A simple moving target defense for power grid security using network address translation

Jacob Ulrich  
Iowa State University, ulrich@iastate.edu

Follow this and additional works at: https://lib.dr.iastate.edu/creativecomponents

Part of the Other Computer Engineering Commons, and the Power and Energy Commons

Recommended Citation
https://lib.dr.iastate.edu/creativecomponents/37
A simple moving target defense for power grid security using network address translation

by

Jacob Ulrich

A thesis submitted to the graduate faculty
in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

Major: Computer Engineering and Information Assurance

Program of Study Committee:
Doug Jacobson, Major Professor

Iowa State University
Ames, Iowa
2018
TABLE OF CONTENTS

LIST OF TABLES ........................................ iv
LIST OF FIGURES ........................................ v
CHAPTER 1. OVERVIEW ................................ 1
CHAPTER 2. REVIEW AND ANALYSIS OF RELATED WORK .... 4
  2.1 Fingerprint Hopping ................................ 5
  2.2 CHAOS SDN ........................................ 7
  2.3 RPAH - Random Port and Address Hopping .......... 9
  2.4 Gateway Proxy Hopping ............................ 11
CHAPTER 3. ARCHITECTURE AND ATTACK CHARACTERISTICS .... 13
  3.1 MTD Testbed Architecture ........................ 13
  3.2 Attack Characteristics ............................ 14
CHAPTER 4. MTD THEORY OF OPERATION AND IMPLEMENTATION ... 16
  4.1 Theory of Operation ............................... 16
  4.2 MTD Implementation .............................. 19
  4.3 MTD Protocol Handshake .......................... 21
CHAPTER 5. EXPERIMENTAL EVALUATION .................. 25
  5.1 Effects of MTD on SCADA Traffic Round-Trip Time .... 25
  5.2 Additional Traffic Due to MTD .................... 26
  5.3 Decreased Throughput Over a Saturated Link .......... 27
  5.4 Comparative Analysis of Performance Impact .......... 27
## LIST OF TABLES

<table>
<thead>
<tr>
<th></th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>5.1</td>
<td>MTD Round Trip Time</td>
<td>26</td>
</tr>
<tr>
<td>5.2</td>
<td>MTD Total Traffic Utilization</td>
<td>26</td>
</tr>
<tr>
<td>5.3</td>
<td>Time for Transfer of 10GB</td>
<td>27</td>
</tr>
</tbody>
</table>
LIST OF FIGURES

2.1 Network diagram showing the components of Fingerpring Hopping MTD ................................................. 6
2.2 Network diagram displaying the components of the CHAOS MTD .......................................................... 7
2.3 Network diagram showing the components of the RPAH MTD ............................................................... 9
2.4 SCADA diagram for Gateway Proxy Hopping ......................................................................................... 11
3.1 Network diagram of experimental setup .............................................................................................. 13
4.1 The flow of a packet in a gateway interface MTD .................................................................................. 16
4.2 The flow of a packet in a NAT MTD .................................................................................................. 17
4.3 Packet manipulations by queuing discipline filters ........................................................................... 22
4.4 Packet manipulations by queuing discipline filters ........................................................................... 23
5.1 Network traffic over time for various hopping intervals ....................................................................... 28
CHAPTER 1. OVERVIEW

The Smart Grid is a Cyber Physical system which relies on the interaction between complex Information Technology (IT) networks and Operational Technology (OT) networks. In the Smart Grid, control and monitoring of physical devices is facilitated through the SCADA network. Recent attacks, such as the Ukrainian Power Grid Attack, show a rising trend of sophisticated and persistent attackers targeting the static vulnerabilities of power systems and SCADA networks [2].

While SCADA networks are secured with traditional security devices like firewalls and intrusion detection/prevention systems (IDS/IPS), vulnerabilities persist. For example, an IPS can create a firewall rule to block traffic coming from an attacker who is attempting to flood a SCADA system and create a Denial of Service (DoS). The attacker can bypass this security measure by spoofing the IP address of a trusted or whitelisted machine. For a power system to function properly, there must be timely communication over SCADA networks. A loss of communication could result in major consequences that have the potential for cascading effects such as major blackouts. For this reason, any security measures implemented in a power system must be tested thoroughly to ensure SCADA communication is within an acceptable threshold.

A moving target defenses (MTD) attempts to address the security issues caused by the static nature of SCADA systems. Internet Protocol (IP) addresses, port numbers, operating systems, and application software usually remain constant during network operations.
This allows attackers the ability to accurately and easily footprint networks. An accurate footprint is needed to understand the attack surface of the system. An MTD continuously changes the static variables over to some time interval. This creates complexity for the attacker and it makes footprinting a costly activity. Additionally, it provides only a brief window for an attacker to execute a particular exploit.

This paper proposes a novel MTD for SCADA networks using an IP-Hopping algorithm to mutate the source and destination IP addresses of an IP packet as it is sent from one gateway router and then received by another gateway router. Since the MTD is implemented at the gateway routers of a SCADA system, the service is transparent to any end systems. This reduces computational overhead at the end systems and requires absolutely no configuration changes to individual hosts on a subnet.

This MTD method prevents attacks targeting internal networks because the IP addresses of the packets sent between the gateways appear random. These IP addresses change randomly and dynamically giving an attacker viewing this traffic the impression there is an immense number of hosts behind each gateway. Additionally, footprinting becomes a challenge as hosts will seem to go off-line as the IP addresses change. Therefore, this method prevents attackers from targeting resources on a SCADA network as well as reducing unwanted traffic to any particular end device.

The remainder of this paper is organized as follows: Section 2 is a review and analysis of design choices in related MTD research and implementations. Section 3 briefly describes the architecture of the SCADA network on the Iowa State University PowerCyber Testbed used to test the MTD; it also details the attack characteristics of the MTD algorithm. Section 4 describes benefits of our method and describes the functionality of the MTD algorithm. Section 5 presents our experimental results using our MTD method on the architecture de-
scribed in section 3. This section also analyzes the effectiveness of previous works. Section 6 analyzes performance characteristics of our MTD method to the MTD methods described in section 2. Section 7 provides a conclusion to the paper and discusses opportunities for future research.
CHAPTER 2. REVIEW AND ANALYSIS OF RELATED WORK

There is a growing interest and a vast array of research regarding MTD at the government, academic, and industry levels [14] [1] [2]. MTD can be applied to any layer of the OSI model. This section will mainly focus on research involving MTD at the network layer.

The Department of Homeland Security (DHS) is one of the largest government supporters of MTD research [14]. The DHS Cyber Security Division funds several MTD projects. They do this through support of their prime performers to include, Florida Institute of Technology, Carnegie Mellon University Software Engineering Institute, Def-Logix, IBM, and Princeton University.

Researchers from the Argus Cyber Security Lab at the University of South Florida, supported by the Air Force Office of Scientific Research released a product called Ancor [4]. Ancor is a cloud automation framework that creates an application layer MTD by changing the dependencies between various layers of the application stack. Network layer MTD using port and address hopping have been proposed in [5].

An MTD using the IPv6 address space is proposed in [9] and [10]. IPv6 uses a 128 bit address space, providing vast amounts of entropy necessary for an effective MTD. The model used by these sources is closer to a traditional IT environment, making them less useful for drawing conclusions about the effectiveness of an MTD in a SCADA environment.
Several papers have been published on developing Software Defined Networks (SDNs) to implement an MTD. Wang et al. [6] used SDNs in his proposal for MOTAG, an MTD algorithm to prevent Internet scale Distributed Denial-of-Service (DDOS) attacks. Kampanakis et al. [7] implemented an SDN-based MTD using Cisco’s One Platform Kit and showed its ability to prevent TCP host discovery and Operating System (OS) fingerprinting. Jafarian et al. [8] used an SDN Open Flow controller to implement an OpenFlow Random Host Mutation (OF-RHM) IP address randomization scheme, taking advantage of virtual IPs.

### 2.1 Fingerprint Hopping

In [22] Zhao et al. proposed an SDN-based fingerprint hopping MTD with the goal of preventing fingerprinting attacks. The goal of this MTD is to prevent Operating System (OS) detection from active and passive scanning tools. Knowing the version and OS of a system could allow attackers to launch scripted attacks. This information is especially valuable if the OS version shows the system is missing critical security patches.

Fingerprint hopping is achieved by changing several parameters used for OS detection in open-source and commercial scanning tools such as Nessus and Nmap [12]. This includes values like the TCP initial sequence number, the TCP window size and the time to live value of IP packets. This implementation randomly assigns these values to packets in the network to create an effective MTD. It accomplishes this through OpenFlow SDN rules controlled by the open source POX controller.

Figure 1 is a diagram showing the components of the fingerprint hopping MTD. These include the SDN controller, an IDS and the fingerprint hopping engine. The blue line is the path taken by trusted data. The red line is the traffic flow of untrusted data subjected to the MTD.
Figure 2.1  Network diagram showing the components of Fingerprint Hopping MTD

All data flows through the IDS. If an inbound packet is determined to be a fingerprinting attempt the IDS contacts the POX controller. The controller is responsible for all routing decisions on the network. It accomplishes this by pushing flow tables to the switches on the network. When the controller receives suspicious inbound traffic from the IDS it creates and tags a normal outbound packet to send to the attacker. It then sends the tagged packets to the Fingerprint Hopping Engine (FHE). The FHE receives the tagged outbound packet and randomly applies a new fingerprint to the packet from a database of false fingerprints. The controller then creates and pushes the appropriate flow to forward the modified packet to the appropriate outbound stream.

The fingerprint hopping MTD implementation succeeds in its goal of preventing OS footprinting attacks. The selective filtering of traffic using an IDS helps to improve performance in terms of latency and overhead since all network traffic is not obfuscated by the FHE. The problem with this approach is the assumption the IDS will catch all suspicious traffic. This is a weak adversarial model. Every IDS has some false negative rate meaning
some amount of fingerprinting traffic will not be detected. This is especially the case for traffic which has no fingerprint in the IDS database. In this case, the MTD will not be applied to the malicious traffic. This makes the MTD only as effective as the IDS.

2.2 CHAOS SDN

In [20] Shi et al. introduced an SDN-based MTD which utilizes decoy servers, port hopping and IP address hopping. This MTD provides security by preventing the targeting of hosts in a network. Additionally, the use of decoy servers provides defenders extra time to respond to an attack while also recording the attackers actions. This can be useful for forensically understanding an attack.

![Network diagram displaying the components of the CHAOS MTD](image)

The CHAOS MTD consists of several components. These include the CHAOS obfuscation module, the CHAOS tower module and the bro IDS. The components can be clearly identified in figure 2. All network traffic flows through the bro engine before it is passed to the tower module. Certain traffic will be chosen for obfuscation while benign traffic will be
unchanged.

The CHAOS MTD if implemented on Open vSwitch using the OpenFlow protocol and the Floodlight controller. The most unique component of this implementation is the tower module. The tower module is implemented completely within the scripting language of the Floodlight controller. It is a data structure which groups all hosts on the network into security groups by CVSS risk scores. Each security group is put into a priority layer and assigned a weight value by an administrator. The CHAOS tower defines a set of rules which determine the allowed communication between groups and layers. Any attempted communication which does not fall into these rules is marked as an unexpected communication.

The operation of the CHAOS MTD is straight forward. All traffic is first monitored by the IDS. Any traffic fitting a fingerprint is sent directly to the obfuscation module. Benign traffic is sent to the tower module. If the tower module determines traffic to be unexpected, then this traffic is also sent to the obfuscation module. An index score set by the administrator, based on the security policies in the tower module, determines the amount of obfuscation which occurs. For example, if the index score is low, only IP address obfuscation will be applied. If the index passes an administrator set threshold, then port obfuscation will be applied and traffic may be redirected to a decoy server. After traffic is obfuscated with random values, the proper flows are created by the controller to ensure routability of the obfuscated traffic.

The experimentns in this paper showed low latency and overhead of MTD traffic. It is speculated this is due to the selective application of the MTD to only suspicious traffic. This method was also shown to be effective at reducing but not eliminating footprinting and DoS attempts.
Again, a weak adversarial model was chosen. The tower module does help to prevent false negatives from the IDS. However there must be a false negative rate based upon the selection rules and security groups chosen for the tower module. This means the MTD is only as effective as the false positive rate of the detection system. Additionally, as the network begins to scale the management of the security groups and rules in the tower module will increase. This could cause severe maintenance overhead and cost.

2.3 RPAH - Random Port and Address Hopping

In [18] Luo et al. implemented an MTD making use of port and IP address hopping. This is a routable MTD which does not rely on an SDN and is instead implemented as a series of hosts within a network. It is capable of providing IP address and port obfuscation and is shown to effectively hide hosts.

![Network diagram showing the components of the RPAH MTD](image)
Figure 3 shows the components of the RPAH MTD. They include the Client Hopping Gateway (HGc), the Server Hopping Gateway (HGs) and the Port Hopping Engine (PHE). The HGc works by first authenticating hosts over PKI. This ensures they are a valid member of the MTD. The HGc uses the private key in conjunction with the known real IP of the server, a timestamp, and the clients virtual IP to calculate the virtual IP of the server. It uses a similar process to determine the virtual port numbers for the server. After determining these values, the HGc obfuscates the real source and destination IP of an egress packet and replaces them with the virtual values. Using the same information the HGc is able to deobfuscate ingress packets and forward them to the appropriate host.

The HGs performs the same functions as the HGc however it does not obfuscate or de-obfuscate port values for ingress and egress packets. Instead, the PHE located on the server performs this function. These functions are separated because if the destination address of a deobfuscated ingress packet is not in the subnet of the server, then it is invalid and the server does not need to process the packet.

As an added security measure, each component keeps a hash chain table. This is used to easily determine if incoming packets are valid by only doing a look up. This means the virtual IP address and port values could be calculated based on the PKI key values.

This MTD is unique because the various gateways do not need to communicate with one another once they have established PKI and an initial time value. This is made possible by the mathematical hash functions which generate the virtual IP addresses and port numbers.

The drawbacks of this method are the possible administrative overhead of managing PKI as this method scales. Several components must be implemented for each broadcast domain. This increases cost.
2.4 Gateway Proxy Hopping

Previous work on MTD at the network layer of the OSI model has been conducted at the Iowa State University PowerCyber testbed for target cyber-attacks by Pappa, et al. [3]. Their method dynamically mutates the IP addresses of the external interfaces of SCADA gateways. This method consists of few components and produces low overhead.

![SCADA diagram for Gateway Proxy Hopping](image)

Figure 2.4 SCADA diagram for Gateway Proxy Hopping

Figure 4 shows the experimental setup used by Pappa. The MTD is implemented using proxy rules in the Linux OS. The MTD is straightforward. A seed value is shared from one gateway to the other. The value is used to generate a random list of IP addresses for both gateways. The proxy rules are then updated to change the IP address of the interface on the gateway. After each IP hop, a TCP connection is opened and closed to ensure the
hops are synchronized.

Their tests on a realistic SCADA network showed that availability of SCADA systems could be maintained with a minimum four second hopping interval. This method does not hide the IP addresses of hosts behind these gateways. It does prevent targeting of the gateways themselves. However, An attacker who can intercept packets between these two gateways can build an accurate picture of the devices behind the gateways.

Our MTD method builds off of this work to create a defense-in-depth solution, obscuring the hosts in a particular SCADA subnet. The remainder of this paper describes how this can be accomplished.
CHAPTER 3. ARCHITECTURE AND ATTACK CHARACTERISTICS

3.1 MTD Testbed Architecture

Figure 5 depicts the architecture used for analysis of the MTD described in this paper. The test network was implemented in the Iowa State University PowerCyber testbed [15]. The network consists of the two gateway routers that facilitate communication between a substation and a control center. The control center consists of one SCADA server set to passively monitor and control the relay located on the substation network. An attacker is present and is capable of monitoring traffic and sending traffic between both gateways on the Wide Area Network (WAN).

![Network diagram of experimental setup](image-url)
The substation relay is the only physical device in this setup. The gateways, the attacker and the SCADA server are virtual machines implemented in VMware. All virtual communication is sent over virtual networks using a 10 gigabit per second (10Gbit) interfaces. The physical relay is connected to the virtual substation network by bridging the relay’s NIC to the substation virtual network.

As stated, the attacker is capable of monitoring traffic and carrying out a Man-In-The-Middle (MITM) attack. Normally, the attacker would monitor network traffic sent between the gateways which act as routers. If they can perform reconnaissance and successfully footprint the network, then from their position, they can easily exploit any vulnerabilities they may find.

3.2 Attack Characteristics

Prevalent security devices like firewalls and IDS/IPS system would have difficulty stopping an attacker in this situation because they rely on known information such as attack signatures and static IP addresses. A successful reconnaissance needed for an attack becomes far more difficult when an MTD is run in conjunction with these traditional security devices. This is because an attacker is given a small window of time to find an active IP address that can be attacked.

The defensive value of an MTD implementation values greatly, and is difficult to evaluate experimentally. MTD, as part of a defense-in-depth strategy seeks not to directly prevent attack execution, but raise attack cost due to increased complexity in reconnaissance and target discovery. Put simply, the objective is to confuse the attacker. As such, it has no easily-tested metrics, but the paper by Pappa, et al. does propose a model for a maximum probability of attack success. This was done by designating the time between MTD hops
as an attack window, and calculating the probability of successfully discovering the target
during this window. Further assumptions include no attack cost beyond target discovery,
and that any host reconnaissance information expires and becomes useless once the MTD
cycles to a new set of addresses [1]. The MTD implementation presented in this paper has
defensive behavior similar to the implementation described in that paper.

To discover an active IP and create a footprint, the attacker must scan for active network
hosts on a subnet using a host discovery tool such as Nmap. It took an average of 25 ms to
scan and discover a single active host on our network. On average, it takes approximately
6.47s to scan an entire range of hosts on a subnet with 254 host IP addresses. The lowest
hopping interval we tested was one second. These two values can be considered a lower and
an upper bound.

The worst case scenario for an attack is when an attacker begins their scan at the same
time that a new MTD IP address is initiated. While this is purely theoretical mapping, it is
intuitive the probability of a successful attack is reduced as the time interval of the MTD is
reduced. Taking the previously mentioned upper and lower bounds into consideration, the
probability of a successful attack would increase linearly between these bounds. Again, in
this case, a successful attack is when an attacker discovers an active MTD IP address. Once
they have this information they can attempt an additional attack further down the cyber
kill-chain. However, due to the nature of the MTD they will only have a brief moment to
carry out a follow up attack such as vulnerability scanning, OS detection, or a replay attack.
CHAPTER 4. MTD THEORY OF OPERATION AND IMPLEMENTATION

4.1 Theory of Operation

Previous network layer MTDs have utilized IP-hopping at the gateway interface level [1]. This MTD method provides security benefits by preventing attackers from targeting the gateway interfaces directly. However, on publicly routed SCADA traffic, such as in our scenario, this method is ineffective. If an attacker can intercept packets between the two gateways, then they have access to the IP addresses of the systems behind the gateways. With this information, attackers can still target the critical SCADA devices in a subnet.

Figure 4.1 The flow of a packet in a gateway interface MTD

Figures 5-6 show the various IP addresses of packets sent from the substation (SS) WAN gateway to the control center (CC) WAN gateway both without (Fig. 5) and with (Fig 6.)
address transaltion MTD in effect. If an attacker were to intercept or observe this packet, the attacker would then be equipped with the IP addresses of the devices on the SS LAN and CC LAN. While the interfaces of the routers on the MTD WAN are performing a network layer MTD by randomly choosing new IP addresses on a given time interval, and traffic sent from an attacker to the CC LAN or the SS LAN will be accepted and forwarded by either gateway.

Figure 6 demonstrates an MTD method using NAT based rules on the gateway. As a packet is sent from the SS LAN, the source and destination IP addresses are obfuscated to an MTD IP. The packet maintains this configuration as it moves over the MTD WAN. Once the packet reaches the CC LAN, the source and destination IP addresses are translated back to their original values and sent to the appropriate host. An attacker capable of sniffing packets on the WAN can now only attempt to send traffic to the machines on the CC and SS LANs. While the attacker does have a short window to send traffic this IP address will quickly change confusing the attacker and dropping the session.

One point of concern to address regarding the functionality of NAT rules for an MTD
is their default stateful nature. This is a problem if an attacker is able to gain a connection with a host during their brief window of opportunity while the MTD IP address is constant before switching to a new IP address. This is because stateful NAT rules will allow a connection over the MTD WAN to persist even is the NAT rules on the gateway are updated. This is not the functionality that we desire. Connections by devices outside of the CC and SS LANs should be dropped because they are not part of the MTD network. To force this behavior stateless NAT rules must be enforced. This way an attacker will be dropped upon an NAT rule change.

One final design decision concerns the routability of packets after the MTD obfuscation has been applied. This becomes a design trade-off between scaling the MTD namespace (which increases attack cost) and minimizing the amount of configuration overhead needed on intermediate routers. Several options can be enumerated:

- Only hop between IP addresses belonging to the same subnet as the actual host.
- Select and assign one or more MTD address ranges to each “real” subnet range.
- Select IP addresses from the IPv4 namespace as a whole.

The first and last options presented represent extremes of this trade-off. The first option requires no additional configuration of intermediate routers, but limits the MTD namespace to the address space of the original subnet. For a /24 CIDR network, this is only a namespace of $2^8$ bits, or $2^{16}$ bits for a /16 network. The third option represents the opposite extreme: it provides maximum namespace and therefore the greatest increase in attack complexity, but the packets become un-routeable unless intermediate routers coordinate with the MTD system to decode the packets.
The third represents a balanced medium. Each “real” address is assigned one or more “fake” ranges as the MTD namespace. These MTD ranges need not be of the same subnet size; for example a single /24 network may have a selection of four /16 address spaces assigned to it. When MTD is applied to hosts within the original /24 network, the address is selected from any one of the corresponding /16 address spaces. This approach does require additional configuration on intermediate routers, but very little more than is needed to configure the underlying routing in the first place. Static routes for each of the MTD ranges must be added to direct packets to the same next-hop as if they had not been obfuscated. Since each MTD range is explicitly associated with a fixed “real” range, this configuration is static, and there is no need for the intermediate routers to associate or synchronize with the MTD system.

4.2 MTD Implementation

The above theory of operation requires four sets of stateless Network Address Translation (NAT) rules. As packets travel from one host to another (for example, from a Control Center SCADA server to a remote RTU) it will undergo four transformations, two translations of the datagram’s source address (SNAT) and two of the destination (DNAT).

- SNAT from the real source address to an obfuscated source IP
- DNAT from the real destination address to an obfuscated destination IP
- DNAT from the obfuscated destination to the real destination
- SNAT from the obfuscated source to the real source

After the first two translations, the packet has been fully obfuscated according to the MTD scheme. Following the last half, the packet has been translated back to its original
During the packet’s transit through the network, it will pass through four interfaces. These interfaces are the LAN interface of the origin host’s gateway (CC LAN), the WAN interface of the origin host’s gateway (CC WAN), remote WAN (SS WAN), and remote LAN (SS LAN). It is logical to perform each of the four address translations at a different interface. As such, the final implementation strategy involves the following operations:

1. Packet transmitted by origin host
2. SNAT translation at the CC LAN interface to obfuscate the datagram’s source address
3. Routing decision at the CC Gateway
4. DNAT translation at the CC WAN interface to obfuscate the datagram’s destination address
5. Packet transits across the WAN
6. DNAT translation at the SS WAN interface to deobfuscate the source address
7. Routing decision at the SS Gateway
8. SNAT translation at the CC LAN interface to deobfuscate the source address
9. Packet arrives at destination host with no apparent modification

This implementation only requires trivial modification of packets – static, stateless translation of the source and destination fields in the IP datagram header. However, this requirement is not trivial to implement on the Linux operating system, as stateless NAT is not a typical requirement of networking systems. It is usually desirable to keep the addressing
of a connection stable throughout the duration of a connection. The widely-used IPTables utility manipulates the Linux netfilter, which heavily utilizes conntrack to perform stateful operations, especially with regards to address translation [13].

Implementation was done using the Traffic Control utility (tc), which manipulates the Linux packet scheduler. The packet scheduler provides active queuing management (AQM) for the packet queuing structures called ”qdiscs” (queuing disciplines). Queuing disciplines can be attached to the ingress of an interface, or in a tree-like structure to handle and process outbound packets. Queuing disciplines can perform filter-based actions on packets, including stateless address translation [11]. As such, the tc utility was chosen as a mechanism for implementing this MTD scheme on Linux gateways.

The implementation is done using a Python script on each gateway. The script is responsible for maintaining a time-synchronized mapping of the “real” IP to the obfuscated address, as determined by the synchronization algorithm described below. The tc utility is used by the script to create ”qdiscs” attached at appropriate points to each interface of the gateway (a ”qdisc” for both inbound and outbound packets is required on each interface). The tc utility is also used to apply the appropriate translations to each interface. These translation filters are replaced near-atomically [11] as MTD intervals elapse.

Figure 8 illustrates the interaction between the DNAT/SNAT rules and the tc scheduling rules.

### 4.3 MTD Protocol Handshake

The selected MTD algorithm, as proposed and demonstrated in Pappa, et al. is implemented through a handshake consisting of three steps [1]. First, upon initialization of the
MTD on the gateways, a seed is chosen. Next, a random list of IP addresses is generated by each gateway. Third, the gateways begin the IP-hopping process according to the chosen time interval. To ensure they perform IP-hopping simultaneously, when the gateways hop to a new IP address, they use a simple synchronization. Once the list of IPs is exhausted, this process is repeated from the first step with the selection of a seed and generation of a random list of IP addresses. Figure 9 illustrates this process.

The first step of our MTD algorithm is the seeding process. In our current implementation, the seeding process is done in a simple manner. N, 24-bit network addresses have been
Figure 4.4  Packet manipulations by queuing discipline filters

chosen, leaving eight bits for host addresses. 254 host addresses are appended to the each of
the N, 24-bit network addresses and they are added to a list. This gives a list of 254 * N IP
addresses. A random seed is then chosen and shared over an encrypted channel between the
gateways. In the second step this seed is used to shuffle this list. After the list is shuffled,
it is divided into two lists independently by each gateway. Each gateway now has the half
of the IP address list it will use and the list of the IP addresses the opposite gateway will
use. This is crucial to ensure that IP packets are encoded and decoded correctly by both
gateways.

The third step of the algorithm is to begin the MTD by setting the DNAT/SNAT rules
on each gateway according to the IP address lists. Synchronization of the gateways is ac-
complished by using a simple TCP connection. After both gateways have generated the
necessary IP address lists, they begin the MTD. Once the chosen time interval passes a
TCP socket is opened on each gateway and a TCP session is established. This session is immediate dropped by both gateways and the SNAT/DNAT rules are updated to the next entry in the IP address list. This process continues until the list of IP addresses has been exhausted. At this point the final step is to choose a new seed and start from the first step of the algorithm.
CHAPTER 5. EXPERIMENTAL EVALUATION

This chapter analyzes the security and availability characteristics of our MTD algorithm in a SCADA environment. The security benefits gained from our MTD implementation do not effect the availability of SCADA communication.

5.1 Effects of MTD on SCADA Traffic Round-Trip Time

The first test measured the impact of the MTD implementation (and MTD hopping interval) on the round-trip time of Distributed Network Protocol 3 (DNP3) transactions. This test was performed using the defined testbed environment, with only idle/monitoring SCADA traffic present on the network. The test was run for a span of five minutes, and traffic was recorded. The test was performed with no MTD active, then repeated with active MTD at intervals of 5, 4, 3, 2, and 1 second. The results summarized represent the average round-trip time (in milliseconds) between a DNP3 Read request and the corresponding response. The time is measured beginning from when the Control Center Gateway LAN interface first receives the request packet until the CC GW LAN interface transmits the response packet.

Table 1 contains the resulting round-trip time (RTT) of this test. The results show a slight increase of the RTT when an MTD of two to five seconds is used compared to the baseline RTT. The RTT of the one second MTD is nearly double the RTT of the baseline. Still, this is far below the DNP3 timeout value of 2000 ms and it is well within the
acceptable range. From these results, we can conclude this MTD implementation is efficient and produces little overhead in our test environment. This is important as we do not want to create a situation in which a large amount of packets are dropped or re-transmitted.

### 5.2 Additional Traffic Due to MTD

The second test performed measures the network traffic on the link depending on the presence and hopping interval of the Moving Target Defense. This test measures the network activity (bytes per second) over time, and includes DNP3 traffic as well as overhead due to MTD synchronization. The purpose of this test is to establish the additional amount of network traffic due to overhead as well as retransmissions.

<table>
<thead>
<tr>
<th>Average Traffic (B/s)</th>
<th>MTD Hopping Interval in Seconds</th>
<th>No MTD</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>580.74</td>
<td>385.21</td>
</tr>
</tbody>
</table>

An increasing trend can be observed as the MTD hopping interval becomes more frequent. This is expected, as a more frequent hop demands more synchronization overhead, as well as a greater likelihood of TCP segments dropping, which necessitates a retransmission. However, the total network utilization is trivial given the capabilities of modern networking systems, even at scale.
5.3 Decreased Throughput Over a Saturated Link

The final test was performed to examine the impact of MTD on a saturated link. This test was performed by timing the transfer of 10GB of test data between the two hosts. All of the links are nominal 10Gbit Ethernet links, fully virtualized. The test was performed using a single stream, and no disk IO was performed.

<table>
<thead>
<tr>
<th>Stream Transfer (s)</th>
<th>MTD Hopping Interval in Seconds</th>
<th>No MTD</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1</td>
<td>2</td>
</tr>
</tbody>
</table>

As can be seen, a high MTD hopping frequency can have a significant impact on the link throughput near saturation. In the most demanding case examined (1s hopping interval), MTD overhead accounts for over one sixth of the link throughput. However, this case is expected of a SCADA environment, and even scaling up, the worst case throughput impact is still tolerable for the requirements of a SCADA system.

5.4 Comparative Analysis of Performance Impact

Each paper provided an analysis regarding latency. In our example, we took the RTT of our DNP3 traffic. We can compare this to the RTT of TCP acknowledgements provided by each implementation. Looking at table 4 it can be seen the SDN implementations, CHAOS and fingerprint hopping, create the largest latency. While these are large spikes in latency, they are far from the time out value of DNP3 traffic. Unfortunately, these studies did not provide any data related to throughput overhead. Regardless, from a performance perspec-
RPAH and Proxy Hopping both outperform our implementation in regarding latency. We could most likely reduce our latency by optimizing our codebase. We could also look at alternate implementation methods such as modifying the Linux kernel or userspace TCP/IP stack. This could however introduce unecessary security vulnerabilities. In terms of throughput, our implementation has the best results with only a 5.95 percent overhead.
5.5 Comparative Analysis of Security Impact

One main factor in determining the security benefits of an MTD, is determining the configuration space. The configuration space is the total possible number of obfuscated values an MTD solution can produce. For example, in a class C network where 8 bits are available for host assignment, there are 256 IP addresses. Two of those addresses cannot be assigned because they are reserved for the network address and the broadcast address. This means there are 254 IP addresses available. In the case of a network layer MTD such as ours, using a class C address, the size of our configuration space is 254.

A logical hypothesis is an increase in the configuration space creates an increase in entropy. This makes it more difficult for an attacker to guess the proper configurations to footprint or address a host in an MTD. Another factor however, is the interval at which hopping occurs. Pappa showed in [3] there is an exponential relationship between the size of the configuration space and the hopping interval. In our case, as the number of IP addresses in our configuration space increases for a certain hopping interval, the probability of a successful attack decreases exponentially.

In the case of each of these papers, a successful attack was carried out if an attacker scanned a network, found an IP address and executed an attack within the time window of a single hop. The percentages shown in table 4 for the average success rate are the percentages for how often an implementation prevented an attack. So you can see, our implementation prevents attacks 85 percent of the time.

This is in line with the other implementations and could be improved by increasing our hopping interval or our configuration space. Additionally, the results for fingerprint hopping and CHAOS are somewhat dubious. Their results were taken under the assumption their IDS detected all malicious traffic. This is not realistic.
CHAPTER 6. CONCLUSION

This paper has proposed a network layer MTD for SCADA systems utilizing stateless address translation implemented with AQM and NAT rules. Our results show this method is a promising, automated tool which provides security while meeting the critical availability requirements for SCADA networks. The results of our testing have been positive. Still, additional work is needed to both thoroughly test this method and develop a useful industry security tool.

Future work will focus on expanding the MTD capabilities to provide complete cryptographically random and secure distribution of the seeding process. Additionally, a more robust handshaking method must be developed. The implementation of a port hopping scheme in conjunction with the IP-hopping MTD could provide additional entropy in the system. This would increase the security benefits of the MTD.

Another challenge is expanding the tool to be used as a scalable distributed system. Error correction methods are one design feature that must be implemented and refined to make this a reality. A standardized evaluation criteria should be developed for the purpose of testing the effectiveness of this and future network layer MTD tools. Specifically, tests should be developed to determine the security benefits and effects on availability of future solutions.
BIBLIOGRAPHY


