Research Article

Strengthening MT6D Defenses with LXC-Based Honeypot Capabilities

Dileep Basam, J. Scot Ransbottom, Randy Marchany, and Joseph G. Tront

Bradley Department of Electrical and Computer Engineering, Virginia Tech, Blacksburg, VA 24061, USA

Correspondence should be addressed to Dileep Basam; dileepba@vt.edu

Received 5 November 2015; Revised 8 February 2016; Accepted 6 March 2016

Academic Editor: Elias P. Duarte

Moving Target IPv6 Defense (MT6D) imparts radio-frequency hopping behavior to IPv6 networks by having participating nodes periodically hop onto new addresses while giving up old addresses. Our previous research efforts implemented a solution to identify and acquire these old addresses that are being discarded by MT6D hosts on a local network besides being able to monitor and visualize the incoming traffic on these addresses. This was essentially equivalent to forming a darknet out of the discarded MT6D addresses, but the solution presented in the previous research effort did not include database integration for it to scale and be extended. This paper presents a solution with a new architecture that not only extends the previous solution in terms of automation and database integration but also demonstrates the ability to deploy a honeypot on a virtual LXC (Linux Container) on-demand based on any interesting traffic pattern observed on a discarded address. The proposed architecture also allows an MT6D host to query the solution database for network activity on its relinquished addresses as a JavaScript Object Notation (JSON) object. This allows an MT6D host to identify suspicious activity on its discarded addresses and strengthen the MT6D scheme parameters accordingly. We have built a proof-of-concept for the proposed solution and analyzed the solution's feasibility and scalability.

1. Introduction

Nodes participating in MT6D periodically relinquish IP addresses and hop on to new addresses. This address-hopping trait of MT6D improves the security and privacy of IPv6 networks; however, there is no feedback mechanism in the current implementation; that is, a node engaged in a MT6D conversation has no means of realizing if there is an attacker uncovering its MT6D addresses and trailing the MT6D node along its address hops.

(i) Challenge. In this work, we propose a mechanism to use relinquished MT6D addresses in order to discover any anomalies and gather intelligence on attacker methods. We also build a proof-of-concept solution to impart darknet and honeypot capabilities to MT6D scheme.

(ii) Proposal. Our approach to solve this problem is to design a solution that identifies and acquires the addresses discarded by the MT6D hosts and enumerating incoming traffic on these relinquished addresses in addition to providing the ability to deploy a virtual container-based honeypot configured with a specific discarded address upon detecting a suspicious traffic pattern. Our previous effort [1] included building a Central Node (CN) that passively listens to local link traffic in promiscuous mode, identifying and acquiring discarded addresses by MT6D nodes and performing traffic enumeration on these addresses. The CN uses typical IPv6 neighbor discovery protocol (NDP) messages like Neighbor Solicit (NS) and Multicast Listener Discovery (MLD) messages to reconstruct who is relinquishing which addresses and when to acquire them.

The solution's ability to deploy a honeypot tied to a discarded MT6D address of interest can potentially take up the conversation forward with the attacker to gather intelligence on attacker methods besides collecting attack traffic samples for further analysis. The solution also allows an MT6D node to query the database for incoming traffic on its discarded addresses, thus giving the MT6D node the ability to analyze the activity to identify malicious traffic or an attacker persistently trailing on its discarded addresses, to change

the scheme parameters, for example, move to a stronger secret-key or a faster hopping interval to evade the attacker.

(iii) Threat Model and Assumptions. We assume a threat model involving an attacker with infinite resources (compute cycles and time), with an ability to brute-force the scheme parameters, for example, the secret key involved in address computation (hashing scheme), to discover the MT6D addresses. But the attacker validating these uncovered MT6D addresses will generate some traffic, leaving a trail of his/her activity on these discarded MT6D addresses.

(iv) Goals. In this paper, we seek to impart honeypot capabilities to Moving Target IPv6 Defense (MT6D) in order to provide a feedback mechanism for evaluating the strengths of MT6D scheme by analyzing the activity on discarded MT6D addresses. Among the goals of this effort, the first goal is to implement the database integration into CN along with developing a way to build web server to offer real-time visualization of the local MT6D addresses that are being relinquished and incoming traffic on these discarded addresses. Our next goal is to implement a capability of deploying honeypot service on a virtual container on-demand bound to a discarded MT6D address of interest with minimal interruption observable to an attacker. Lastly, we want to analyze the feasibility and scalability of the proposed solution by analyzing the CPU and memory overhead trends while scaling to multiple honeypot containers.

The paper is organized as follows. First, we provide background on Moving Target IPv6 Defense (MT6D), Linux Container (LXC), Honeypots, Dionaea, and various components used in building the solution in Section 2. Related work on virtual honeypot based solutions and adaptive/dynamic honeypot architectures is discussed in Section 3. Section 4 explains our idea and high level approach. Section 5 describes the solution testbed and implementation details. In Section 6, we present our results, visualizations, and our observations. Sections 7 and 8 present future work and conclusion.

2. Background

In this section, we present background, starting with IPv6 followed by Moving Target IPv6 Defense (MT6D), Dionaea, LXC, and other components used in building the solution.

2.1. IPv6. On September 24, 2015, the American Registry for Internet Numbers (ARIN) announced that it allocated the final IPv4 address blocks in its free pool and the IPv4 address space is completely exhausted. The IPv6 protocol by design overcomes the address exhaustion limitation with its address length of 128 bits. This new address size allows for 2^{128} possible addresses, approximately 7.92×10^{28} addresses for every possible IPv4 address.

2.2. Moving Target IPv6 Defense (MT6D). The MT6D [2, 3] involves nodes that change network addresses while maintaining active sessions. Leveraging the large address space in IPv6, the MT6D protocol changes IP addresses for communicating hosts in synchronization. This achieves an effect similar to frequency hopping in radio networks, for an attacker passively listening to the conversation of two MT6D hosts will see multiple hosts communicating with each other rather than a pair of hosts. MT6D achieves synchronization between the involved nodes by having both ends of a communication pair compute both their own IP address and their remote-peer's IP address during the particular time window "t". The scheme involves computation of the interface-identifier part of the address, that is, IID$'$ extracted from first 64 bits from a hash digest (H) of concatenated string made of a symmetric Secret-Key (SK), initial IID, and current time window "t":

$$\text{IID}' = H\left[\text{IID\_initial} \parallel \text{SK} \parallel t\right]_{0 \to 63}. \quad (1)$$

MT6D mechanism offers privacy and anonymity by hiding the original network layer addresses of involved IPv6 nodes with dynamically generated addresses besides protecting against intrusion-based network attacks. MT6D protects against address tracking and traffic correlation, thus helping hosts keep their network identities private while conducting sensitive communications.

2.3. Dionaea and DionaeaFR. Dionaea is a low-interaction honeypot offering IPv6 support with an ability to capture malware in addition to logging suspicious traffic. Dionaea offers flexibility to configure services and interfaces to listen on. Dionaea uses Sqlite as the backend database allowing custom queries to be made to extract interesting information from logs stored in the database [4].

DionaeaFR is an open-source library that offers front-end visualization of Dionaea's logs that are stored in the Sqlite database. DionaeaFR's web server is implemented in Python and Django framework. This front-end console allows retrieving statistics of traffic hitting the honeypot in addition to offering a way to download attack traffic samples, for example, malware [5].

2.4. LXC. LXC is a light-weight virtualization technology that relies on isolated user spaces to create virtual containers that share same kernel. LXC uses Linux kernel features like namespaces, Chroots, CGroups, and so forth, to isolate the container processes. LXC differs from the concept of standard virtual machines (VM) by sharing the same kernel and underlying hardware which obviates the need for a hypervisor [6].

2.5. Other Components Used in Building the Solution

2.5.1. Scapy, Pcapy, Impacket, Pyroute2, and Wireshark. Scapy is a Python module for packet crafting and manipulation tool for computer networks [7]. Pcapy and Impacket are python libraries for capturing and decoding raw network traffic. Pyroute2 is a network configuration library for binding IP addresses to a Linux host. Wireshark is a popular packet capture and analysis tool [8].

2.5.2. MongoDB and PyMongo. MongoDB is a NoSQL database that stores each record in the database as a JSON object and PyMongo is a Python library that acts as an application programming interface (API) to interact with MongoDB from Python code.

2.5.3. D3.js. D3.js is a JavaScript library that leverages HTML, SVG, and CSS to produce complex visualizations on a web browser [9].

2.5.4. Dstat. Dstat is a free tool that combines vmstat, iostat, netstat, and ifstat to view system resources in real-time. The tool is used in this work to collect real-time CPU (usr/sys) and free-memory (RAM) statistics during the experiments [10].

3. Related Work

A honeypot is a security resource that delivers insights by getting probed, attacked, or compromised by malicious entities [11] and subsequently reports the characteristics of the attack. Honeypots are classified as either low or high interaction based on their designed level of interaction with the potential attacker. Low-interaction honeypots, as their name suggests, are often limited by the degree of interaction. Examples of low interaction honeypots include Honeyd and Dionaea. On the other hand, high-interaction honeypots support complex behavior by emulating a real operating system with a full suite of applications. High-interaction honeypots carry more risk by allowing full compromise by the attacker. Honeypots have no legitimate production use for incoming traffic, so in theory any traffic hitting a honeypot can be deemed suspicious and warrant inspection.

On the other hand, a Honeynet is a basic network of commonly used operating systems in their default configurations. As a Honeynet does not broadcast its identity, all incoming traffic is deemed illegitimate and further investigated. Kuwatly et al.'s work [12] discusses a dynamic honeypot solution built from the fingerprinting tool (p0f) and Nmap to dynamically configure and leverage Honeyd-based emulated virtual hosts and a physical high-interaction honeypot cluster to capture and analyze attacker traffic. Hecker et al.'s paper [13] proposes a solution that uses nmap for fingerprinting the local network (topology, hosts, ports, etc.) and Honeyd-configuration manager for dynamically building honeypots.

Kishimoto et al.'s work [14] on dynamic honeypot commissioning involves detection of incoming address scans targeting an unallocated IP address. Research work done by [14] on commissioning honeypots based on incoming address scans shows that disabling Duplicate Address Detection (DAD) makes the address acquisition faster. During the prework tests, we analyzed the relevant network captures and found that we can significantly reduce the delay further by enabling a node to proactively claim ownership of the acquired address without waiting for NS messages from router. More details are presented in Section 4. Hieb and Graham [15] employ dynamic honeypots and monitoring network activity of deployed honeypots to set up anomaly-based intrusion detection for the network. Brzeczko's work [16] on Turnkey Honeynet framework involved automatic commissioning of Honeypots (Dionaea, Kippo, and Glastopf) depending on the composition of live attack traffic.

Memari et al. [17] built a virtual Honeynet and compared LXC virtualization with other virtualization methods including VMware, VirtualBox, and KVM and concluded that the LXC approach provides better performance. Work by Sokol and Pisarcik [18] on Distributed Virtual Honeynets framework talks about a master control center and a network of high-interaction virtual Honeynets based on OpenVZ and LXC virtualization.

Previous research work in this area involved creating Darknets, Network Telescopes, and Honeynets, with no advertised services and listen for illegitimate traffic. There are no current implementations adapting such a scheme to moving target defense schemes like MT6D. Also unlike a static darknet that passively listens on unallocated address space, our solution is dynamic as it learns IP-address and MAC address associations from live network traffic to actively acquire the addresses being purged in order to gather intelligence on attacker methods. Work presented in this paper extends the related work discussed above in terms of a full IPv6 implementation framework of on-demand honeypot commissioning using Linux containers (LXC) in addition to proposing a method to leverage such a solution to fortify MT6D defenses. This solution presents a simplified visualization of attacker traffic by ignoring the MT6D nodes that do not have interesting incoming traffic to enable an uncluttered view of the suspicious traffic. This work also presents a memory and CPU overhead analyses of the proposed solution while scaling to multiple honeypot containers.

4. Idea and Approach

The central idea of this research effort is to devise a mechanism to gather intelligence on attacker methods by watching for suspicious traffic on relinquished MT6D addresses in addition to be able to deploy a honeypot on a specific discarded address upon identifying a suspicious traffic pattern.

To implement such a scheme, as shown in Figure 1, we have a MT6D-host that periodically hops onto new addresses while relinquishing old addresses. A CN passively listens to the network traffic in promiscuous mode to parse the NS and MLD messages from MT6D hosts to populate a database collection of who is relinquishing what address, identifying the right time-instant to acquire these discarded addresses and subsequently binding these addresses to its local network interface. After the CN binds these discarded addresses to its network interface, it analyzes the incoming traffic on these discarded addresses and may place a request to the honeypot-host for a honeypot container to be deployed on a specific IPv6 address. The challenge is to migrate the IP address from the CN to virtual LXC container on the honeypot-host (honeypot container) with minimal interruption observable to an external attacker who is sending the traffic to the discarded address. The steps involved in this process are unbinding a specific IP address on the CN, binding this address on honeypot container, and finally invoking and

binding Dionaea and DionaeaFR services to this IP address on the honeypot container.

We propose an efficient and quick approach for honeypot deployment by maintaining hot spares of honeypot containers. To facilitate the hot spares approach, we packaged an LXC container with all prerequisites such as Dionaea, DionaeaFR, and supporting python libraries to act as a honeypot LXC-template. The honeypot-commissioning-module on honeypot-host clones a fixed number of containers from the honeypot LXC-template in advance and maintains a list of available containers. By keeping honeypot containers waiting in a queue, deploying a honeypot configured with a target IPv6 address requires that the only configuration needed on the container-side is binding the IP address to the network interface of an available container and starting Dionaea honeypot and DionaeaFR (management web server) services. This minimal configuration requirement ensures minimal delay in bringing up honeypot service bound to the desired IPv6 address.

In order to minimize the interruption window observable by an attacker during the migration of IP address from CN to honeypot container, we followed the approach of holding the address for a fixed-period after receiving the message that a honeypot is being deployed from the honeypot-host. For a smoother IP transition from CN to the honeypot container, based on our initial tests, we implemented the heuristic of CN holding the address for one second. This address-holding-period compensates for the time it takes the IP address to be bound on container's network interface and to start Dionaea and DionaeaFR services on the honeypot container.

We went with the approach of using the honeypot-host instead of running a honeypot service on the CN itself for two reasons. First, running honeypot on the CN may not be a secure approach as the CN not only maintains a database of the discarded addresses but also actively collects the addresses that are currently active on MT6D hosts from the parsing of NS and MLD messages. Secondly, a design including a dedicated honeypot host with its own database allows flexibility for such a honeypot-host running virtual machines with vulnerable-services to sit in a segregated zone like a DMZ in the future while its partner CN can still be on the local-network with MT6D nodes.

Even though we used a low-interaction honeypot, Dionaea, for experiments, by the virtue of using LXC containers in our solution, the solution allows high-interaction honeypot deployment on the container. So we are not limited to emulated services or virtual hosts in future experiments.

5. Testbed and Implementation

The test setup as shown in Figure 1 involves three Linux desktops running the Ubuntu 12.04 operating system connected to a layer-2 switch with an uplink to the university network router. Virginia Tech has fully operational production IPv6 network. The MT6D host periodically acquires new addresses while relinquishing old addresses. The CN identifies and acquires the discarded addresses on the local network and is also responsible for performing traffic enumeration, analysis,

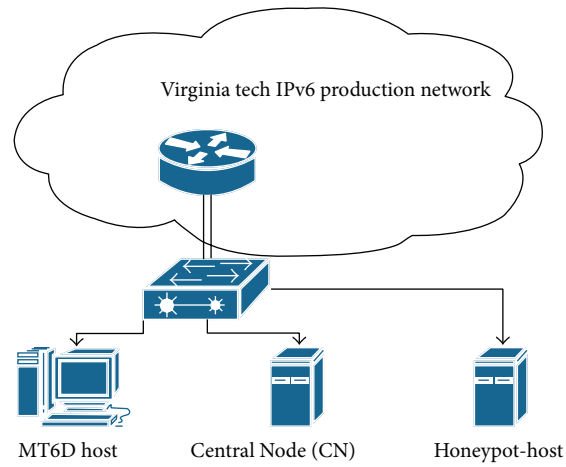

Figure 1: Block diagram of test setup.

and visualization. Third Linux machine is the honeypot-host system that is installed with LXC software to support on-demand creation of containers to act as individual honeypots. To set context for the memory and CPU overhead analyses in Section 6, the honeypot-host system is a Dell Optiplex Desktop running 64-bit 12.04 Ubuntu operating system installed with Intel Core-2 6700 2.66 GHz CPU, 4 GB RAM, and 80 GB of secondary memory.

Figure 2 depicts the architecture of our solution. The solution consists of CN block integrated with the database (MongoDB) and web server (Node.js) modules for inter-module communication and real-time visualizations. The honeypot-host block is implemented with on-demand honeypot deployment capability integrated with its dedicated (MongoDB) database.

The implementation involves building honeypot-host block and implementing database integration for CN and honeypot-host modules. The CN includes traffic-parsing, address-acquisition, enumeration, analysis, and visualization modules. The CN uses Pcapy and Impacket libraries in its traffic-parsing module to listen to neighbor discovery (NS and MLD messages) conversations of all MT6D nodes on local link to learn the addresses that are being relinquished by the MT6D nodes and stores these associations in a database. It employs the Pyroute library in address-acquisition module to acquire and bind these learned addresses to local host and Scapy to send out a forced NA claiming ownership of the bound address as an unicast to the local-router. The CN's traffic-enumeration module uses the same Pcapy and Impacket libraries to parse the incoming traffic on these acquired addresses and does traffic enumeration (Source IP address, Destination Port, and Packet Count), storing this data in the form of a native python dictionary. These python-dictionaries holding enumerated traffic data get converted to JSON objects for facilitating the visualizations. A node.js web server has been built on the CN to offer visualizations based on real-time traffic (MT6D addresses that are being spawned and incoming traffic on acquired addresses). The visualized data offers two views, MT6D_view and attacker_view (Figures

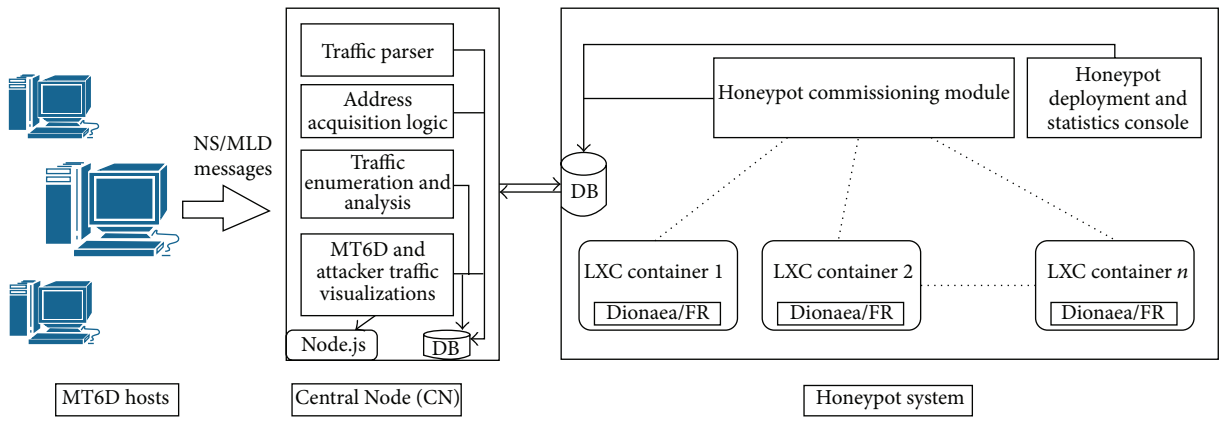

Figure 2: Solution architecture.

4, 7, and 8), allowing an administrator to see whether an attacker is able to trail the MT6D node along the spawned addresses.

The honeypot-host block includes the honeypot-commissioning module, the LXC, and a database. The honeypot-host database hosts two collections, hpot_requested_collection and hpot_deployed_collections, for exchanging information on addresses on which the honeypot is being requested or deployed. Figure 3 depicts the flow-diagram that explains the sequence of messages exchanged between the blocks and actions performed during the operation. Step 1 involves the CN listening to and parsing local link NS/MLD messages to keep a list of MT6D addresses being relinquished on the local network. In Step 2, upon seeing an MT6D node relinquish an address, the CN assigns the address to itself to facilitate traffic enumeration on this address. In Step 3, we have interesting incoming traffic hitting a discarded MT6D address (IPv6-address-of-interest) that is currently bound to the CN. Step 4 involves the CN requesting that a honeypot be deployed on a specific IPv6 address by writing the IPv6-address-of-interest to hpot_requested_collection on honeypot-host's database. In Step 5, the honeypot-host periodically checks hpot_requested_collection, retrieves the IPv6-address-of-interest, and acknowledges the CN by writing the IP address to hpot_deployed_collection in honeypot-host database.

In Steps 6 and 7, the CN continually checks for records in hpot_deployed_collection and upon finding a record the CN waits for a fixed time (the address-holding-period) and unbinds the address retrieved from hpot_deployed_collection's record. In the mean-time, as shown in Step 8, the honeypot-host checks for an available spare LXC container in the queue and invokes a shell-script to run lxc-attach commands to bind the IP-address on which honeypot is requested to the container's network interface, start the Dionaea service to bind the honeypot service to the newly assigned IPv6 address, and run the python web-server for launching the DionaeaFR front-end console. To speed up the address binding process, as specified in previous research effort [1], we disable Duplicate-Address-Detection (DAD) and use the concept of a forced Neighbor Advertisement (NA) thus advertising the new IPv6 address with the container's mac-address to the local router for quick address-acquisition. This completes the deployment of a honeypot tied to a desired IPv6 address and all future attacker traffic hits the honeypot container.

The MT6D_view visualizations (Figures 4 and 7) show each MT6D host represented by its mac address and all the addresses relinquished by each MT6D host and incoming traffic on each of these spawned addresses by source IP address and Protocol. To simplify visualization of incoming traffic without getting cluttered by new addresses that are being discarded by MT6D nodes and to observe the incoming traffic from the attacker perspective, the attacker_view visualization (as shown in Figure 8) view would show Source IP (potential attacker) of incoming traffic at its root and the visualization would then branch from each of these source-IPs of incoming traffic to a MT6D mac address and then branching to MT6D IPv6 addresses and incoming traffic composition in terms of protocol and port numbers and corresponding packet-counts.

In this chapter, we have discussed how we implemented various components of the proposed solution and explained how information flows between different modules. The next section presents evaluation of the solution in terms of our observations and results.

6. Results

Our solution at its heart involves efficiently spinning up containers on the honeypot-host and migrating the IP addresses from the CN to honeypot containers. To characterize such a solution, we will evaluate it in terms of interruption window observable by an attacker, as well as CPU and memory overheads. The interruption window observable to an attacker is a critical factor for judging the feasibility of the solution, since the duration of interruption window may offer a cue to the attacker about migration of his/her traffic from one machine (the victim) to another (the honeypot). CPU and memory overhead analyses offer valuable insights into scalability of the solution, that is, the number of honeypot containers that can be accommodated for a specific honeypot-host system

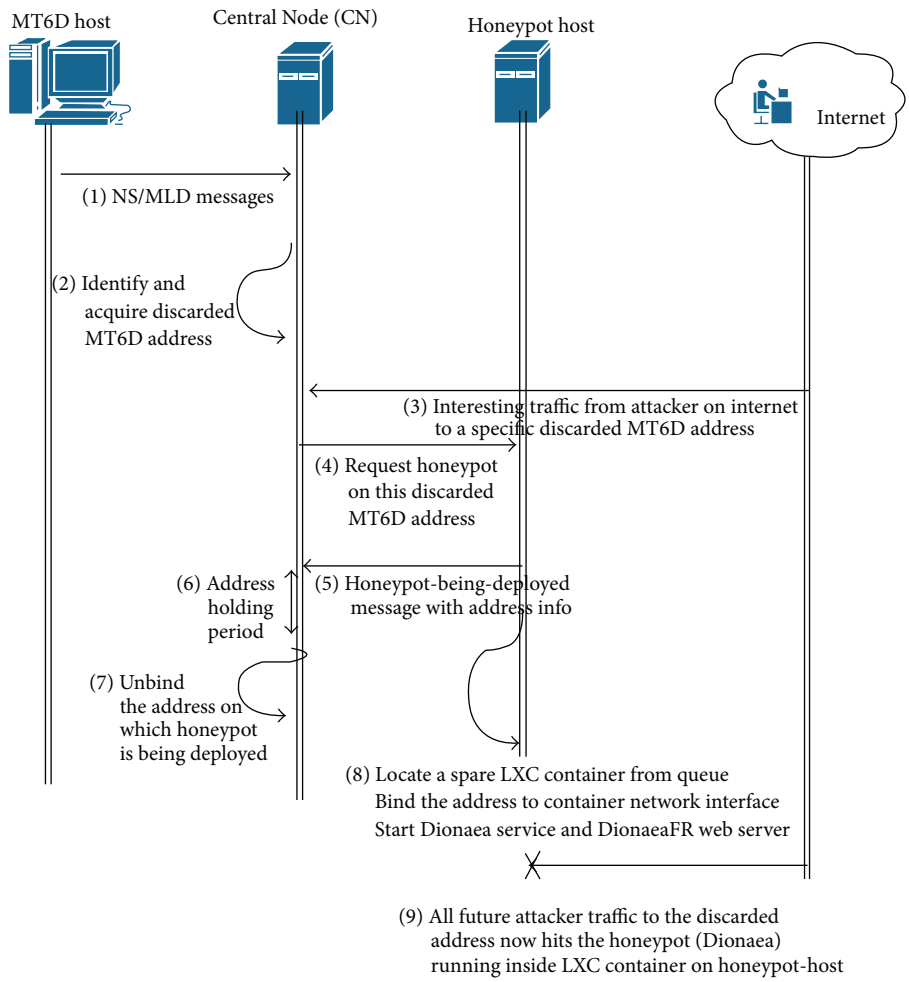

Figure 3: Flow diagram: sequence of messages exchanged between various components.

(CPU and memory) configuration. We will also discuss a mechanism for MT6D nodes to leverage our solution. In this section, we will explain the methodology of the testing and present our results.

6.1. Experiments and Results

6.1.1. Test Scenario. We first present a particular instance of potential attack-traffic hitting a discarded MT6D address and launch a honeypot-service bound to the specific address on an LXC container and supporting traffic visualizations. We then present results detailing the address-migration window times in terms of hold-packet-count and lost-packet-count for a ten-honeypot container deployment trial. We also present our observations from CPU and memory overhead analyses for single, five, and ten-honeypot container deployment trials. Hold-packet-count refers to the number of packets still hitting the CN since the trigger that deploys the honeypot, that is, the first ICMP probe hitting the CN. Lost-Packet-Count refers to the number of packets lost as seen by the attacker during the migration of an IP address to the honeypot container from the CN.

The experiment starts with an MT6D host with mac address 00:24:e8:42:c4:7a spinning off new addresses while dropping its old addresses. The CN listens to MT6D host's NS and MLD messages and parses them to identify and acquire the addresses that are being discarded by the MT6D host and visualizes the incoming traffic on these discarded addresses. Figure 4 shows the first level of a MT6D_view visualization with the MT6D host represented by its mac address, 00:24:e8:42:c4:7a, at the center, with each newly generated MT6D address (e.g., 2001:468:c80:c111:8c23:9665:ae9a:c757) represented by the last 64 bytes (e.g., 8c23:9665:ae9a:c757). The /64 subnet prefix remains the same, that is, 2001:468:0c80:c111 for all the MT6D address children nodes.

The next step involves sending ICMP echo probes (simulated attack traffic) every 50 ms from the machine (at 2601:5c0:c000:773e:1565:8df2:1408:2077) on a remote Internet connection to a particular discarded MT6D address, 2001:468:c80:c111:8c23:9665:ae9a:c757 that is now assigned to the CN. The first ICMP echo request packet serves as a trigger (as per configuration in the traffic enumeration and analysis module) for the CN to flag this activity and request a honeypot be deployed on this specific address.

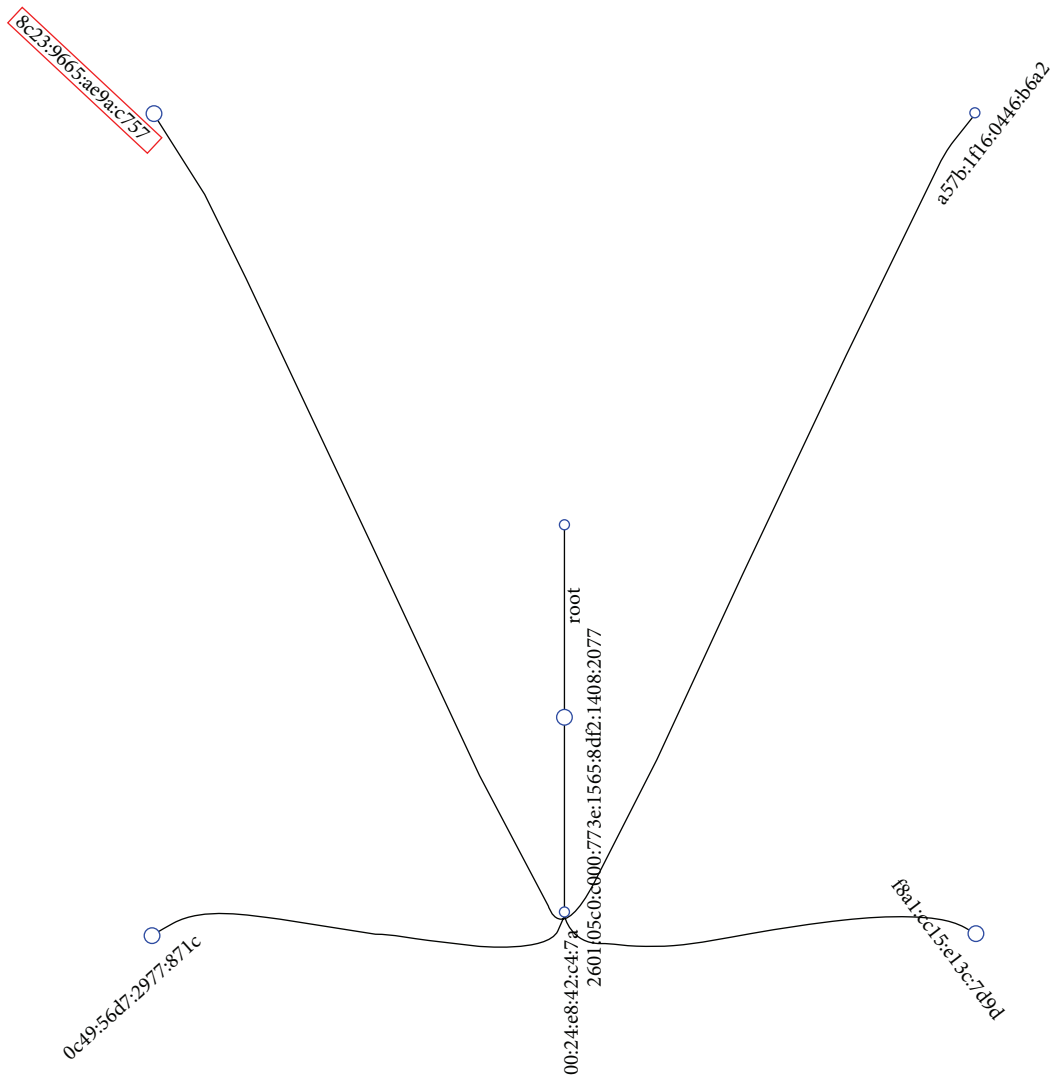

Figure 4: MT6D view depicting addresses spawned by a MT6D host.

The CN sends a "honeypot-requested" message with the address information to the honeypot-host database. The honeypot-commissioning-module retrieves the "honeypot-requested" message, acknowledges the CN with a "honeypot-deployed" message, picks the IP address, and looks up the container-queue for an available honeypot container. The honeypot-commissioning-module identifies an available container (in this case "hpot-container 1") and assigns the IPv6 address under attack, 2001:468:c80:c111:8c23:9665:ae9a:c757, to the local network interface of the honeypot container, and starts the Dionaea service and DionaeaFR python web server on the container.

Figure 5 shows an ICMP ping (sending ICMP echo request probe every 50 ms) results as seen by the attacker. Figure 6 shows the Wireshark capture running on the CN to identify the point when CN stops responding to the ICMP echo request packets incoming on the IPv6-address-of-interest indicating the unbinding. Figure 7 presents a second-level MT6D_view visualization of incoming attack traffic, showing an MT6D host with MAC address 00:24:e8:42:c4:7a receiving ICMP traffic of type 0x80 and a count of 24, on a specific discarded address 8c23:9665:ae9a:c757 from an attacker at address 2601:5c0:c000:773e:1565:8df2:1408:2077. Figure 8 presents attacker-view visualization depicting an attacker at address 2601:5c0:c000:773e:1565:8df2:1408:2077 sending traffic to MT6D host with MAC address 00:24:e8:42:c4:7a on a particular spawned MT6D address, that is, 8c23:9665:ae9a:c757, and the traffic composition is ICMP Type 0x80 and Code 0x00 (ICMP echo request) with a count of 24. MT6D_view and attacker_view present the visualization of the same traffic from two perspectives; the former shows the MT6D host at the center whereas the attacker_view puts the attacker-address at the center of the visualization and plots MT6D hosts that are getting hit from that specific attacker-address.

6.1.2. Observations on Interruption Window. From Figures 5, 6, 7, and 8 we can infer that the first 24 ICMP echo

```
16 bytes from 2001:468:c80:c111:8c23:9665:ae9a:c757, icmp_seq=19 hlim=52 time=25.866 ms
16 bytes from 2001:468:c80:c111:8c23:9665:ae9a:c757, icmp_seq=20 hlim=52 time=28.564 ms
16 bytes from 2001:468:c80:c111:8c23:9665:ae9a:c757, icmp_seq=21 hlim=52 time=30.185 ms
16 bytes from 2001:468:c80:c111:8c23:9665:ae9a:c757, icmp_seq=22 hlim=52 time=25.166 ms
16 bytes from 2001:468:c80:c111:8c23:9665:ae9a:c757, icmp_seq=23 hlim=52 time=24.148 ms
16 bytes from 2001:468:c80:c111:8c23:9665:ae9a:c757, icmp_seq=24 hlim=52 time=34.412 ms
16 bytes from 2001:468:c80:c111:8c23:9665:ae9a:c757, icmp_seq=26 hlim=52 time=31.686 ms
16 bytes from 2001:468:c80:c111:8c23:9665:ae9a:c757, icmp_seq=27 hlim=52 time=32.178 ms
16 bytes from 2001:468:c80:c111:8c23:9665:ae9a:c757, icmp_seq=28 hlim=52 time=32.715 ms
16 bytes from 2001:468:c80:c111:8c23:9665:ae9a:c757, icmp_seq=30 hlim=52 time=23.711 ms
16 bytes from 2001:468:c80:c111:8c23:9665:ae9a:c757, icmp_seq=31 hlim=52 time=30.459 ms
16 bytes from 2001:468:c80:c111:8c23:9665:ae9a:c757, icmp_seq=32 hlim=52 time=24.892 ms
16 bytes from 2001:468:c80:c111:8c23:9665:ae9a:c757, icmp_seq=33 hlim=52 time=24.720 ms
16 bytes from 2001:468:c80:c111:8c23:9665:ae9a:c757, icmp_seq=34 hlim=52 time=26.199 ms
```

Figure 5: ICMP pingflood traffic from simulated attacker on a remote Internet connection at 50 ms interval.

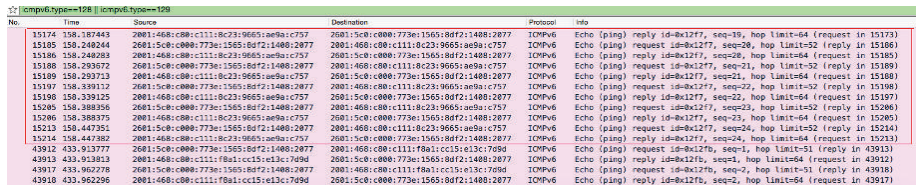

Figure 6: Wireshark capture showing packets hitting CN before the IP address migration.

request (ICMP type = 0x80 and code = 0) packets with sequence IDs till "24", hit the CN and one ICMP echo request packet with sequence ID "25" was lost, while packets (with sequence IDs from "26", "27",...) were routed to honeypot container resulting in subsequent ICMP echo replies in Figure 5 indicating successful IP address transition from CN to honeypot container.

To verify the address-transition and Dionaea service binding we used an nmap NSE auth-spoof (generally used for testing authentication servers for malware infection) script as shown in Figure 9 to generate test-traffic on various ports targeting address of interest, that is, 2001:468:c80:c111:8c23:9665:ae9a:c757. Figure 10 shows the DionaeaFR console displaying incoming connections for the address 2001:468:c80:c111:8c23:9665:ae9a:c757 on ports HTTPD, SMB, SIP, and so forth.

Figure 11 shows the hold-packet-counts and lost-packet-counts for the 10-honeypot deployment trial. We can also observe the IP address migration from the CN to the honeypot container on honeypot-host involves loss of one packet or none, from the perspective of an attacker probing a discarded MT6D address with an ICMP echo request packet every 50 ms. From the Wireshark packet captures, we attribute this packet-loss, that is, unresponsive ICMP echo requests, to the router on Virginia Tech production network either routing the packet to the CN even after the IP address is migrated to the honeypot container or the router forwarding the packet to honeypot container while the container is still in the process of binding the address to its network interface. In either case, even though the packet makes it to CN or the honeypot container depending on routing-table entry, neither of them can respond with ICMP echo reply as the address is not active on their respective network interfaces.

6.1.3. Observations on CPU and Memory Overheads. Figures 12, 13, and 14 show CPU and memory (RAM) loading characteristics for three independent trials of single honeypot container, five-honeypot container, and ten-honeypot container deployment. CPU time comprises Usr-CPU and Sys-CPU. Usr-CPU is CPU time spent in user-mode outside kernel code and Sys-CPU is the CPU time spent in the kernel within the process handling system calls and other kernel-space events. We observed that the trend of memory consumption with honeypot-service being deployed on each container is proportionally linear (~125 MB for each honeypot container launch) while CPU consumption peaks temporarily and then remains constant. The baseline free-memory consumption for the honeypot-host after a clean reboot was observed to be around 3025 MegaBytes. The baseline CPU consumption was observed to be nominal. From the observed memory and CPU consumption trend we substantiate the virtue of using the hot-spare LXC containers approach as the containers waiting as spares place nominal load on system resources and only request resources, Memory and CPU, on-demand when we invoke Dionaea and DionaeaFR management-server services on the container.

Figure 12 shows memory and CPU consumption for a single honeypot container experiment. From the origin until the point "A" on the free-memory curve represents the memory consumption on machine serving as baseline. At epoch time ~1443048570, the LXC container "hpot-container 1" is cloned from honeypot-LXC-template and the "hpot-container 1" is started and lies waiting as hot spare in the queue. We can observe a drop in free-memory at point "A". At epoch time ~1443048801, the honeypot-commissioning-module runs a script to bind the requested IPv6 address, start the Dionaea service, and start the DionaeaFR python web server on container "hpot-container 1". This step brings up Dionaea service bound to desired IP address on "hpot-container 1". We can observe a significant drop (~125 MB) in free-memory at point "B" on the curve in response to Dionaea and DionaeaFR service invocation. We can also observe

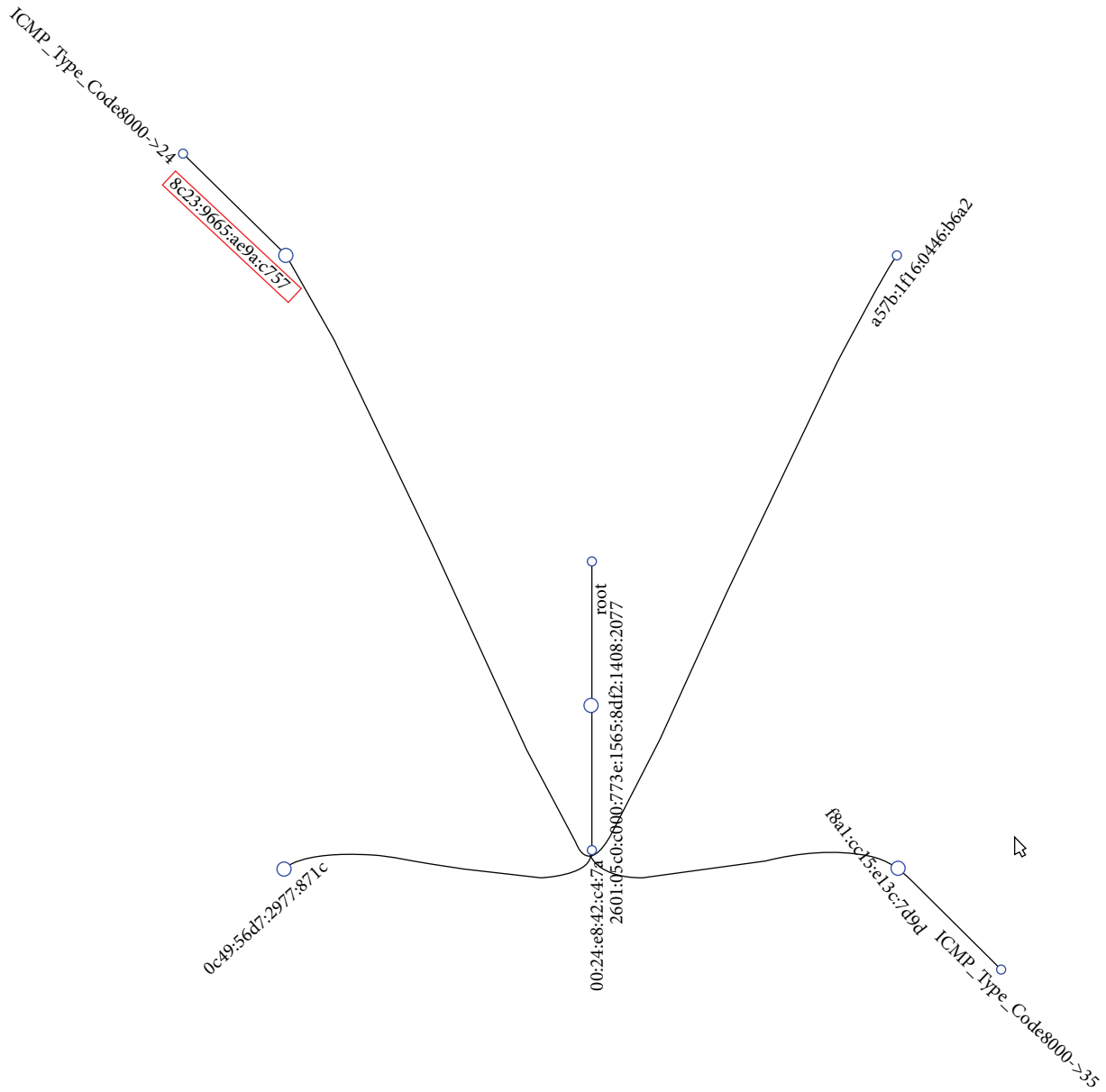

Figure 7: MT6D view of incoming traffic hitting the CN visualized.

the CPU (Usr and Sys CPU) consumption spike to a peak value at the time instant corresponding to point "B", that is, at the launch of honeypot services on the virtual container, and subsequently remain constant.

Figure 13 depicts the CPU and memory consumption trend for a five honeypot container trial. From origin until the point "A" on the free-memory curve represents the baseline memory consumption.

At epoch time ~1443023165 five containers ("hpot-container 1", "hpot-container 2",..., "hpotcontainer 5") are cloned from honeypot-LXC-template. These cloned containers are started and made to wait as hot-spares in queue. At epoch times corresponding to points "B", "C", "D", "E", and "F" on the free-memory curve, one can observe drops in available-memory corresponding to instants of binding requested IP address and invocation of the Dionaea service and DionaeaFR web-server on each LXC honeypot container. We can also observe the CPU consumption spike to peak value at the time instants corresponding to points "B", "C", "D", "E", and "F", that is, at the launch of honeypot services on the virtual-containers, and subsequently lie constant.

Figure 14 shows similar memory consumption behavior for a ten-honeypot-deployment trial, at epoch time corresponding to point "A", that is, ~1443029196; ten containers ("hpot-container 1", "hpot-container 2",..., "hpot-container 10") are cloned, started, and lie waiting in hot spares queue. Epoch times corresponding to points "B", "C", "D", "E", "F", "G", "H", "I", "J", and "K" on the free-memory curve correspond to IP address binding and invocation of Dionaea services. We can also observe the CPU consumption spike to

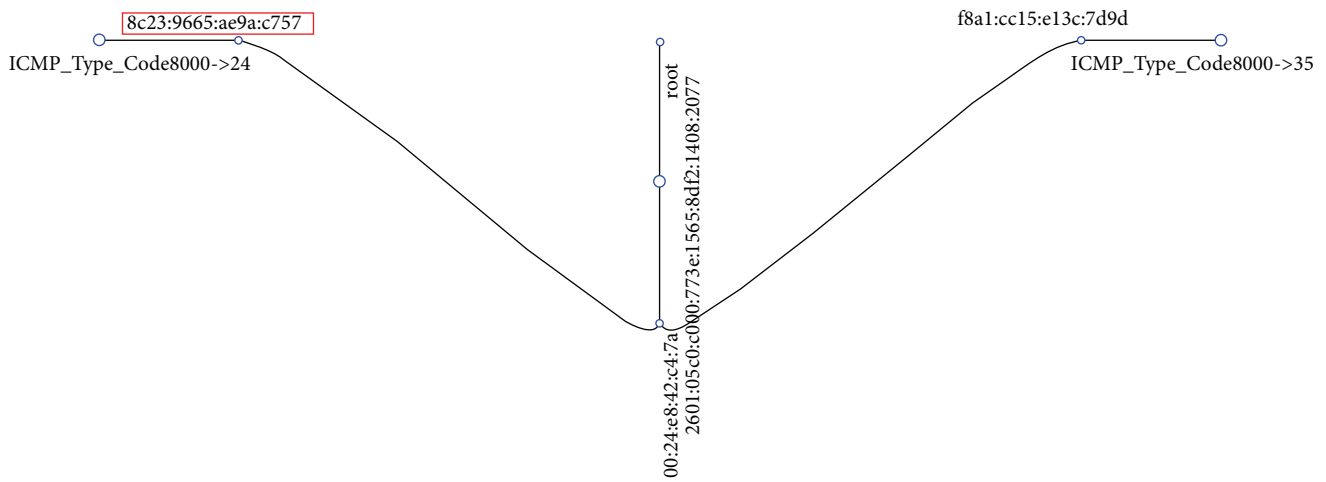

Figure 8: Attacker view of incoming traffic hitting the CN visualized.

```
pc1@pc1-OptiPlex-960:~$
pc1@pc1-OptiPlex-960:~$ sudo nmap -6 -sV --script=auth-spoof 2001:468:c80:c111:8c23:9665:ae9a:c757

Starting Nmap 6.40 ( http://nmap.org ) at 2015-10-02 16:51 EDT
```

Figure 9: Nmap script to send test traffic on various services to a deployed honeypot.

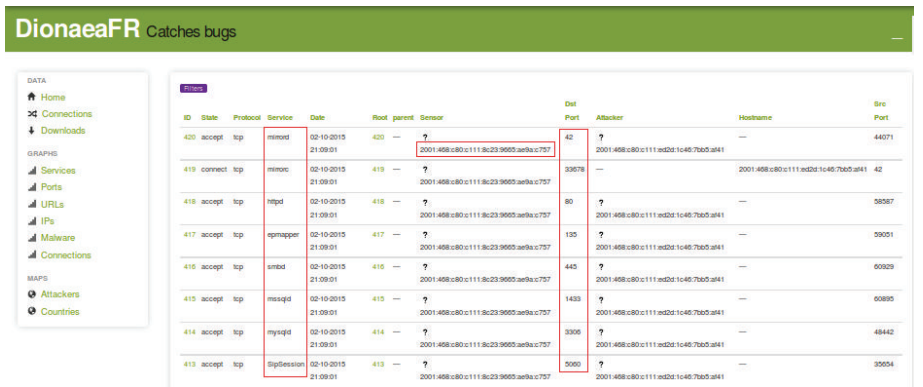

Figure 10: DionaeaFR console showing deployed honeypot connection statistics.

peak value at the time instants corresponding to points "B", "C", "D", "E", "F", "G", "H", "I", "J", and "K", that is, at the launch of honeypot services on the virtual-containers, and subsequently remain constant.

6.2. How Can a MT6D Node Leverage Our Solution. The MT6D nodes can consume the CN's intelligence on attacker-activity using the solution's Mongo database API. The JSON object responsible for Figure 8, the attacker-view visualization, can be queried from the CN's database by any MT6D host to detect a trailing attacker by identifying all the attacker-address nodes that have the MT6D host's MAC address as child node. In this case the MT6D host with the MAC address 00:24:e8:42:c4:7a can analyze the JSON object responsible for Figure 8 to understand its MAC address is a child node for the attacker address 2601:5c0:c000:773e:1565:8df2:1408:2077 and among its relinquished addresses particularly MT6D addresses ending with 8c23:9665:ae9a:c757 and f8a1:cc15:e13c:7d9d were hit with ICMP echo request traffic. Based on the attack traffic composition, the MT6D host can communicate with its MT6D partner and change its scheme parameters (e.g., stronger secret-key and faster hopping-interval) to evade any trailing attackers.

With this solution in place, nodes participating in MT6D can uncover any suspicious activity on their relinquished addresses and be able to use it in fine-tuning the scheme parameters. The solution's traffic enumeration and honeypot deployment capabilities offer insights into attacker methods. In this chapter, we evaluated our solution in terms of interruption window observable to an attacker, CPU, and memory overhead analyses in addition to discussing a mechanism to offer gathered intelligence on relinquished addresses to MT6D nodes through an JSON API. In the next section, we will conclude our findings and provide directions for future work.

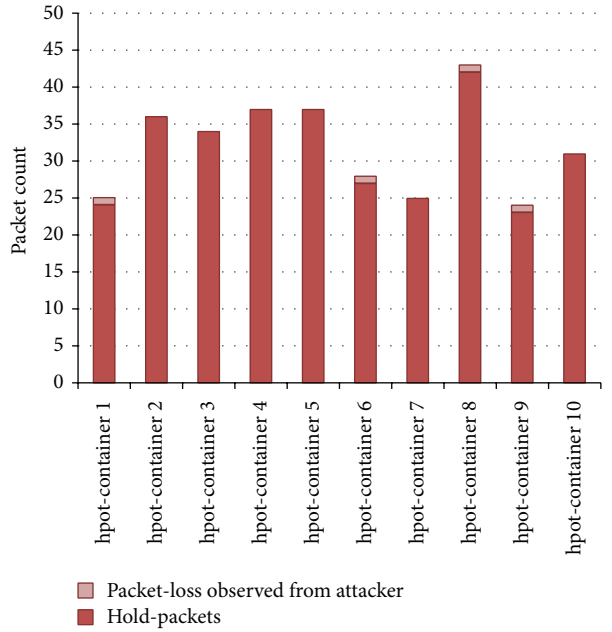

Figure 11: Chart giving out Hold-packet counts and packet-loss observed by an attacker during one of the 10-honeypot deployment trials involving sending an ICMP echo request every 50 ms, that is, rate of 20 packets/second.

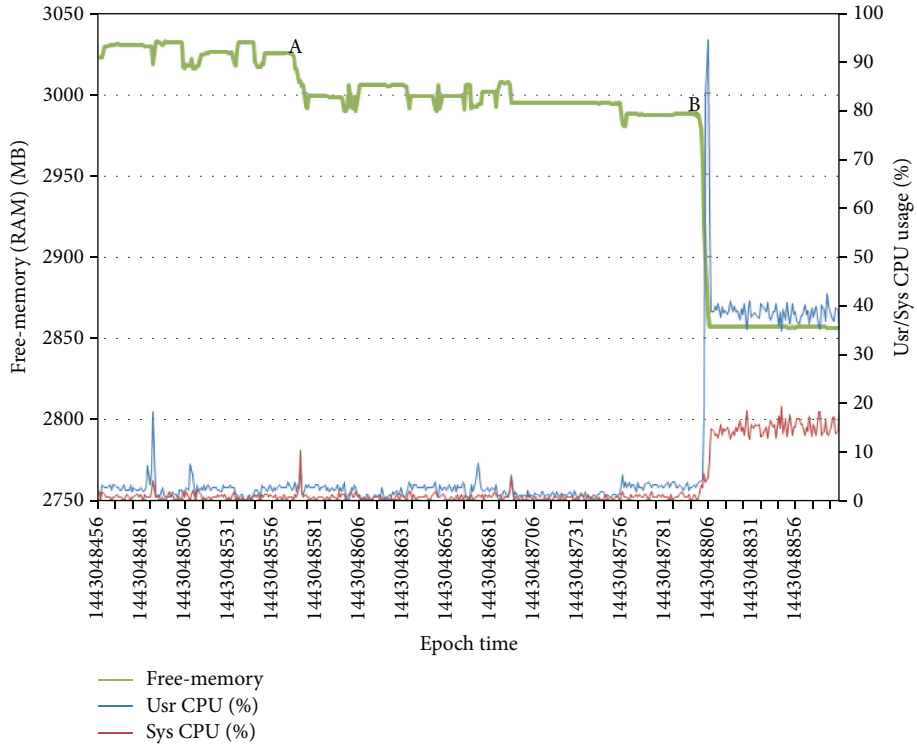

Figure 12: Memory and CPU loading characteristics for the scenario of 1-honeypot container hot spare and subsequent deployment.

7. Future Work

Future work should include CPU and memory overhead analyses of the solution under complex attack scenarios and evaluation of the solution's performance (IP-address migration times and packet-loss observable to attacker) with the honeypot-host sitting in a segregated environment like a DMZ. It is also important to perform overhead analysis on network devices, as such a solution grabbing discarded addresses on a complex MT6D network would place

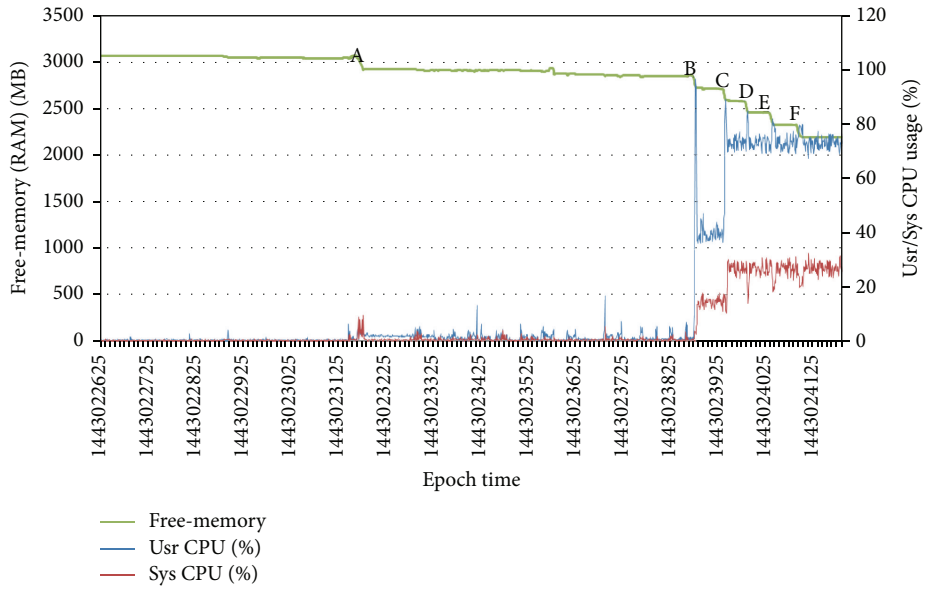

Figure 13: Memory and CPU loading characteristics for the scenario of 5-honeypot container hot spares and subsequent deployment.

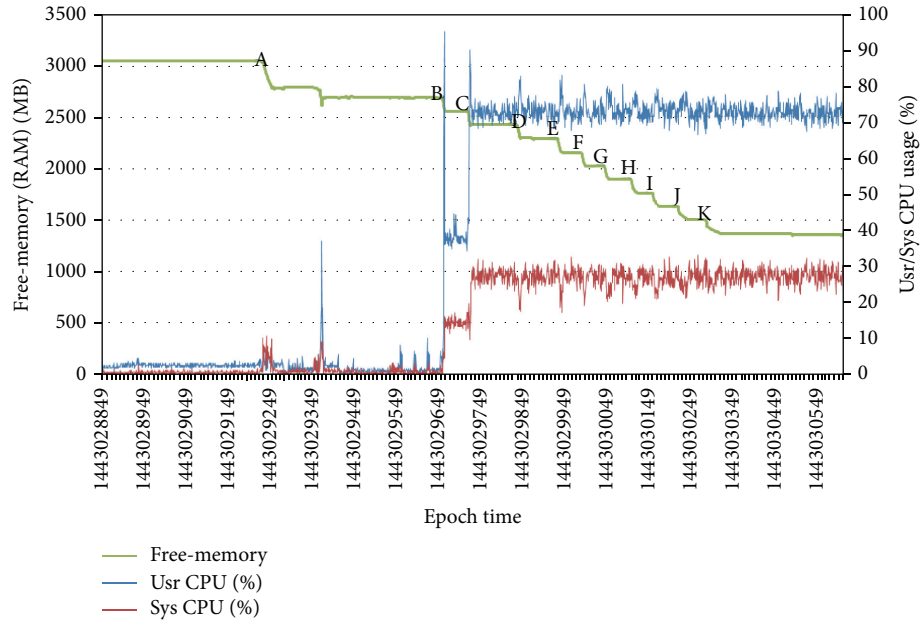

Figure 14: Memory and CPU loading characteristics for the scenario of 10-honeypot container hot spares and subsequent deployment.

overhead on the router in terms of persistent routing table entries. Future work can also include enhancing the solution with a new feature that would reclaim active honeypot containers based on no or low-priority incoming attacker traffic hitting a deployed honeypot container.

8. Conclusion

In this work, we proposed a solution to dynamically deploy a LXC container-based honeypot upon detecting suspicious activity on discarded MT6D addresses. We also evaluated the built solution in terms of interruption observable to an external attacker, CPU, and memory overhead analyses to demonstrate the feasibility and scalablility of such a solution. From our tests, we observed that the interruption observable to an attacker is one packet or none during the migration of the IPv6 address from the CN to the honeypot container. We also observed that the trend of the memory consumption with number of honeypot containers being deployed to be proportionally linear while CPU consumption peaks to a value and stays constant during the launch of the honeypot-service on each container. This supports the argument that

the presented solution in this effort is scalable provided that the memory and compute resources can be catered by the underlying honeypot-host hardware.

The implemented solution offers two visualization views, MT6D_view and attacker-view. MT6D_view displays nodes representing discarded addresses from all MT6D hosts in a local network and their corresponding incoming traffic. Attacker_view displays nodes corresponding to attackers addresses sending traffic and those specific MT6D hosts and discarded addresses that are receiving attack traffic. As we can observe from Figures 7 and 8 attacker_view simplifies the visualization over MT6D_view by the virtue of ignoring MT6D nodes with no interesting traffic. MT6D nodes with no incoming traffic no longer appear on the visualization, thus resulting in less-cluttered visualization of interesting activity. This paves way for visualizing suspicious activity on large MT6D networks without getting burdened by the periodic address-hopping activity.

The solution with database-integration also allows any MT6D node to query for traffic activity on its discarded MT6D addresses as a JSON object. This can be integrated as a feedback mechanism into work that our fellow researchers Morrell et al. are doing in the area of MT6D Client-Server stack in terms of an MT6D Server looking up suspicious activity on discarded addresses and communicating new MT6D scheme parameters such as longer secret-key or faster address-hopping interval to its MT6D client [19].

Previous work done by Morrell et al. [20] showed that a server can successfully bind up to 60000 randomly generated addresses, with each bound address communicating with an individual client. Based on these observations, the CN should be able to bind and listen on a large number of discarded MT6D addresses. We would also like to point out the solution's honeypot-deployment ability is unaffected by the number of MT6D nodes on network, as the incoming traffic enumeration and analysis are delegated to the CN, and the honeypot-host resources are engaged only when the CN matches incoming traffic with a configured attack-traffic signature.

Competing Interests

The authors declare that they have no competing interests.

Acknowledgments

The authors would like to thank Chris Morrell, Mike Cantrell, and Dr. David Raymond, Virginia Tech, for their many helpful suggestions and comments during the course of research.

References

[1] D. Basam, R. Marchany, and J. G. Tront, "Attention: moving target defense networks, how well are you moving?" in *Proceedings of the 12th ACM International Conference on Computing Frontiers (CF '15)*, article 54, ACM, 2015.

[2] M. Dunlop, S. Groat, W. Urbanski, R. Marchany, and J. Tront, "MT6D: a moving target IPv6 defense," in *Proceedings of the IEEE Military Communications Conference (MILCOM '11)*, pp. 1321–1326, IEEE, Baltimore, Md, USA, November 2011.

[3] M. Dunlop, S. Groat, W. Urbanski, R. Marchany, and J. Tront, "The blind man's bluff approach to security using IPv6," *IEEE Security & Privacy*, vol. 10, no. 4, pp. 35–43, 2012.

[4] Google, "Dionaea, the honeynetproject's 2009," 2009, http://dionaea.carnivore.it/.

[5] R. Espadas, *Dionaeafr*, 2014, https://github.com/rubenespadas/DionaeaFR.

[6] Canonical, "Lxc," 2015, https://linuxcontainers.org/lxc.

[7] P. Biondi, "Scapy," 2011, http://www.secdev.org/projects/scapy.

[8] G. Combs, *Wireshark*, 2007, http://www.wireshark.org.

[9] M. Bostock, "D3.js," Data Driven Documents, 2012.

[10] D. Wieers, "Dstat," 2013, http://dag.wiee.rs/home-made/dstat/.

[11] L. Spitzner, "Honeypots: catching the insider threat," in *Proceedings of the 19th Annual Computer Security Applications Conference*, pp. 170–179, IEEE, December 2003.

[12] I. Kuwatly, M. Sraj, Z. Al Masri, and H. Artail, "A dynamic honeypot design for intrusion detection," in *Proceedings of the IEEE/ACS International Conference on Pervasive Services (ICPS '04)*, pp. 95–104, IEEE, Beirut, Lebanon, July 2004.

[13] C. Hecker, K. L. Nance, and B. Hay, "Dynamic honeypot construction," in *Proceedings of the 10th Colloquium for Information Systems Security Education*, University of Maryland, Adelphi, Md, USA, June 2006.

[14] K. Kishimoto, K. Ohira, Y. Yamaguchi, H. Yamaki, and H. Takakura, "An adaptive honeypot system to capture IPv6 address scans," in *Proceedings of the IEEE International Conference on Cyber Security (CyberSecurity '12)*, pp. 165–172, Washington, DC, USA, December 2012.

[15] J. Hieb and J. H. Graham, "Anomaly-based intrusion detection for network monitoring using a dynamic honey pot," Tech. Rep. TR-ISRL-04-03, Intelligent Systems Research Laboratory, University of Louisville, Louisville, KY, USA, 2004.

[16] A. W. Brzeczko, *Scalable framework for turn-key honeynet deployment [Ph.D. dissertation]*, Georgia Institute of Technology, Atlanta, Ga, USA, 2014.

[17] N. Memari, S. J. Hashim, and K. Samsudin, "Design of a virtual hybrid honeynet based on lxc virtualisation for enhanced network security," *Defence S&T Technical Bulletin*, vol. 7, no. 2, p. 120, 2014.

[18] P. Sokol and P. Pisarcik, "Digital evidence in virtual honeynets based on operating system level virtualization," in *Proceedings of the Security and Protection of Information*, pp. 22–24, Brno, Czech Republic, May 2013.

[19] C. Morrell, R. Moore, R. Marchany, and J. G. Tront, "DHT blind rendezvous for session establishment in network layer moving target defenses," in *Proceedings of the 2nd ACM Workshop on Moving Target Defense (MTD '15)*, pp. 77–84, ACM, Denver, Colo, USA, October 2015.

[20] C. Morrell, J. S. Ransbottom, R. Marchany, and J. G. Tront, "Scaling IPv6 address bindings in support of a moving target defense," in *Proceedings of the 9th International Conference for Internet Technology and Secured Transactions (ICITST '14)*, pp. 440–445, London, UK, December 2014.