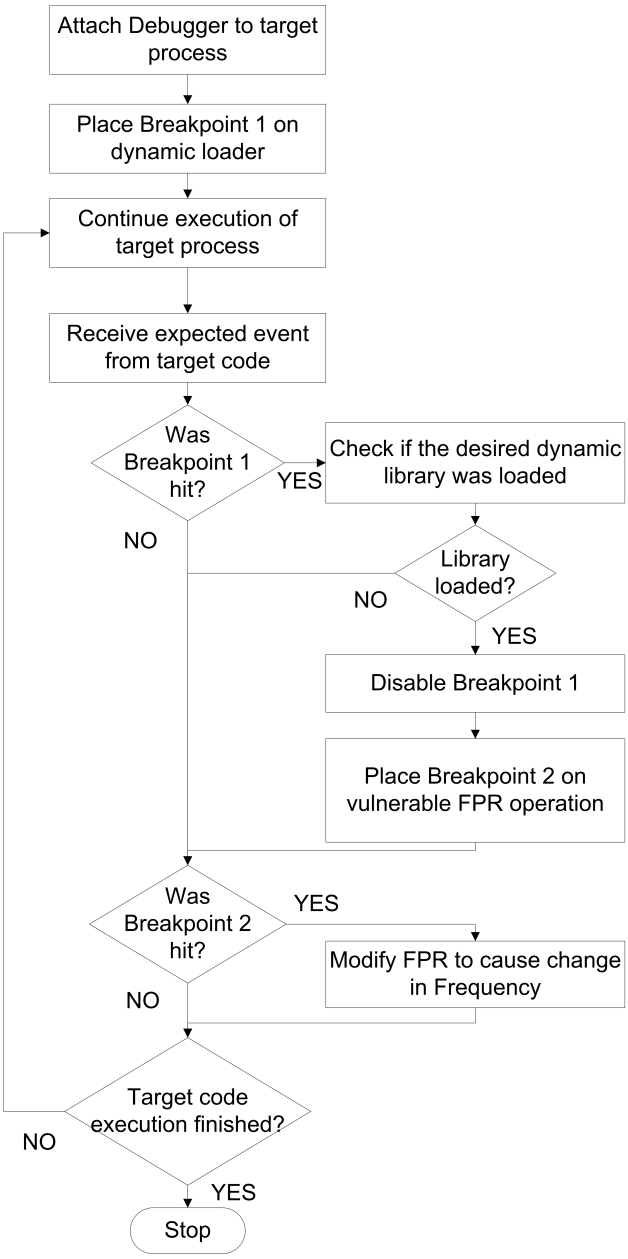

Figure 3.3: Flowchart of the attack procedure

3.3.4 Changing the Transmission Frequency by Altering the FPR Values

To change the transmission frequency, all we had to change was the value of the appropriate FPR. At a specific point during the execution, the NCO ratio in the USRP board is computed and stored on top of the FPR stack. Changing this value causes a change in the transmission frequency. To make the change, we used the `get fpregs` and `set fpregs` *Ptrace* commands. It can be seen that the way the attack was carried out would thwart checksumming-based integrity checking tamper resistance mechanisms, if they are used for hardening the signal processing software of the SDR. The attack thus exploits the reconfigurability of SDRs to generate transmissions that may not be legal. The effect of this attack on the transmission behavior is described in the experimental evaluation in chapter 6.

Chapter 4

The Policy Enforcer

4.1 The Policy Enforcer as part of PCCs

Authors in [28] segregate a policy-based radio into four functional components- the *Sensor*, *Radio Platform*, *Policy Conformance Reasoner* and *System Strategy Reasoner*. This approach defines the Policy Conformance Reasoner as the component required for assuring that the radio use of spectrum conforms to policy, utilizing machine-readable policies. The Policy Conformance Reasoner thus manages accredited policy information. It also interprets the policy language and reasons based on accredited policy and related background knowledge to determine whether the proposed spectrum use is policy-conformant [28]. Authors in [6] describe the design of a Policy Reasoner following this approach. Authors in [36] alter this approach and define a set of policy conformance components (PCCs) that are responsible for assuring that the radio operates correctly and does not cause harmful interference by enforcing that the SSR configures the radio approved states and by filtering illegal transmission requests. It has to be noted that in both approaches, the Policy Conformance Reasoner and the PCCs represent standardized, accredited components which are system independent and responsible for ensuring policy-compliant behavior. The PCCs described in [36] consist of the *Policy Manager*, *Policy Database*, and *Policy Reasoner* and *Policy Enforcer*. The Pol-

icy Reasoner and the Policy Enforcer are the on-node components responsible for ensuring policy-conformant behavior in this case. We describe their functions, and the advantages of separating the reasoning module (Policy Conformance Reasoner or the Policy Reasoner) and the enforcing module (Policy Enforcer) for policy-based radios in the following subsection.

4.1.1 Separating the PR and the PE

The authors of [6] and [28] describe an architecture that utilizes a PR for reasoning as well as enforcement. In such a case, the PR includes a module similar to the PE which performs the task of ensuring policy conformant behavior by monitoring changes requested by the SSR to the transmission modem. As described above, the PR applies reasoning to verify if transmission parameters from the SSR are valid for the currently active policies. Authors in [18] and [27] describe algorithms and languages that may be used in a PR to achieve this. As seen in these discussions, the PR needs to perform complex calculations to merge all active policies and also to evaluate and reason transmission requests. Adding enforcement functionality to the PR would increase its workload, and make it a more complex module. The PR would also then need to be tightly coupled with the transmission hardware. This complexity might hamper efficiency as well as make it hard for accreditation.

On the other hand, separating the PR and the PE allows the PR to only focus on reasoning transmission requests and arranging active policies. This allows the PE to only perform the function of enforcing the policy. Also, the PE may utilize a cache of recent PR decisions so that the PR need not repeat reasoning for identical transmission requests, improving system efficiency. The PE can have complete control over the transmission hardware. The PE can be accredited easily as it is a relatively small module with simple functionality. It can also be easily made tamper resistant or implemented on tamper resistant hardware.

4.2 An Ideal Policy Enforcer

The primary function of the Policy Enforcer component in a policy-based CR is to avoid causing harmful interference to primary users by filtering inappropriate transmission commands sent to the transmission modem. With respect to the policy-based radio architecture described in Section 2.2.2, we define the following requirements for an "ideal" PE:

- *Real-time communication with the Policy Reasoner*: The PE sends the transmission requests to the PR, which responds with the decision to either allow or deny the transmission. For incomplete transmission requests, the PR may give a feedback with allowed parameter values which the PE forwards to the SSR. It is therefore necessary that the PE can always communicate with the PR.

- *Real-time monitoring of transmission signals*: The PE filters all transmission signals from the SSR going to the transmission hardware. It then allows or denies transmission based on previously stored decision values from its cache, or after interaction with the PR. It may need to convert the transmission parameters into a suitable format before sending it to the PR. These operations have to be performed in the shortest time possible to provide real-time validation of transmission parameters.

- *Ability to stop transmission*: If the transmission signals from the SSR are deemed to not follow currently active policies, the PE will not allow the transmission hardware to transmit — thus stopping non-conformant transmissions. Ideally the PE should have complete control over the communication hardware to enforce policies.

- *Minimum overhead to the communication process*: The PE should be entirely independent from the component generating the transmission signals (i.e., SSR) and should cause minimum overhead to the transmission process. Thus, the signal should be allowed or disallowed in the time permitted by the opportunistic spectrum access standards.

- *Placement within the system*: The PE has to be placed at a location in the system such that it can easily perform its expected functions. It has to have access to the transmission modem as well as the PR, and the placement needs to ensure that communication between the PE and these components is not cumbersome.

- *High assurance requirements*: Since the PE is the block that is responsible for enforcing policies, it is imperative that the PE be tamper resistant. The PE has to be always invoked, non-bypassable, tamperproof and verifiable. A "*defense in depth*" approach (as described in [56]) would be ideal for a PE.

- *Sufficient independence from other components of the SDR*: The PE should be independent of the SSR and PR implementations and only monitor the transmission requests sent over to the modem. This is important since it is likely that the SSR implementation is not certified or accredited. New advances in technology may directly be implemented in the SSR without putting it through the certification process, with the assurance that the PE will block policy-nonconforming transmissions.

- *Secure communication with other components*: As described in the interactions between policy-based radio components in Section 2.2.3, there will be communication between the components. The PE needs to exchange critical control messages with the SSR and the PR that determine the radio's transmission behavior, and these messages need to be protected against threats such as message manipulation. This is especially essential if the PE is part of a distributed system.

4.3 Approaches for Implementing a PE

As described in Section 4.1, the XG vision describes the functional components of a policy-based radio, which have been further described in architectures based on the XG vision [6] [5]. It has to be noted that the XG vision does not prescribe the placement of these functional blocks in hardware or software on the radio. This is left to be determined by the designers

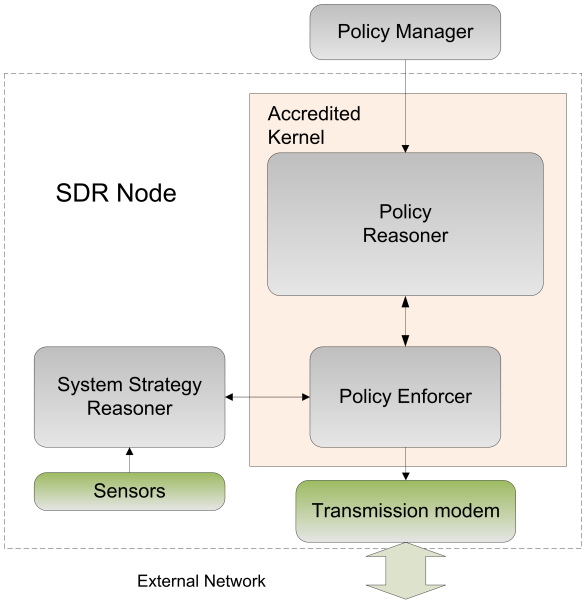

Figure 4.1: PE implemented as a "man-in-the-middle"

of each individual system [28]. As described in the "ideal" PE requirements in Section 4.2, the placement of the PE should be such that it can perform its functions with ease. For the architecture described in Section 2.2.2, we have identified two locations where the functional block of the PE can be implemented. Figure 4.1 and Figure 4.2 show these two possible locations in the policy-based radio architecture. Figure 4.1 shows the PE placed as a "man-in-the-middle" (*MITM*) between the transmission modem and the SSR. Figure 4.2 shows the PE placed as a firewall, monitoring signals sent from the SSR to the transmission modem. We describe both these approaches in the following subsections, and also describe their design and implementation.

4.3.1 The *RF Firewall* approach

To provide for increased tamper resistance, the PE can be placed beyond the point where the transmission parameters of the radio can be easily modified in software. For this, the PE can be on the *reconfigurability boundary*, as close as possible to the transmission hardware.

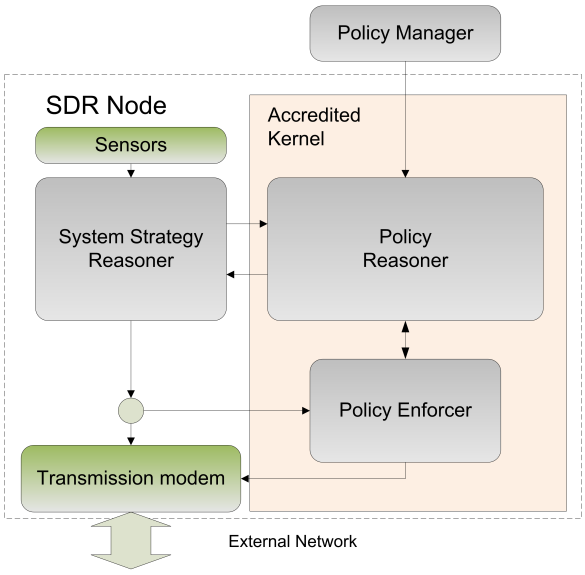

Figure 4.2: PE implemented as an *RF Firewall*

Within the *reconfigurability boundary*, the CR system is composed of reconfigurable software-driven components. In contrast, beyond the boundary, the CR system is composed of mainly firmware — or hardware-driven components whose parameters are hardwired into the system itself, and thus unauthorized tampering of those components is equally difficult as tampering with conventional radios. Because it is difficult for an adversary to tamper or manipulate the hardwired components, one can be assured that transmissions once validated by a suitably tamper resistant PE at this location will not be modified at any later stage in the radio. For the policy-based radio architecture as described in Section 2.2.2, it can be seen that the *reconfigurability boundary* is between the SSR and the transmission modem. Thus, Figure 4.2 shows the PE placed at this location. We call such a PE implementation as the *RF Firewall*. Thus the *RF Firewall* is located between the two different regions — one containing reconfigurable software components, the other having hard to manipulate hardwired components. These regions can be considered as two networks, since each region has modules connected with each other, similar to nodes in a network. An *RF Firewall* PE is comparable to a traditional firewall in the following ways:

- *Single point of defense*: An *RF Firewall* is essentially a single point of defense, protecting one network from another, thus similar to traditional firewalls [57].

- *Implements network policies*: Traditional firewalls enforce network security policies [58]. Firewalls do so by evaluating each network packet against a network security policy. Network security policies identify a network's resources and threats, define network use and specify action plans for policy violations. Analogously, an *RF Firewall* enforces spectrum access policies by evaluating the transmission parameters against the policies. A spectrum access policy specifies available spectrum resources, defines how they should be used, and may define authorized transmission parameters.

- *Gateway to all communications*: A firewall is a non-bypassable gateway to all communications between the two networks on the boundary of which it lies. The *RF Firewall* also acts as a non-bypassable gateway monitoring all traffic on the *reconfigurability boundary*.

Through continuous research and development, today's network firewalls have become very sophisticated systems that are effective in network access control and policy enforcement. Some of the principles used in building today's sophisticated firewalls may help us in the difficult task of designing and developing PEs. While the *RF Firewall* PE is analogous to traditional firewalls in many aspects, it also differs in some aspects due to the nature of the boundary on which it resides. Despite the differences between the traditional firewall and this PE, the two share a number of fundamental attributes (some of them discussed above). Table 4.1 highlights comparable functionalities of traditional firewalls with the functions of an ideal RF Firewall PE.

4.3.2 Working of the *RF Firewall*

The working of our *RF Firewall* can be summarized as follows:

Table 4.1: Comparable functionalities of the *RF Firewall* PE and traditional firewalls

Function of an Ideal PE implemented as an RF Firewall	**Comparable function of traditional firewalls**
The *RF Firewall* continuously monitors all transmission requests/control signals in real-time over the reconfigurability boundary.	A traditional firewall continuously monitors all packets (or connection setups for circuit level firewalls) over the network interface.
The *RF Firewall* can stop transmission for transmission requests not conforming to spectrum access policies.	Firewalls disallow packets not conforming to network security policies across the network interface.
A *RF Firewall* has high tamper resistance and is independent of the process generating the transmission requests.	Hardware firewalls are tamper resistant blocks that only monitor traffic over the network interface, independent of the processes generating the traffic.
The *RF Firewall* can enforce different policies as specified by the PR.	Firewalls enforce network security policies which can be reconfigured based on requirements.

1. SSR checks for available spectrum, determines the optimum transmission parameters (e.g., frequency, modulation, and power) and sends a transmission request that includes those parameters to the PR.

2. The PR approves/disapproves or suggests changes to the transmission parameters specified in the request.

3. Once the PR approves the transmission request, The PR sends the parameters of the approved request to the PE, while also sending permission to the SSR to transmit.

4. The PE checks this transmitted signal, and verifies that it is indeed following the parameters approved by the PR.

In our implementation, the *RF Firewall* only performs the function of enforcing the policy. The only communication it has with the PR is getting the approved policy input, and indicating policy breaches to the PR if required. Depending on the actual implementation of

the CR, the need for two-way communication between the PR and the *RF Firewall* may be unnecessary. The *RF Firewall* may only need to receive updates from the PR. Unapproved transmissions in our architecture are handled by severing communication to the hardware device and optionally logging to the PR. The *RF Firewall* is thus a unit focused only on enforcement. It is also relatively small in size since it does not have any operations to formulate policy requests. These properties make it easy to be made tamper resistant.

4.3.3 Implementation of the *RF Firewall*

Our implementation of the *RF Firewall* is a proof of concept design that implements a minor subset of the features required for an ideal PE. The implementation also allows us to approximate performance in real world applications in terms of memory usage and processing overheads. In our design, we assume that the hardware is the point in the system that is least vulnerable to tampering. The SDR however is assumed to be prone to tampering and can send erroneous or malicious transmission parameters to the hardware components. The PE acts as a gateway between these two, enforcing policies specified by the PR.

The *RF Firewall* was implemented on a SDR that uses GNU Radio and a USRP board, as shown in Figure 4.3 Inside the SDR, there would be a reconfigurability boundary in its structure after which the radio software cannot modify the radio parameters, as described in Section 4.3.1. This could be a boundary just before the DAC/ADC in a real SDR. For our implementation, we considered the USB port to be this boundary. For the architecture of the USRP board, actual values for the transmission parameters such as center frequency are not sent directly over the USB communication. The values are encoded into a set of hexadecimal values along with essential information about the daughterboard and other settings being used. By using GNU Radio source code we developed a function that extracts the actual value of frequency from the "encoded" values being sent to the hardware. By reverse engineering the encoding process, we were thus able to extract the correct values for the frequencies in real-time, for comparison against active policies.

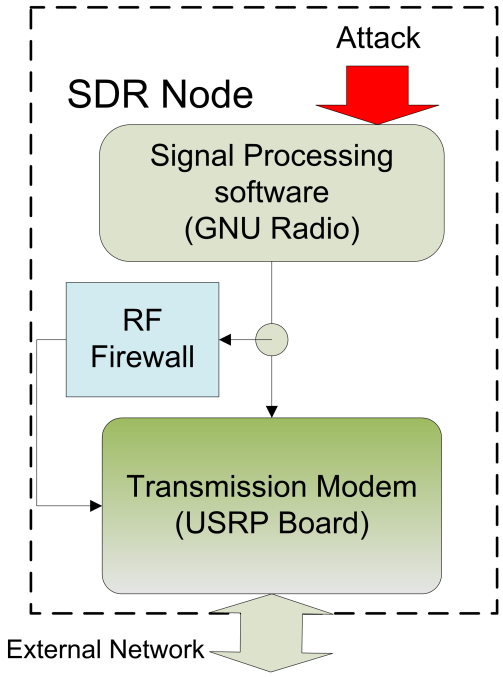

Figure 4.3: High level view of *RF Firewall* implementation

For our implementation we focused only on the frequency parameter to act as a proof of concept design. However, the process of extracting the actual values can be applied to other parameters as well, as long as there is a prior knowledge of the way the hardware interprets the encoded control signals being sent to it. By effectively coding functions that extract the actual values of the parameters from control signals being monitored, an RF Firewall can enforce any type of parameter in a fully specified policy.

The algorithms that convert control signals to the actual value for any transmission parameter would be specific to the hardware. They would therefore have to ideally be provided by the manufacturer of the hardware. These algorithms would have to account for all possible parameter values and corresponding control signals that the hardware is capable of understanding. This would ensure that all possible control signals for transmission parameters would be interpreted correctly by the PE. Our *RF Firewall* implementation performs the following three steps:

1. *Initialization*:

   Before monitoring, the program must recognize what frequencies are allowed to be transmitted. At several instances in the system (as described in Section 4.3.2) the PR would update the *RF Firewall* with currently active policies. For an actual CR system, this policy update would include a multitude of permitted values for different parameters, which could come in a variety of policy languages depending on the system setup. As mentioned before, our system focuses only on the frequency aspect of policy enforcements. It was assumed that permitted frequencies would come in the form of a range of allowed frequencies or a single allowed frequency. Because we did not use an actual PR for this system, we left this to user input. User input was collected at the initialization stage of the PE and stored to act like actual policies.

2. *Monitoring Process*:

   After collecting the allowed frequencies, our implementation used the Linux kernel module *usbmon* to monitor the traffic over the reconfigurability boundary. This module

allowed us to see the raw data being sent to the hardware over the USB. Since this raw data included the control signals being sent by GNU Radio to the USRP, we could obtain control signals for every change in transmission behavior. Now, using the previously described function that extracts actual values from control signals, we correctly identified the frequency at which GNU Radio was trying to transmit. Once identified, we evaluated this frequency against the currently active policy by a lookup against the list of permitted frequencies for this policy.

3. *Enforcing Policies*:

   If the frequency was found in the lookup, the *RF Firewall* would take no action and allow the transmission. However, if the *RF Firewall* determines that the control signals being sent refer to a frequency that is not within the allowed policy specified by the PR (in our case the user input); it can perform a suitable action to stop the transmission. For our experiment, the enforcer program identifies which process is accessing the radio hardware and kills it. This is done by identifying what USB bus and port the USRP is attached to, then identifying the process ID (PID) utilizing the USB. Once identified, the Linux kernel command 'kill' is used to interrupt the process and end it. It is not determined whether the behavior of the radio is malicious or simply erroneous, however once a process is flagged as trying to perform unauthorized transmissions, it is not allowed to continue execution.

4.3.4 The "Man-In-The-Middle" Approach

As shown in Figure 4.1, the PE can be implemented as a *MITM* between the SSR and the transmission modem. This approach has the advantage that all transmission signals from the SSR to the transmission modem have to go through the PE. The PE will first check if these transmission signals conform to currently active policies (using the PR), and only then forward them to the transmission modem. The PE thus acts as the middlemen for all communication between the SSR and the PE. For this approach, it is obvious that there

would be increased interactions with between the PR and the PE, unlike the *RF Firewall* approach. The *RF Firewall* only got current parameters from the PR after policy updates. The *MITM* approach would however forward requests from the SSR to the PR, and get responses to make its decisions Such a PE would therefore need to use some mechanism such as a cache to reduce repeated identical interactions with the PE. The use of a cache mechanism and its implementation for such a PE is discussed in the next subsection.

4.3.5 Implementation of the Cache-based PE

We have implemented a cache-based PE. For each transmission request by the SSR, the PE will search its cache of recent transmission requests to see if it was recently approved. Since only approved requests are stored in the cache, a *cache hit* implies that the transmission is approved. The PE will then apply the configuration associated with that request to the transmission modem and the radio starts transmitting. If there is no hit in the cache, then the PE consults the PR to evaluate the request. The flowchart for the system is shown in Figure 4.4. The cache-based PE has to perform the following functions:

1. Cache only approved transmission requests from recent transmission requests generated by the SSR that were approved by the PR.

2. Search the cache for an exact replica of the request based on the transmission request parameters and consult the PR if there is a *miss* in searching the cache.

3. Get the response from the PR which could be transmission responses in the form of approved, denied, or opportunity constraints, and forward them to the SSR. Approved requests should be added to the cache.

4. Only transmit on approved transmission responses, and must not carry out denied or under-specified responses.

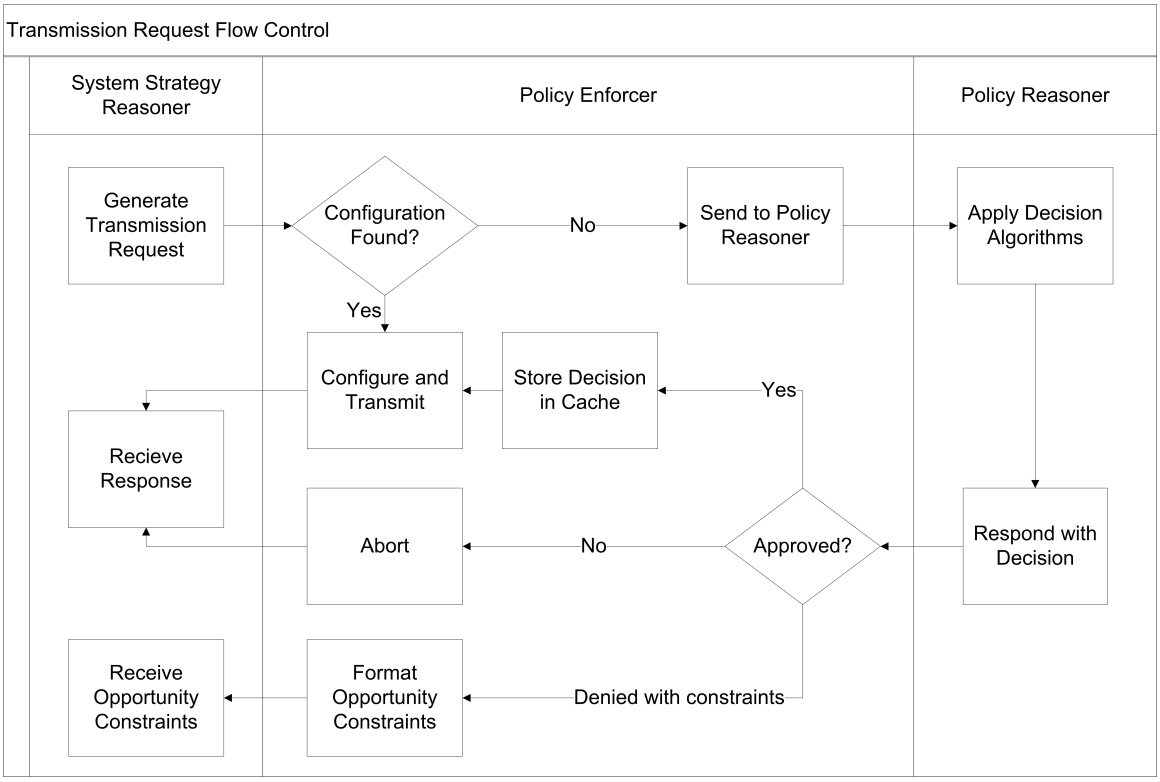

Figure 4.4: Flow of control for the cache-based system

It can be seen that maintaining *cache coherency* is an important requirement in such an implementation. If the PE allows a transmission request based on its cache that should not be allowed based on currently active policies, it fails in its primary function of not allowing illegal transmission behavior. Authors in [5] have implemented cache mechanism in the PE using a set of pre-approves states with associated validity time periods. We however use a novel approach as described in Section 4.3.6.

4.3.6 Maintaining Cache Coherency

We use a C++ *hash_map* to implement the cache as an associative array, and store key-value pairs of the transmission requests and corresponding responses from the PR. We only put approved requests into the cache. We concatenate all values passed in the transmission request structure to create a key representation of the transmission request for use in the associative array. This will distinguish requests that differ from each other even in a single parameter. In addition to the values of the transmission request, we also include a representation of the *meta-policy* in the key as well. We convert the *meta-policy* available to the PE into a hash value called the *metaPolicyHash*. We attach the *metaPolicyHash* to each transmission request that is processed, by concatenating it with the other parameters. The *meta-policy* for which the request was approved will thus be included with the key in the cache.

When the PM updates the PR, the *meta-policy* value in the PE also gets updated. The PE thus always attaches the most recent representation of the meta-policy to the approved transmission request for creating the lookup key. After a policy update, since the key is now different than the key for a similar request (same parameter values) before, there may not be a corresponding value in the cache. Thus, the same requests approved with a previous policy will not be approved with the new policy. Our approach thus does not need an explicit cache time period as in [5], or any other cache flushing mechanism. A "First in, First-out" (FIFO) or least recently used (LRU) algorithm can be used to remove old keys from the cache if there is a limitation on the size of the cache.

4.3.7 Comparison of the Two Approaches

It can be seen that both approaches have distinct advantages, and both of them can be used to efficiently enforce policies in policy-based radios. The *RF Firewall* has the advantage that it is located at the boundary between the reconfigurable software components, and the inherently tamper-resistant hardware components. One can therefore be assured that transmissions once validated by the *RF Firewall* will not be modified at any later stage in the radio. Also, the *RF Firewall* approach does not need to create or modify transmission requests to send them to the PR; it only depends on the current parameters from the PR, making it a simple component to implement. The *RF Firewall* works on the lower level of parameter values, and not on the high level abstraction of transmission requests. The *RF Firewall* basically follows the "*monitoring*" approach to policy-enforcement, with the signals being monitored, and then their validity determined. The signals may be allowed to reach the transmission hardware in the time they are getting validated. The transmission can then be stopped later if deemed invalid. This approach provides the least interference to the transmission process from the SSR to the transmission modem. However, the *RF Firewall* approach does require a direct link between the SSR and the transmission modem. If, for some reason, the *RF Firewall* fails to function properly, SSR signals to the transmission modem will not get monitored, and illegal transmissions may take place.

The *MITM* approach on the other hand does not have any path from the SSR to the transmission modem. All signals from the SSR to the modem have to go through the PE. If the PE malfunctions, it would not let any control signals reach the transmission modem, and no transmission can take place. The PE thus follows the "*pro-active*" approach of first validating the transmission signals, and only then forwarding them to the transmission hardware, thus safe from the disadvantage of the *RF Firewall* described above. However, the PE in this approach will in most cases be interacting with a transmission modem component that includes an SDR. This SDR will convert the high-level abstraction of transmission requests into low-level parameters that will then be sent to the transmission hardware. There

is thus the fear that the SDR beyond the PE can be modified and illegal transmissions can be carried out. It therefore has to be ensured that the transmission modem component for Figure 4.1 is a "*black box*", which cannot be maliciously modified, even if it contains SDR components. The PE for the *MITM* approach has to create transmission requests to send to the PR, and in most cases will also involve some mechanism such as a cache to prevent the burdening of the PR with repeated requests. The PE in this case is thus far more complex than the simple *RF Firewall*.

For both approaches, the PE ensures that the radio behavior is compliant with system policies. It can be seen that the *MITM* approach is better suited for a distributed policy-based CR implementation. This is due to the fact that monitoring signals to the transmission modem (as in the case of the *RF Firewall*) while allowing direct connections to the transmission modem in a distributed system is not convenient. It is more convenient to only have limited access to the transmission modem, the *MITM* approach provides this as only the PE has access to the transmission modem. We therefore use the cache-based PE in our prototype distributed policy-based radio implementation discussed in the next chapter.

Chapter 5

Securing a Distributed Implementation of a CR Using CORBA

In this chapter, we describe the implementation of a distributed policy-based CR. We also highlight potential vulnerabilities in a distributed CR system using CORBA, based on our implementation. Some of the vulnerabilities discussed here are highlighted in the SCA specification and guidelines have been provided to counter them. However, these guidelines may not always be followed in SCA non-compliant implementations. Even after following guidelines, there can be vulnerabilities in certain implementations which we discuss in this chapter. We also discuss the effect of these vulnerabilities, particularly focusing on a distributed CR system.

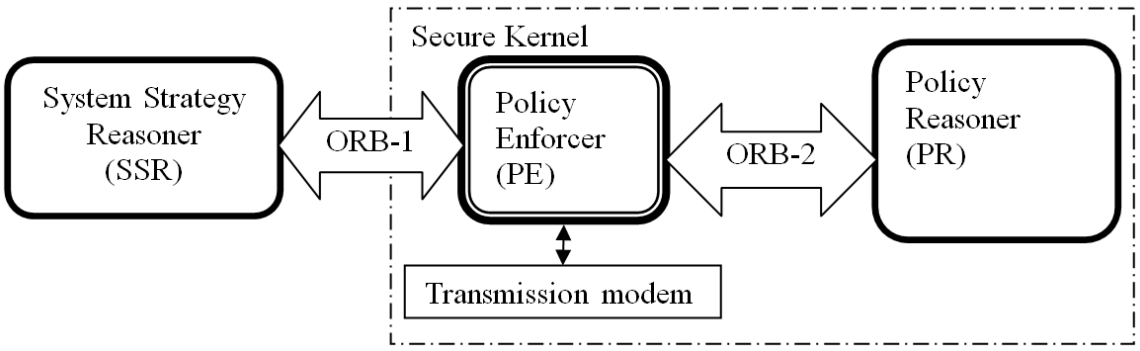

Figure 5.1: High level view of the distributed system implementation

5.1 Implementation of a Distributed Policy-based CR

We have developed an actual distributed CR system based on the system architecture described in Section 2.2.2, which demonstrates the advantages of the proposed cache-based PE. The architecture also lets us identify potential vulnerabilities due to the use of CORBA in a distributed CR system and implement countermeasures for the same. The performance penalty of providing security in a distributed CR implementation is experimentally evaluated for this implementation. Figure 5.1 shows a high-level view of our implemented system.

The components interact with each other as described in Section 2.2.3. We use the *BRE-SAP* PR (described in Section 2.6.3). A *dummy* SSR was implemented to make random transmission requests one after the other after random wait times. This could easily be replaced by an advanced cognitive SSR, which can use sensors to sense the environment and create transmission requests as described in [38]. We use a *MITM* PE which uses a cache to improve performance, and its cache mechanism can be switched ON or OFF. The PE is connected to a transmission modem which utilizes GNU Radio and an attached USRP board for generating RF signals. We used COTS ORBs to connect the components for our architecture. We also defined interfaces that the components use to communicate with each other. Any component that is implemented using the interfaces defined (using CORBA Interface Definition Language) for this system can replace the components we have used. Our implementation can be regarded as a basic policy-based radio framework that could be used

as the basis for a more advanced cognitive radio system. The following subsections describe the various major components of our implementation.

5.1.1 Middleware

The *BRESAP* PR was implemented in Java whereas our PE and SSR were implemented in C++. For connecting the PE and the PR, we used the Java IDL ORB [59]. Between the PE and the SSR, we used MICO (version 2.3.13) [60] and omniORB (version 4.1.4) [61] ORBs. Thus, for Figure 5.1, ORB-1 is MICO (or omniORB) whereas ORB-2 is Java IDL ORB. We thus use multiple ORBs for our system. There may be technical reasons for the use of multiple ORBs in an actual SDR, e.g. advanced support for a programming language or platform in one ORB as compared to the other. For a SDR that can contain GPPs, DSPs and FPGAs in a single system, use of multiple ORBs is a common phenomenon. The use of multiple ORBS can also be for security reasons as described in the SCA security supplement [8]. Some components could be a part of the BLACK side or in a separate tamper resistant kernel, and use a different ORB than the one used between components in the RED side. For simplifying our experimental evaluation, we concentrated on changing the security properties and transport mechanisms only for ORB-1.

5.1.2 Interfaces for Communication between Components

The CORBA object model is based on the *Interface Definition Language* (IDL), which enables a formal specification of the types used in the object model as well as interfaces from objects. The IDL is used to specify object interfaces independently of a specific programming language. The mapping of IDL interfaces to a particular programming language is defined by IDL language mapping which is provided by all COTS ORBs. We have defined an IDL for our architecture, which defines the interfaces provided by each component. We have also defined the *transmission request* and *transmission response* as the `TXRequest` and `TXResponse` types

in the IDL and provided enumerations for the transmission modes and decisions from the PR. The IDL definitions are shown in Table 5.1.

We have used a module called `SDRNode` to define our structures and interfaces to prevent namespace contamination and ensure uniqueness for our interfaces. As can be seen from the IDL definitions, the transmission request structure (`TXRequest`) consists of transmission characteristics such as transmission frequency, power and the transmission mode. It also includes environmental and sensed information such as transmission time, location and the peak-sensed power. It is customized for the *BRESAP* PR being used. The `nonce` and `metaPolicyHash` elements are used by our cache-based PE implementation. The transmission response (`TXResponse`) consists of the decision from the PR and a `nonce` as well as opportunity constraints (if any).

The PE and the PR specify two interfaces each. These are the functions that could be called for any implementation of the PE and the PR based on the IDL definition. The PR object provides two functions that can be called. These are:

- *consume_request*: This function takes in a *transmission request* as an argument, and returns a *transmission response* after reasoning the request.
- *updateMetaPolicy*: This function is implementation specific for our cache based implementation. This function informs the PR that the active policies have changed, and asks it to update its *meta-policy*.

The PE also provides two functions. These are:

- *consume_request*: This function takes in the *transmission request* as an argument. This function would be called on the PE by the SSR. The PE would then use its cache to verify if the request satisfies current policies. It can also call the *consume_request* function on the PR and forward the request to the PR to get the response. If the request is approved, the PE utilizes the transmission modem to transmit (based on the parameters of the request). It also returns the *transmission response* to the SSR.

Table 5.1: CORBA IDL for our distributed system

```
module SDRNode{
enum TXMode {
GMSK,
CPM,
D8PSK,
QAM8,
DBPSK,
DQPSK
};
enum PCRDecision {
APPROVED,
DENIED,
DENIED_W_CONSTRAINTS
};
struct TXRequest{
long Frequency;    /* Need 123000000 for 123 MHz */
short Power;
TXMode Mode; /* cpm, d8psk, qam8, dbpsk, dqpsk, gmsk */
long time;
string location;
short peakSensedPower;
long nonce;  /* corresponding to the time the request was sent */
long metaPolicyHash;
};
struct TXResponse{
PCRDecision Decision;
long nonce;  /* same nonce as the request, for verifying */
string constraints;
};
interface PolicyEnforcer{
TXResponse consume_request(in TXRequest request);
void setMetaPolicyHash(in long metaPolicyHash);
};
interface PolicyReasoner{
TXResponse consume_request(in TXRequest request);
long updateMetaPolicy(); //also returns the metapolicy
}; };
```

- *setMetaPolicyHash*: This function is used to provide the most recent *meta-policy* to the PE.

5.1.3 System Implementation

The SSR is implemented as a C++ object that has access to the PE object using the PE's inter object reference (IOR), and can call the *consume_request* function on the PE. The SSR creates `TXRequest` structures based on a pool of transmission characteristics that can be changed to change the percentage of request approvals. The SSR thus acts as a client while the PE object acts as the server, servicing *consume_request* function calls from the SSR. The PR is implemented as a Java program that creates a PR object which has access to the *BRESAP* API. This object converts the `TXRequest` argument into a string that the *BRESAP* understands and converts the response from *BRESAP* into a `TXResponse` for returning to the calling object. The PR thus acts as a server, servicing *consume_request* function calls from the PE. It can be seen that the PE acts as both a client and a server, for the PR and the SSR respectively. Ideally, only the PE has access to the PR and the SSR has access to only the PE interfaces.

It can be seen that this policy-based radio implementation should work as a policy-conformant radio, provided the PE functions properly, and the system is secure from malicious modifications. However, there may be security vulnerabilities that need to be countered for ensuring the system is secure. These are discussed in the next Section.

5.2 Potential Vulnerabilities

5.2.1 Bootstrapping

Vulnerability

In the distributed system described in Section 5.1, we assume that the objects are connected to each other and are passing messages between each other. However, how the objects were initially connected was not discussed. The problem of how the client obtains the first object reference of the server is called the *bootstrapping* problem [62]. This problem exists in all distributed systems. In distributed systems using CORBA, *interoperable object references* (IORs) are usually used to provide server side information to the client side. Every ORB has the ability to create a human readable, text-based version of the object reference call, known as a *stringified* IOR. Stringified IORs can be created using one ORB and read and interpreted by another ORB, especially for IIOP based IORs [29]. The problem is how to provide this IOR to the client object when it is started for the first time at system startup.

This problem is usually solved by providing the IOR outside of the CORBA framework using file-based URLs, object URLs or command line arguments [62]. If a CORBA *naming service* [9] is used, the client only needs the reference to the naming service which may be provided using any of the above methods. It can then use the *resolve()* method to get the reference to the object it wants to connect. It however has to be ensured that the reference to the correct naming service is provided, and that the communication medium as well as the naming service is not compromised. Any implementation thus has to ensure that the bootstrapping process is secure, and cannot be compromised.

Effect on CR Security

The bootstrapping process has to be especially secure in a SDR since all the security systems may not be fully active at system startup. If an attacker can target the bootstrapping

process, and modify the initial connections such that the objects bind to a rogue naming service instead of the original one, the attacker can force the SDR objects to connect to malicious objects. This can lead to denial of service (*DoS*) attacks, as well as bypass certain security measures by not letting security objects connect to the distributed system.

5.2.2 Inter-object Communications

Vulnerability

Apart from function call messages, there will be connection setup, access control and security messages between objects in a distributed system. The transport mechanism of the middleware used thus has to be made secure from eavesdropping and/or data manipulation attacks. The transport mechanism used by CORBA therefore needs to provide integrity and confidentiality. The communication has to also be secure from *DoS* attacks. A secure distributed system thus needs to provide authentication, integrity and confidentiality for its transport mechanism. Integrity and confidentiality for the transport mechanism are required by both the CORBA security specification [9] and the SCA security supplement [8]. By default, most COTS ORBS use IIOP as the transport mechanism. IIOP is also the preferred mechanism when multiple ORBS communicate with each other. However, the use of IIOP does not provide any security in terms of integrity and confidentiality since the data is passed over TCP without encryption. The data being transferred over the middleware thus has to be secured. The SCA security supplement identifies this, and delegates the responsibility of securing inter-object communication to the *Domain Manager* and the *Application Factory*. However, no specific guidelines are given and it is assumed the middleware will provide some form of security in this aspect.

Effect on CR Security

In CR architectures such as our architecture, different types of critical control messages are exchanged between the various components. It has to be ensured that these messages are not modified, repeated or lost by the transport mechanism due to any attack on the system. If the transport mechanism is vulnerable, an attacker can modify it to make the system act maliciously. An example would be the attacker modifying the response messages from the PR to the PE, so that it would appear that the transmission request was always approved. The PE would then allow all requests form the SSR to reach the modem and this may cause interference to incumbent systems in an opportunistic spectrum access scenario. The SSR can transmit at a higher transmission power or at a frequency it should not transmit at, leading to interference and *DoS* for other nodes in the network. The policy enforcement mechanism of the system thus fails if this vulnerability is exploited, and the system is no longer policy-conformant.

5.2.3 Authentication and Access Control

Vulnerability

A distributed system allows for location transparency and allows objects across platforms (or/and networks) to have access to each other. Authors in [29] note that there are scenarios in which objects should not be able to talk to each other. No protocol exists intrinsically within CORBA or SCA to provide authentication or enforce the concept of privileged access [29]. Authentication is the process of verifying a claimant's claimed identity. Authentication is critical since the authentication process bootstraps the entire security system and all other components rely on the privileges verified during the authentication process [14]. Access control requires restricting access to resources to prevent their unauthorized use. The problem of unrestricted access has also been noted in the security supplement to the SCA [8]; expressing concern that the use of CORBA middleware creates a security concern in that

information flow between objects could become unrestricted. As noted by the authors in [14], the access control will only be as strong as the authentication, security context establishment and message protection functions for most access control systems.

Most middleware developers using CORBA or *real-time* CORBA utilize CORBA object references as an authorization token. This practice is described by the authors in [13] as implicit authorization, and is defined as "*Implicit authorization means that handing over an object reference to a client object does not only provide the necessary addressing and context information but also authorizes the client entity to access the respective object*". The problem with such implicit authorization is highlighted by the authors in [13] and they describe two attacks which can take place due to the use of implicit authorization. These are:

- Attack 1: An attacker may obtain CORBA object references illegally. This can be done by either eavesdropping on the communication medium, or through some conspiring third party
- Attack 2: An attacker may fabricate CORBA object references. Authors in [13] describe the possibility of creating IORs for an ORB by knowing the object addresses used, the IDL specification and the general structure of IORs that the ORB generates.

The aim of attacks on a system using implicit authorization is to illegitimately invoke an operation on an object. This invocation may affect the confidentiality of information in the object, or change its state and thus affect the integrity of the system. Note that use of encryption (such as SSL) in the communications link may not protect against such attacks since most CORBA server objects do not verify if the client is actually authorized to establish the connection, and do not authenticate the client [13]. It is also noted that once an attacker has one IOR, all other objects using the same CORBA Portable Object Adapter (POA) may be prone to illegal access. This is due to the fact that an object key is used in the IOR, and these keys usually differ only in the value of the object counter for two objects of a particular POA [13]. Thus, an attacker can illegitimately access more than one object of the system

after getting a single reference.

SCA prescribes access control by only allowing objects within the *Domain Manager* process space to use the *resolve()* and *list()* functions to restrict access to object references outside the *Domain Manager* process space [8]. However, once an object has the reference, there are no further authentication checks when a method is invoked using this reference. The SCA thus relies on implicit authorization. Authors in [29] note that apart from needing the IOR, the attacker would also need to have knowledge of the IDL to invoke operations, since an SCA compliant implementation uses *minimum CORBA* [10]. Minimum CORBA does not have support for *Dynamic Invocation Interface*, so the attacker would need to know the IDL to invoke a method. This does make an attack on SCA exploiting the implicit authorization much harder, but it is still possible. For a system not compliant to the SCA and using implicit authorization, this vulnerability is easier to exploit.

Effect on CR Security

Access control is an important requirement in all systems, especially CR systems. It has to be ensured that certain operations are only invoked by objects authorized to invoke them. Let us assume that the access control is not properly implemented, and a rogue object can get access to the transmission modem which it should not have access to. The rogue object can then invoke methods on the modem and cause it to transmit and cause interference. Also, a rogue object can make another legitimate object unavailable by bombarding it with function calls, causing *DoS* to other legitimate objects of the system.

Section 5.3 discusses possible countermeasures for the vulnerabilities described in this section.

5.3 Possible Countermeasures

5.3.1 Securing the Bootstrapping process

Countermeasures

The SCA provides guidelines for securing the radio bootstrap process and it requires a CORBA naming service. For a non SCA-compliant system, it has to be ensured that the bootstrapping process is secure. Most distributed systems today have many objects connecting with the network and disconnecting at the same time. It is therefore beneficial to use the naming service to provide information to an object about other available objects. Objects use the *bind()* method to register with the naming service once they have been initialized. A client object only needs to have the reference to the naming service, and it can then lookup the reference for the server object using the *resolve()* method. If we can ensure that there is a reliable and valid naming service being used, and the reference to it is available to all objects before they start, we can prevent most security problems during bootstrapping. We also need to ensure that the communication to the naming service is secure.

Implementation

In our implementation, we use a CORBA naming service implementation provided by MICO (omniORB also provides one) to help the objects find initial references to each other. The reference to the naming service is hardcoded into the objects and cannot be changed. This reference includes the address and the port at which the naming service is running, and is passed as an ORBInitRef argument in MICO and omniORB. The naming service is started before the system is started. Once an object is started, it binds itself with the naming service. It can be found out by the object wishing to contact it using the *resolve()* method and the object name. To ensure that the correct naming service is in use and that a rogue object cannot masquerade as another, we require that the objects connect to the naming

service over SSL and use the GSSUP authentication (described in Section 5.3.3).

5.3.2 Securing Inter-object Communications

Countermeasures

Most CORBA ORBs provide the option to change the transport mechanism being used for inter-object communication [60] [61] [59]. This can be done for improving security as well as performance. By default, most CORBA implementations use IIOP transport. It is known that the transport layer used by the ORB is a major cause of runtime performance degradation. As described by the authors in [29], most ORB vendors offer various optimizations for common transport mechanisms and scenarios. A common optimization is skipping the marshaling/de-marshaling step for objects within the same address space. Other optimization steps involve the use of pre-connecting or binding objects. Apart from optimizations for performance, there are other optimizations and modifications used for improving security in some ORBs (e.g. [63]). But since such optimizations are outside the CORBA standard and thus reduce portability, they are not possible in strictly SCA-complaint systems. We do not analyze the use of any *real-time CORBA* implementation, since *real-time CORBA* heavily depends on the operating system properties to provide timeliness of ORB calls [29], and thus cannot be considered a portable implementation.

We aim to provide communication security to our implementation using standardized transport and security protocols. We use SSL based solutions (provided by many COTS ORBs) over IIOP for improving security, whereas we use UDP and UNIX domain sockets to improve runtime performance. The use of UDP was intended for increasing performance; whereas UNIX domain sockets also provide a number of security benefits [64]. UNIX domain sockets provide confidentiality, integrity as well as authentication. However, they are an inter-process communication concept and can only be used if all the objects are in the same process space, and thus are not applicable to all distributed systems.

For providing transport layer security in CORBA, SSL is widely used. SSL resides above the TCP layer and below the IIOP layer, and thus from a CORBA perspective, SSL is just another transport layer below IIOP [9]. SSL provides integrity, confidentiality and authentication features over TCP. Typically, most CORBA implementations primarily use SSL for confidentiality and integrity, since the authentication feature requires SSL certificates which may not be available with every client. SSL can thus be used to provide confidentiality and integrity to the transport mechanism, and secure it from vulnerabilities described in Section 5.2.2.

Implementation

Both omniORB and MICO provide the option to use *pluggable* transport mechanisms. We varied the transport mechanisms and security options to measure the effect on performance. We changed the TCP transport to either UDP or UNIX domain sockets. For providing security, we used SSL-IIOP for both ORBs. We used SSL for only providing confidentiality and integrity as explained above, and did not use the client certificates for authentication. We did this since we believe a CR may be implemented on systems for which some client objects may not have certificates as described in [9].

5.3.3 Securing Authentication and Access Control

Countermeasures

As discussed in Section 5.2.3, access control relies on authentication. CORBA provides options for explicit authentication in its security specification as described in Section 2.5.2. One of the specifications in the SAS protocol is the Username Password GSS Mechanism (GSSUP). GSSUP is a Generic Security Service API (GSSAPI) mechanism to support the delivery of authentication secrets above a transport mechanism such that they may be applied by a target security system to authenticate clients at shared secret authentication

systems [9]. This is achieved using a username/password combination that has to be passed from the client to the server. The client may connect to the server object without these credentials using an IOR, but will not be able to invoke any methods. This is an explicit authentication mechanism that combats the implicit authorization problem described in Section 5.2.3. Further security can be provided by the use of CSIv2 attributes as described in [14]. These protocols can provide authentication as well as privilege information over the transport mechanism for fine-grained access control. We however use the simple GSSUP since it is robust enough for our implementation.

Implementation

We used the GSSUP authentication option present in MICO. MICO provides CSIv2 level 0 implementation. We tested for two scenarios, GSSUP over TCP and GSSUP over SSL. For GSSUP, the SSR object had to provide a username and a password while connecting to the PE object for authentication. Even if a rogue object manages to illegitimately connect to the PE object by obtaining its IOR, it will not be able to invoke any method on the PE object since it does not have the required credentials.

Chapter 6

Experimental Evaluation

6.1 Efficacy of the Attack

6.1.1 Experimental Setup

For our experiment, we used two identical laptops, (Dell Vostro 1510, Intel Core 2 Duo (1.8 GHz), 2GB RAM). The operating system was Fedora 9. The GNU Radio version used was 3.1. We used two USRP boards, both equipped with the BasicRX, BasicTX and RFX400 daughterboards [17].

We made some modifications to the GNU Radio code so that it would be similar to what we would expect to find in a commercial software product. In such a product, we would expect the software to be implemented completely in a compiled language, such as C or C++, due to performance and size constraints. Hence, we ignored the Python front-end to the system, and we moved the value of interest, the frequency, as a constant in the C code. We chose the *set_tx_freq* method in the *libusrp* library because it is the first (and only) function outside Python that deals with the transmission frequency. We modified this function to ignore the frequency it receives as a parameter from the Python front-end, and use a constant value

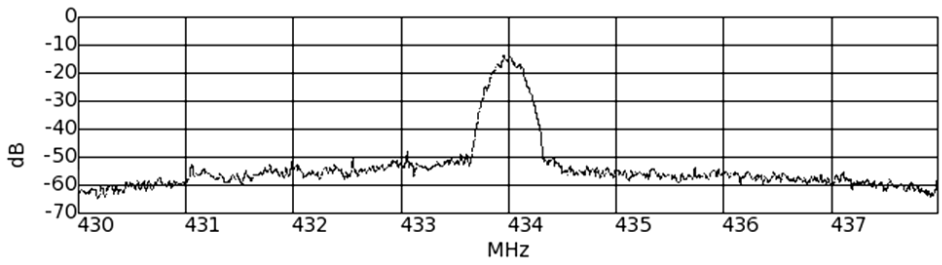

Figure 6.1: Spectral density showing unmodified carrier frequency (434MHz)

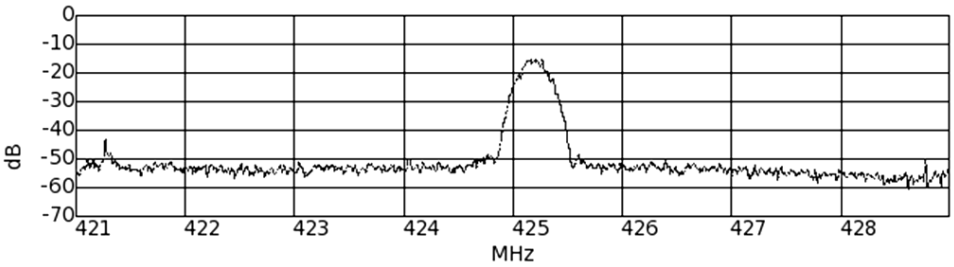

Figure 6.2: Spectral density showing carrier frequency shifted down (425.2MHz)

instead — which we changed with our attack.

6.1.2 Experimental Results

To check that our attack has succeeded, we completed transmissions from one USRP to the other, using the *benchmark_tx.py* and *benchmark_rx.py* scripts. We noticed the transmission changes were accurate over a range of about 24 MHz (NCO ratios of about 0.4, considering the unchanged ratio was 0.125 or 4 MHz). Figures 6.1, 6.2, and 6.3 show the received

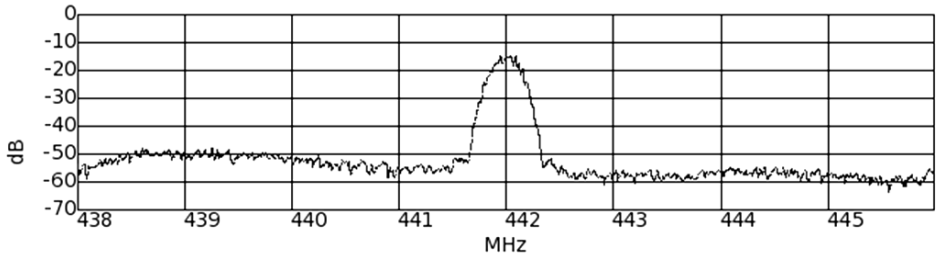

Figure 6.3: Spectral density showing carrier frequency shifted up (442 MHz)

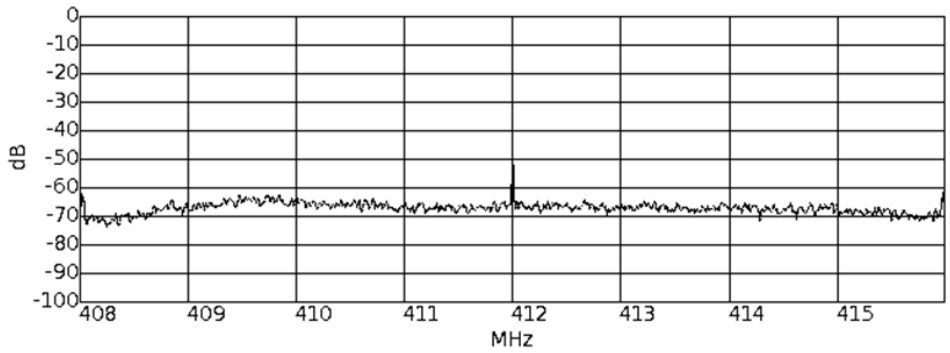

Figure 6.4: Output showing no transmission since frequency had been changed by the attack, and *RF Firewall* disallowed transmission

signal's spectral density (as observed on the receiving USRP using the *usrp_fft.py* script), without frequency modification, with frequency shifted down, and with frequency shifted up, respectively. One can clearly observe that the modifications made by the attack resulted in the change of the radio's carrier frequency as expected.

6.2 Effect of using the *RF Firewall*

Figure 6.4 shows the output of *usrp_fft.py* at the receiving USRP board when the command to send the tampered frequency is executed in a system with the *RF firewall* active. It can be seen that the transmission has been blocked.

Figure 6.5 shows the output from the *RF Firewall* console when control signals for the disallowed frequencies were sent across the USB connection by the SDR software. It also includes minimal analysis of memory and actual transmission time (the time between the determination of a disallowed frequency and killing the process accessing the radio). The table shown in the output is a dump of Python's private heap, which shows the memory used by the *RF Firewall* process at runtime. The algorithm that extracts the actual raw values from the control signals uses a few simple operations (such as bit shifts) to calculate the value. This added no noticeable time delay to the transmission. For real world applications, we assume

```
danny@ubuntu:~$ sudo !!
sudo ./front.py
*********************************
* RF Firewall - Policy Enforcer *
*********************************

Is there a frequency or range of frequencies (MHz) you would like to allow?
420
First transaction recorded
Begining USB monitoring...
Allowed coarse frequency accepted
The frequency has not been fine tuned
Frequency not allowed, terminating process
Signal Killed

Partition of a set of 24 objects. Total size = 2560 bytes.
 Index  Count   %     Size   % Cumulative  % Kind (class / dict of class)
     0      2   8      912  36       912  36 types.FrameType
     1     12  50      792  31      1704  67 str
     2      3  12      472  18      2176  85 list
     3      2   8      240   9      2416  94 function
     4      4  17       96   4      2512  98 int
     5      1   4       48   2      2560 100 datetime.datetime

Total actual transmission time:  0:00:00.106634
```

Figure 6.5: Output at the monitoring console when the attack was performed on the SDR

that the PE is a monitoring device, separate from the rest of the system. It would therefore incur very little performance overhead in simply monitoring the transmission parameters. Depending on the type of system and policy enforcement situation, some noticeable performance overhead may occur when checking a large number of parameters against a large number of policies.

6.3 Effect of using the Cache-based PE

6.3.1 Experimental Setup

For our experiment, we used a single computer to test our system, thus all objects of the policy-based radio were on a single node. We used a system with an Intel Core 2 Duo Processor (2.53 GHz) with 6 GB of RAM running Ubuntu 10.4 (64-bit version). The results were obtained for at least 10000 requests generated by the SSR for every case.

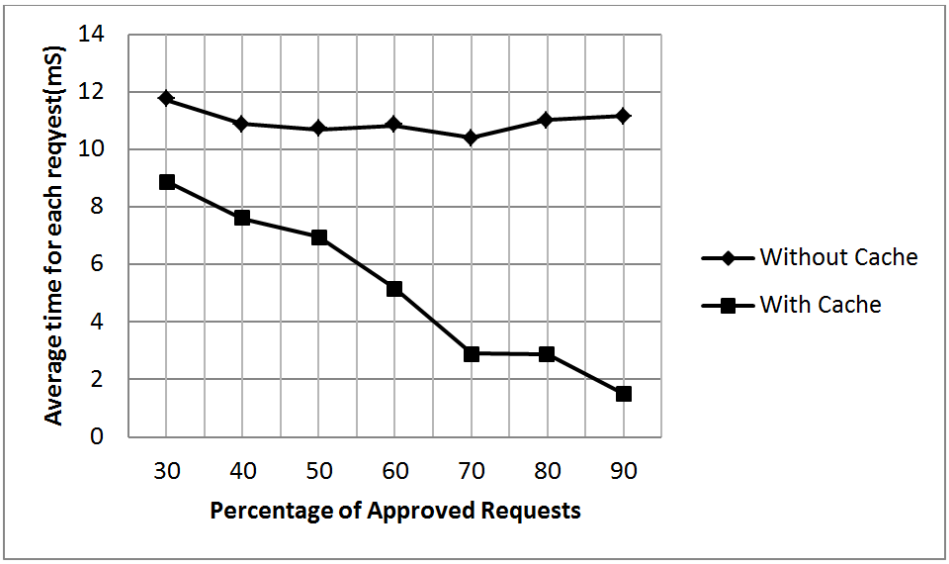

Figure 6.6: Performance improvements due to the use of cache

6.3.2 Experimental Results

As described in the implementation in Section 4.3.5, a cache-based *MITM* approach was implemented for our system. We had the option to turn the cache ON or OFF. For the OFF case, all requests were forwarded by the PE to the PR. The average request time for the OFF case thus reflects the time taken by the PR to service requests for this case. As Figure 6.6 shows, the average request time remains approximately the same for all percentages of approved requests without a cache. With the cache in use (the ON case), we cache the transmission requests that were approved (as described in Section 4.3.5). As the percentage of approved requests increases, more requests are found in the cache and the PE can make a decision for the request itself without referring to the PR. This, combined with the fact that the PE lookup time in the cache is a very small fraction of the service time required by the PR, provides a major performance boost as can be seen in Figure 6.6. It can be seen that the PR decision time would be more than a cache lookup time (approximately less than a millisecond) for most implementations, e.g. the reasoning time is approximately 10 milliseconds for the highly efficient BRESAP. It is expected that the SSR would be

intelligent enough to create transmission requests with higher chances of approval. Thus we can conclude that for a policy-based radio system with a cognitive SSR and a PR, the use of a cache in the PE is truly beneficial.

6.4 Incorporating Security Mechanisms in the Middleware

Our experiments provide quantitative data regarding the degradation in performance that may result due to the use of different security and transport mechanisms in the middleware.

6.4.1 Experimental Setup

For this experiment, we used a single computer to test our system, thus all objects were on a single node. We did this so that we can test UNIX domain sockets as well as other transport mechanisms easily and avoid network uncertainties in our measurements. We used a system with an Intel Core 2 Duo Processor (2.53 GHz) with 6 GB of RAM running Ubuntu 10.4 (64-bit version). The results were obtained for at least 10000 requests generated by the SSR for every case.

6.4.2 Experimental Results

Use of Different Transport Mechanisms

As explained in Section 5.1, we only change the transport layer for the ORB between the PE and the SSR. IIOP is always used between the PR and PE. The results for the various transport mechanisms are shown in Figure 6.7, and analyzed below.

- **TCP**: As can be seen in Figure 6.7, MICO provides a slightly faster implementation

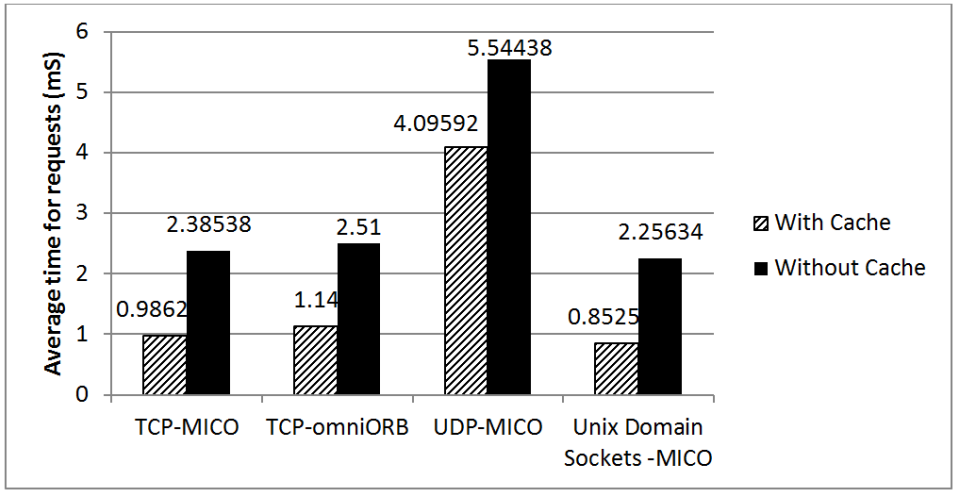

Figure 6.7: Use of different transport mechanisms and ORBs

for TCP than omniORB. This could be explained by the fact that even for a loopback interface, the TCP performance depends on the TCP implementation in use. MICO and omniORB use different TCP implementations and this may explain the difference in performance for these two cases.

- **UDP**: Another anomaly is the case of UDP against TCP for MICO. UDP would have been faster than TCP in the normal scenario, had there been a *3-way handshake* [65] for every request made by the SSR to the PE. However, for most COTS ORBs, the *3-way handshake* is performed only once — before the first function call. The socket is then open until one of the objects disconnects from the other. Thus, the delay due to connection setup for TCP is not present here. Also, another source of performance delay could be a result of measures taken by the ORB implementation to provide reliability over the unreliable UDP. MICO documentation [60] notes that UDP can drop, duplicate, or reorder requests and the authors recommend using UDP only on networks where dropped, duplicated and unordered requests are not likely.

  Note that the performance for the cache case deteriorates more than the case without cache. This is due to the fact that for the cache case, majority of the messages (approximately 70%) are passed between the PE and the SSR over UDP. For the cache

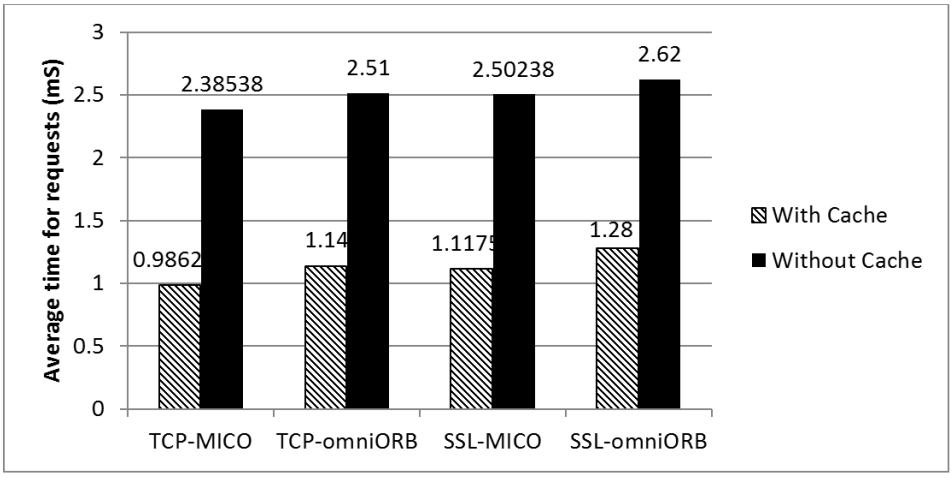

Figure 6.8: Use of security mechanisms with the transport layer

OFF case, the TCP messages between the PE and the PR increase and have an impact on the measurements.

The performance also depends on the UDP implementation being used, and it is possible that another ORB could provide different results than MICO. However, since most CORBA implementations use a form of "persistent" IIOP connection, GIOP over UDP should be used only when it can provide clear advantages over IIOP.

- **UNIX domain sockets**: UNIX domain sockets provide better performance than TCP/UDP when the objects are located in the same process space. Since this may not always be the case for a distributed system, UNIX domain sockets cannot be used as a universal solution. However, as described in [64], the security benefits of UNIX domain sockets make them a good choice for connecting objects in the same process space.

Using Secure Transport Mechanisms

We test the performance of SSL-IIOP for both MICO and omniORB as shown in Figure 6.8. For the same ORB, it can be seen that SSL is slightly slower than TCP. It is known that apart from the connection setup delay due to the extended SSL handshake, SSL performance

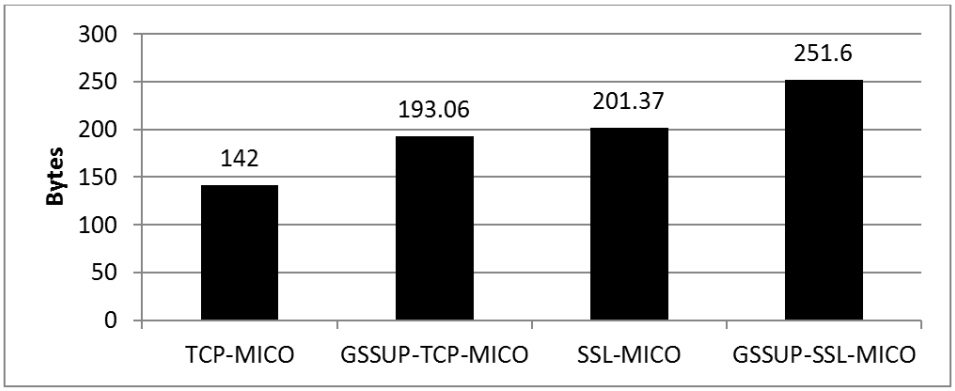

Figure 6.9: Packet size for corresponding transport mechanisms

is usually close to TCP. Since the connection setup for SSL also occurs only once during the startup similar to the connection setup for TCP as described in Section 6.4.2, SSL performance for our system is close to the TCP performance. The loss in performance over TCP could be attributed to the increase in packet size (see Figure 6.9) on using SSL, as well as the extra processing at the endpoints for encryption and decryption [33]. Since both MICO and omniORB use openSSL to provide for SSL support, the poor performance of omniORB SSL as compared to MICO SSL can be attributed to its underlying TCP implementation as described previously.

Use of Object Authentication and Secure Transport

As can be seen in Figure 6.10, the use of GSSUP degrades performance for both the TCP and SSL cases. GSSUP forces the server (PE) to authenticate the client (SSR) for every function call based on the username/password combination passed on as part of the SAS section in the GIOP packets. As can be seen from Figure 6.9, the passing of authentication details for GSSUP also increases the size of the packets, thus increasing the transmission delay.

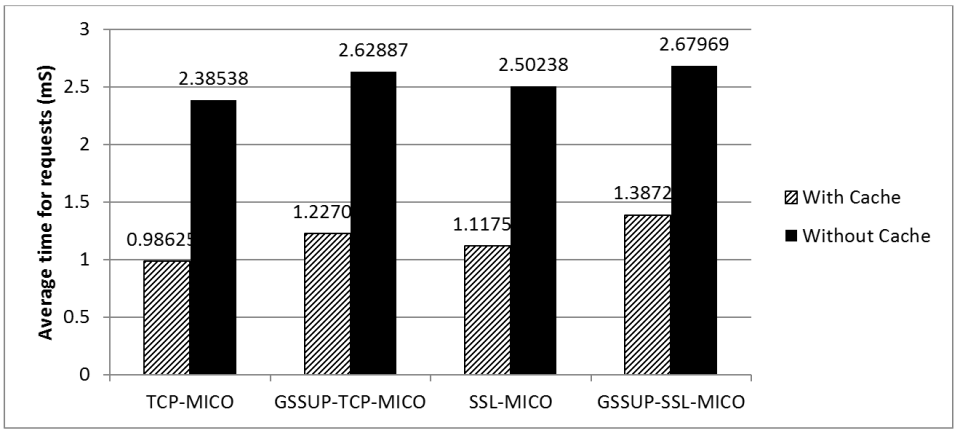

Figure 6.10: Use of authentication along with the use of security mechanisms

6.4.3 Trade-offs between Security and Performance

There is always a trade-off between making the security architecture unobtrusive and providing effective security enforcement for a middleware [14]. ORB vendors try to balance this in different ways in their ORB implementations. The application developer has to identify which features to use from the many available security mechanisms provided by an ORB. For a real-time system such as a CR, the application developer has to make calculated assumptions about the security levels required, and provide certain countermeasures if the ORB does not provide enough security against a particular vulnerability. It can thus be seen that securing a distributed CR system does not have a universal solution, particularly since different ORBs provide different security features and multiple ORBs may be used.

Any approach to counter the bootstrapping problem hardly has an effect on runtime performance of the system since the bootstrapping once performed does not repeat until the system is restarted. For our implementation, since the objects only communicate with the naming service once to obtain the reference to another object, and there are no messages passed between the naming service and the objects apart from during the start-up, there is no effect in terms of performance during normal radio operation. There will be a slight delay due to the use of SSL and GSSUP for connecting to the CORBA naming service at startup, but it does not affect normal system operation. Countermeasures for the other

vulnerabilities of inter-object communications and access control and authorization however do affect the performance.

For securing communications, we could either depend on the inherent properties of UNIX domain sockets, or use SSL-IIOP. The use of UNIX domain sockets is ideal in cases where all objects share the same address space. However, for typical distributed systems, SSL-IIOP is the solution used. We observe that SSL does degrade performance slightly, but it provides confidentiality and integrity which are a necessity for transport layer security. SSL or a lightweight version of SSL should therefore be used in a distributed CR system.

For access control and authentication, the use of GSSUP demonstrated a greater performance loss. It can be seen that the system using GSSUP and SSL is the most secure in terms of middleware transport security as it provides confidentiality, integrity as well as authentication. However, the performance loss is not minor and cannot be ignored.

It is seen from our results that the security of a distributed CR system depends on various factors and that security solutions contribute to performance degradation. However, the magnitude of performance degradation can be reduced by the use of a cleverly selected combination of solutions. The application developer has to understand the trade-offs and choose the transport security and authentication options wisely, as different combinations of these can show a large variance in performance.

Chapter 7

Conclusion and Future Work

In this thesis, we considered several important aspects of securing CRs, with particular focus on the Policy Enforcer component. We described the implementation of an attack on SDRs using advanced features of processors commonly used in them. The attack demonstrates that the emergence of SDRs/CRs raises new security threats that have not been considered in the context of legacy wireless devices (i.e., conventional hardware-based wireless devices). The attack underscores the necessity of the Policy Enforcer in enforcing policy conforming transmission behavior of cognitive radios. We further evince the need for security for advanced SDRs such as policy-based radios that may be implemented as distributed systems by highlighting security vulnerabilities in them.

An in-depth analysis of the Policy Enforcer has been performed in our research. We enumerated the requirements for an "ideal" PE, and also described the design and implementation of two possible implementations of the PE. Our implementations show that the PE can ensure policy-conformance in policy-based radios, even when they are implemented as distributed systems. However, extra security measures do need to be provided for most COTS middleware to ensure complete security for a distributed SDR. This comes at a performance cost. We described the trade-offs between performance and security that exist for such systems, and showed that proper choices by the developer can provide the best performance for a

given security level.

Being a nascent research area, there are several directions existing for future work. The PE component can be analyzed for simpler SDRs, wherein a PR component may not be present. Such systems will use a complex PE enforcing a given set of policies, much like traditional radios. We have provided a framework for a distributed policy-based radio that uses CORBA. As highlighted previously, any component of our system can be replaced with another component that follows the CORBA IDL. We would like to integrate a Cognitive Engine in place of the dummy SSR used in our system, to create an actual prototype policy-based CR. Using a Cognitive Engine from the Cognitive Radio Open Source System (CROSS) [66] would be a good way to start in this direction. Also, other implementations of the PR and the PE can be readily implemented in our system, and their performances can be analyzed. We have utilized open source middleware, which can also be replaced with proprietary ones to provide other features. Future implementations can thus include advanced cognitive, reasoning and enforcing capabilities and security features, based on our prototype framework.

Bibliography

[1] B. A. Fette, *Cognitive radio technology.* Academic Press/Elsevier, 2009.

[2] SDRForum, "SDRF Cognitive Radio Definitions (SDRF-06-R-0011-V1.0.0)," http://www.sdrforum.org/pages/documentLibrary/documents/SDRF-06-R-0011-V1_0_0.pdf.

[3] P. Koch and R. Prasad, "The universal handset," *Spectrum, IEEE*, vol. 46, no. 4, pp. 36 –41, Apr. 2009.

[4] DARPA, "XG Program," http://www.darpa.mil/sto/programs/xg/.

[5] F. Perich, "Policy-based Network Management for NeXt Generation Spectrum Access Control," in *New Frontiers in Dynamic Spectrum Access Networks, 2007. DySPAN 2007. 2nd IEEE International Symposium on*, Apr. 2007, pp. 496–506.

[6] G. Denker, D. Elenius, R. Senanayake, M.-O. Stehr, and D. Wilkins, "A Policy Engine for Spectrum Sharing," in *New Frontiers in Dynamic Spectrum Access Networks, 2007. DySPAN 2007. 2nd IEEE International Symposium on*, Apr. 2007, pp. 55–65.

[7] XG Working Group, "The XG vision. Request for comments. Version 2.0," BBN Technologies, Tech. Rep., 2005, www.ir.bbn.com/~ramanath/pdf/rfc_vision.pdf.

[8] JPEO JTRS, "Software Communications Architecture Specification V2.2.2," http://sca.jpeojtrs.mil/.

[9] OMG, "The OMG's CORBA website," http://www.corba.org/.

[10] OMG, "The OMG Minimum CORBA specification," http://www.omg.org/technology/documents/formal/minimum_CORBA.htm.

[11] P. J. Balister, M. Robert, and J. H. Reed, "Impact of the use of CORBA for Inter-Component Communication in SCA Based Radio," in *Proceeding of the SDR 06 Technical Conference and Product Exposition, Orlando, Florida*, Nov. 2006.

[12] F. Casalino, G. Middioni, and D. Paniscotti, "Experience report on the use of CORBA as the sole middleware solution in SCA-based SDR environments," in *Proceeding of the SDR 08 Technical Conference and Product Exposition, Washington, D.C.*, Oct. 2008.

[13] C. Becker, S. Staamann, and R. Salomon, "Security Analysis of the Utilization of Corba Object References as Authorization Tokens," in *Object and Component-Oriented Real-Time Distributed Computing, 2007. ISORC '07. 10th IEEE International Symposium on*, May 2007, pp. 196 –203.

[14] U. Lang and R. Schreiner, *Developing Secure Distributed Systems with CORBA*. Artech House, 2002.

[15] SDRForum, "High-level SDR Security Requirements. SDRForum Approved Document . SDRF-06-S-0002-V1.0.0," http://www.sdrforum.org/pages/documentLibrary/documents/SDRF-06-S-0002-V1_0_0_High_Level_SDR_Security.pdf.

[16] "GNURadio and its source code," http://www.gnu.org/software/gnuradio/.

[17] "Universal Software Radio Peripheral documentation," http://gnuradio.org/trac/wiki/USRP.

[18] B. Bahrak, A. Deshpande, M. Whitaker, and J.-M. Park, "BRESAP: A Policy Reasoner for Processing Spectrum Access Policies Represented by Binary Decision Diagrams," in *New Frontiers in Dynamic Spectrum, 2010 IEEE Symposium on*, Apr. 2010, pp. 1–12.

[19] M. McHenry, E. Livsics, T. Nguyen, and N. Majumdar, "XG dynamic spectrum access field test results [topics in radio communications]," *Communications Magazine, IEEE*, vol. 45, no. 6, pp. 51 –57, Jun. 2007.

[20] P. Marshall and P. Kolodzy, "A Potential Alliance for World-Wide Dynamic Spectrum Access," in *New Frontiers in Dynamic Spectrum, 2010 IEEE Symposium on*, Apr. 2010, pp. 1 –4.

[21] J. Mitola, "Cognitive Radio: An Integrated Agent Architecture for Software Defined Radio," Ph.D. dissertation, Royal Institute of Technology, Sweden, 2000.

[22] B. Freyens, M. Loney, and M. Poole, "Wireless Regulations and Dynamic Spectrum Access in Australia," in *New Frontiers in Dynamic Spectrum, 2010 IEEE Symposium on*, Apr. 2010, pp. 1 –12.

[23] J. M. Peha and S. Panichpapiboon, "Real-time Secondary markets for spectrum," in *Telecommunications Policy*, vol. 28, no. 7-8, Sep. 2004, pp. 603–618.

[24] K. Woyach, P. Pyapali, and A. Sahai, "Can We Incentivize Sensing in a Light-Handed Way?" in *New Frontiers in Dynamic Spectrum, 2010 IEEE Symposium on*, Apr. 2010, pp. 1 –12.

[25] M. Buddhikot, "Cognitive Radio, DSA and Self-X: Towards Next Transformation in Cellular Networks (extended abstract)," in *New Frontiers in Dynamic Spectrum, 2010 IEEE Symposium on*, Apr. 2010, pp. 1 –5.

[26] "FCC September Commission Meeting Presentation," Sep. 2009, http://hraunfoss.fcc.gov/edocs_public/attachmatch/DOC-293742A1.pdf.

[27] D. Wilkins, G. Denker, M.-O. Stehr, D. Elenius, R. Senanayake, and C. Talcott, "Policy-based Cognitive Radios," *Wireless Communications, IEEE*, vol. 14, no. 4, pp. 41 –46, Aug. 2007.

[28] XG Working Group, "Next Generation (XG) Architecture and Protocol Development (XAP)," Tech. Rep., Aug. 2005, aFRL-IF-RS-TR-2005-2B1.

[29] J. Bard and V. J. Kovarik, *Software Defined Radio: The Software Communications Architecture*. Wiley series in software radio. Chichester, England: John Wiley, 2007.

[30] R. Chen, J.-M. Park, and J. Reed, "Defense against Primary User Emulation Attacks in Cognitive Radio Networks," *Selected Areas in Communications, IEEE Journal on*, vol. 26, no. 1, pp. 25–37, Jan. 2008.

[31] T. Clancy and N. Goergen, "Security in Cognitive Radio Networks: Threats and Mitigation," in *Cognitive Radio Oriented Wireless Networks and Communications, 2008. CrownCom 2008. 3rd International Conference on*, May 2008, pp. 1–8.

[32] T. X. Brown and A. Sethi, "Potential Cognitive Radio Denial-of-Service Vulnerailities and Protection Countermeasures: A Multi-dimensional Analysis and Assessment," in *Cognitive Radio Oriented Wireless Networks and Communications, 2007. CrownCom 2007. 2nd International Conference on*, Aug. 2007, pp. 456–464.

[33] IETF, "TLS Specification. IETF RFC," http://www.ietf.org/rfc/rfc2246.txt.

[34] A. Gad and F. Digham, "DSA: The Path Forward," in *New Frontiers in Dynamic Spectrum, 2010 IEEE Symposium on*, Apr. 2010, pp. 1 –10.

[35] S. Sodagari, A. Attar, and S. Bilen, "Strategies to Achieve Truthful Spectrum Auctions for Cognitive Radio Networks Based on Mechanism Design," in *New Frontiers in Dynamic Spectrum, 2010 IEEE Symposium on*, Apr. 2010, pp. 1 –6.

[36] F. Perich and M. McHenry, "Policy-based spectrum access control for dynamic spectrum access network radios," *Web Semant.*, vol. 7, no. 1, pp. 21–27, 2009.

[37] J. Moskal and M. M. Kokar, "Interfacing a reasoner with an SDR: A platform and domain independent approach," in *Proceeding of the SDR 06 Technical Conference and Product Exposition, Orlando, Florida*, Nov. 2006.

[38] T. W. Rondeau, B. Le, D. Maldonado, D. Scaperoth, and C. W. Bostian, "Cognitive Radio Formulation and Implementation," in *Cognitive Radio Oriented Wireless Networks and Communications, 2006. 1st International Conference on*, Jun. 2006, pp. 1 –10.

[39] C. Gonzalez and J. Reed, "Validation and verification of modular software for Software-defined Radios," in *Proceeding of the SDR 07 Technical Conference and Product Exposition,Denver, Colorado*, Nov. 2007.

[40] M. Kurdziel, J. Beane, and J. Fitton, "An SCA security supplement compliant radio architecture," in *Military Communications Conference, 2005. MILCOM 2005. IEEE*, Oct. 2005, pp. 2244 – 2250 Vol. 4.

[41] J. Davidson, "On the architecture of secure software defined radios," in *Military Communications Conference, 2008. MILCOM 2008. IEEE*, Nov. 2008, pp. 1 –7.

[42] H. Uchikawa, K. Umebayashi, and R. Kohn, "Secure download system based on software defined radio composed of FPGAs," in *Personal, Indoor and Mobile Radio Communications, 2002. The 13th IEEE International Symposium on*, vol. 1, Sep. 2002, pp. 437 – 441 vol.1.

[43] SDRForum, "Security considerations for operational software for software defined radio devices in a commercial wireless domain (DL-SIN SDRF-04-P- 0010-V1.0.0)," http://groups.sdrforum.org/download.php?sid=772.

[44] T. Tsou, P. Ballister, and J. Reed, "Latency profiling for SCA Software Radio," in *Proceeding of the SDR 07 Technical Conference and Product Exposition,Denver, Colorado*, Nov. 2007.

[45] Wireless@VT, "Open-source SCA Implementation-Embedded," http://ossie.wireless.vt.edu/.

[46] Rice University, "Wireless Open-Access Research Platform (WARP)," http://warp.rice.edu/.

[47] H. Chang and M. J. Atallah, "Protecting software code by guards," in *DRM '01: Revised Papers from the ACM CCS-8 Workshop on Security and Privacy in Digital Rights Management*. London, UK: Springer-Verlag, 2002, pp. 160–175.

[48] B. Horne, L. R. Matheson, C. Sheehan, and R. E. Tarjan, "Dynamic self-checking techniques for improved tamper resistance," in *DRM '01: Revised Papers from the ACM CCS-8 Workshop on Security and Privacy in Digital Rights Management*. London, UK: Springer-Verlag, 2002, pp. 141–159.

[49] G. Wurster, P. van Oorschot, and A. Somayaji, "A generic attack on checksumming-based software tamper resistance," in *Security and Privacy, 2005 IEEE Symposium on*, May 2005, pp. 127 – 138.

[50] A. Vasudevan and R. Yerraballi, "Stealth breakpoints," in *Computer Security Applications Conference, 21st Annual*, Dec. 2005, pp. 10 pp. –392.

[51] *Analog Devices AD9862 CODEC Datasheet*, 2002.

[52] Intel Corporation, *Intel Software Developer's Manual, Volume 3B*, Jul. 2008, www.intel.com.

[53] ARM, *ARM Documentation*, http://infocenter.arm.com/help/index.jsp.

[54] Intel Corporation, *Intel Software Developer's Manual, Volume 1*, 1997, www.intel.com.

[55] "Ptrace API man pages for Linux," http://linux.die.net/man/2/ptrace.

[56] D. Murotake and A. Martin, "Updated system threat analysis for high assurance software defined radios," in *Proceeding of the SDR 05 Technical Conference and Product Exposition, Orange County, California*, Nov. 2005.

[57] CISCO Systems, "Evolution of the firewall industry," http://www.cisco.com/univercd/cc/td/doc/product/iaabu/centri4/user/scf4ch3.htm.

[58] W. Weber, "Firewall basics," in *Telecommunications in Modern Satellite, Cable and Broadcasting Services, 1999. 4th International Conference on*, vol. 1, 1999, pp. 300 –305 vol.1.

[59] "Java IDL ORB documentation," http://java.sun.com/developer/onlineTraining/corba/corba.html.

[60] "MICO CORBA," http://www.mico.org/.

[61] D. Grisby, S. Lo, and D. Riddoch, "The omniORB version 4.1 User's Guide," http://omniorb.sourceforge.net/omni41/omniORB.pdf.

[62] K. R. Puder, Arno and F. Pilhofer, *Distributed Systems Architecture: A Middleware Approach*. Elsevier, 2006.

[63] "ORBexpress middleware," http://www.ois.com/Products/communications-middleware.html.

[64] T. Stover, "Demystifying UNIX Domain Sockets," 2006, http://www.wsinnovations.com/softeng/articles/uds.html.

[65] IETF, "TCP specification. IETF RFC," http://www.ietf.org/rfc/rfc793.txt.

[66] Wireless@VT, "Cognitive Radio Open Source System (CROSS)," http://cornet.wireless.vt.edu/trac/wiki/Cross.