Design and Implementation of an Open Network and Host-Based Intrusion Detection Testbed with an Emphasis on Accuracy and Repeatability

Robert F. Erbacher
U.S. Army Research Laboratory
2800 Powder Mill Road
Adelphi, MD 20783
Robert.F.Erbacher.civ@mail.mil

Michael J. Shevenell
ICF Jacob and Sundstrom, for
U.S. Army Research Laboratory
2800 Powder Mill Road
Adelphi, MD 20783

***Abstract*—** The Open Network and Host Based Intrusion Detection Testbed (ONBIT) has been designed to make use of both network and host-based monitoring while validating and evaluating IDS tools and algorithms. This testbed was found to be of critical need for scenarios in which external testbeds cannot be used. The ONBIT testbed can be used to verify algorithms, concepts, and protocols, as well as discover more practical problems for future security research. This testbed is unique in its real-time nature and real-world performance and efficiency metrics; critical metrics for capabilities being readied for deployment. The ONBIT testbed was built using open source software and was designed to take accuracy and repeatability into consideration at each step of experimentation. Using a link emulator called DummyNet, the ONBIT testbed has the ability to control how the network behaves. DummyNet creates controlled packet loss, introduces latency, and allows for the configuration of various size network pipes. We show the benefit of correlating host-based and network-based IDS data in a real-world demonstration of the testbed's use.

Keywords- Network Testbed, Computer Security, Intrusion Detection, Deployability.

I. INTRODUCTION

Network testbeds have become a necessity for the validation and evaluation of networking protocols, algorithms, and techniques. While the need for testbeds holds for all areas of networking, computer security has both a unique need for testbeds as well as unique requirements for such testbeds. While extensive public access testbeds are available, such as DETER [2], there is often a need for an internal testbed, such as when techniques or data are sensitive in nature or when specific capabilities or configurations are needed; i.e., configurations or capabilities not yet supported by public testbeds. In particular, we had the specific need for accurate performance and result metrics DETER cannot provide due to its virtualized nature. These accurate metrics are necessary for evaluation and validation prior to product deployment.

In this paper, we describe the Army Research Laboratory's (ARL) ONBIT testbed. The testbed prototypes a complete network and host-based testing and experimentation solution from the hardware of the agent hosts, snort sensor and server host to the agent software executed on the hosts and servers. The testbed comprises various types of networks, network components, and network monitoring paradigms, including: emulated wireless, Snort-based collection, DummyNet [8] based network shaping, multiple variants of UNIX desktops, PC desktops, Android wireless agent hosts running host-based detection software, UNIX-based host agent event collection, and UNIX-based host-based alert generation. A UNIX workstation is used to capture network test data, graph and record experiments, and generate repeatable network exploits. The testbed has been built to test and evaluate the effects of limited bandwidth in network and host-based intrusion detection systems (IDS) and how various independent and dependent variables – such as agent host CPU load, disk IO, and types of applications – change the detection results. Wireless emulation is added to include the effects of delay and interference using devices such as Android phones.

Typical approaches to intrusion detection are either network or host-based methods. A network-based IDS (NIDS) collects packets from a sensor's network port and reconstructs the network flows. The IDS software applies various IDS tools to each reconstructed flow. Network-based intrusion detection seems to offer the most detection coverage while minimizing the IDS deployment and maintenance overhead. However, the main problem with implementing a NIDS using the reconstructed flows is the high rate of false alarms. Modern day enterprise network environments amplify this disadvantage as a result of the massive amounts of dynamic and diverse data that needs to be analyzed. A host-based IDS monitors a host and generates alarms when suspicious activity is observed. A weakness of host-based IDS systems is that the intrusion detection occurs on the single host running the IDS software, requiring massive deployment strategies for complete coverage. Another weakness is that the vulnerability occurs on the host, thereby exposing the IDS to whatever happens when the system is compromised. If attacks occur across several hosts then alerts may be missed, since each host must run the IDS software. Network and Host-Based intrusion detection each have their own weaknesses that are overcome using the testbed. The testbed will show that the combined application of network and host-based intrusion detection helps to overcome the problems of only using network-based or host-based detection.

A. Background

At a fundamental level, honeynets [12][13] provide the simplest mechanism for simulating virtual networks. These networks are not designed for experimentation but are rather geared towards simply providing the appearance of a network; there is no true functionality behind the simulation.

Guruprasad et al. [3] discuss Emulab, which uses both emulation and simulation in a hybrid environment; this is in contrast with live testing. The key with Emulab, as with the majority of such network testbeds, is that it focuses on creating large-scale virtual networks with sufficiently modeled detail at an extremely fine grain to allow experimentation with novel networking algorithms, protocols, and network configurations. However, the fact that this is a virtual network with simulated and emulated components ensures that some characteristics cannot be represented accurately, such as the real-world performance of the techniques that are the subject of the experimentation. Applications and higher-level tools can be experimented with in such environments; however, there is significant overhead due to the extensive underlying software infrastructure.

Barceló et al. [1] describes an open access network shared by multiple service providers. This approach segregates network access from network services. In terms of research, this would allow for experimentation with providing novel service capabilities in a dynamic fashion.

Knežević et al. [6] discuss a cost effective network testbed. This research is particularly critical for many scenarios in which large-scale network testbeds such as DETER cannot be used, e.g., in the case of sensitive data or techniques. In such scenarios, an organization will need to create their own testbed. The main concept with this technique is the simulation of multiple routers in distinct kernel threads; ultimately modeling entire topologies. This technique is again geared principally towards testing protocols.

Hellbruck et al. [5] focus on wireless sensor networks and the ability to perform experimentation with support for both users and operators. Of note is that testbeds for wireless sensor networks appears to have received significant attention. Importantly, the authors state: "In recent networking research, testbeds gain more and more attention, especially in the context of future Internet and wireless sensor networks (WSNs). This development stems from the fact that simulations and even emulations are not considered sufficient for the deployment of new technologies as they often lack realism." Given our focus on the need for realism with the goal of testing capability for production, the goal of employing a live testbed as opposed to simulations or emulations is critical; though our testbed must be more generic.

Motelab [10] is another testbed for wireless sensor networks. It is unique in that it is web-based. The web-based interface is used to create and schedule jobs. The network configuration appears to be standard with multiple sensor nodes connected to a central server.

DETER [2] is built on top of Emulab with the specific goal of supporting cyber security related experimentation needs. DETER has been enormously successful with large numbers of projects being supported [11]. However, DETER has well documented weaknesses, many of which directly affect the goals of our research [2]:

> "… (1) users limit scanning behavior of self-propagating malware, (2) users limit targets of disruptive actions, such as denial-of-service, to addresses within experimental network, (3) users limit their malware choice to well-known malware contained in the DETER-supplied library, (4) users limit experimental connectivity with the outside world to a set of machines under their control, and to specific protocols, (5) users limit the rate of traffic in their experiments, (6) users limit experimental traffic to the experimental network, (7) users implement signatures or self-terminating behavior in malware they plan to use."

As an example of the specificity and extent of research into testbeds, consider the work by Sheu et al. [9]. This work, while focusing on the creation of a distributed testbed for wireless sensor networks, also provides analysis of energy consumption.

B. Design Goals

The ONBIT testbed shows how network and host-based alerts can be correlated. The ONBIT testbed is a collection of 802.3 and emulated 802.11 MAC layer monitors, servers, agents, and host-based alert detection software designed to test network vulnerabilities under controlled conditions.

There are clear benefits to include host-based analysis and host-based tools in the ONBIT testbed. Researchers will be able to measure and evaluate how network based intrusions correlate with host-based intrusions. We can show that packet loss, latency, CPU load, and traffic saturation change IDS detection rates. The ONBIT testbed has been constructed using a wide variety of open source network testing and analysis tools, including:

- Wireshark. Network capture and graphing.
- TCP replay. Network replay tools.
- CILF. CPU and IO loading tools.
- Metaspolit. Exploit and vulnerability generation.
- Trafserver Trafclient. Host load generators to stress a network.
- Dummynet. Network traffic shaper.
- KauNet. Wireless traffic shaper.
- Snort. Network signature detection and OSSEC [4] for host-based detection.

By combining the network tools, the system allows for a wide system view that permits capturing and recording network activity on all aspects of the testbed experiment. The recorded session activity may be analyzed later or stored and replayed later to produce an attack on demand.

In addition, we include simulated network noise/load for realistic impact. ONBIT does not focus on simulating a compete network; rather, the focus is on *accurately* simulating the represented links.

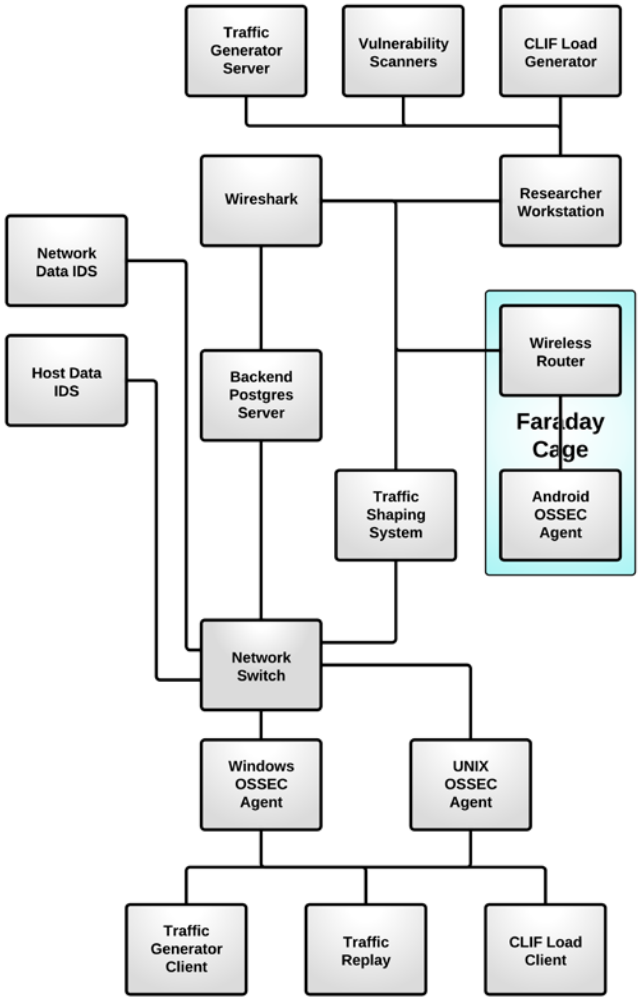

Figure 1: ONBIT testbed configuration.

II. The Architectural Design of ONBIT

The ONBIT testbed architecture is composed of multiple servers, hosts (agents), a network switch, network data sensors, and a system used to generate exploits and capture data. Using this architecture, we can define design principles and the ONBIT high-level goals. A few of the high-level goals are

- To provide researchers with an experiment testbed that supports an architecture that is realistic, scalable, robust, and decentralized.
- To provide a testbed such that novel network security vulnerability testing may be deployed and serve the research community.
- To allow for a design that supports experimental repeatability and accuracy required by modern network evaluation.

For the testbed to be useful, it needs to be scalable and cover a broad range of IDS research problem anticipated over the next few years. The system architecture of the ONBIT testbed is based on the general goals and requirements discussed above. The key design goals are summarized as follows:

- Reproducibility of experiments that can be captured and replayed on demand.
- An auditable trail of researchers experiments to retrace and reproduce activity.
- High degree of measurement capability at physical, MAC, and network levels, with the ability to correlate both network and host-based alerts. This will help to provide a realistic testbed environment.
- Tunable and flexible experimental control over software by providing the researcher with a rich set of networking tools.
- Scalability by allowing the number of agents to be expanded without significantly adding to the testbed hardware cost.

Figure 1 shows the ONBIT testbed configuration, architecture, and associated tools. The architecture is configured using the following primary components:

- Network IDS Collection Sensor
- Host-Based Collection Server
- Host-Based Collection Agent OSSEC [14]
- Traffic Shaping DummyNet/Kaunet System
- Researchers Workstation

A. Network IDS Collection Sensor

The network IDS collection sensor is used to collect network traffic and perform network IDS operations. The IDS runs Snort version 2.9.1 *Network Intrusion Detection System (NIDS) mode* in which snort reads the packets off the network and displays them for the analyst in a continuous stream on the console. This collection sensor also captures and filters packets using the Berkeley Packet Filter or BPF for post analysis and replay of data flows. Snort rules are tuned to collect alerts based on the criteria defined by the experiment. The collection sensor generates alerts based on network alerts and it is used to collect host-based alerts sent by the OSSEC host-based server. The collected host-based alerts are then correlated with the network snort alerts in a web portal display. For experimental purposes, the collection sensor is also the application web server.

B. Host-Based Collection Server

The host-based collection server collects data from clients, parses the data, and provides rule-based detection. This collection server performs log analysis, file integrity checking, policy monitoring, rootkit detection, real-time alerting, and active response. This collection server will compress and decompress data using zlib on data over a communication link. The template for our host-based detection system used for testing is based on the OSSEC open source code framework. OSSEC was chosen for two major reasons. First, OSSEC has proven detection

capabilities and is a leading host-based alert system. Second, since OSSEC is open source we have access to the source code, which is needed to customize the software to meet our specific requirements. We did not select a commercial off the shelf (COTS) solution due to our requirements to alter the source code and remain an open source solution. Using COTS solutions are difficult because the source code is not usually available. OSSEC relies mainly on real-time log file analysis and file integrity checking for intrusion detection. The server collects, analyzes, and correlates the data from clients. Client/server communication is encrypted (using pre-shared keys with blowfish) and compressed using zlib. Once an intrusion is detected, OSSEC alerts the security engineers by email.

C. *Host-Based Collection Agent OSSEC*

The host-based collection agent is a small program installed on all monitored hosts. It collects data in real time and forwards this data to the host-based collection server for analysis and correlation. This agent has a very small memory and CPU footprint by default, which will not affect the system's usage. These agents are responsible for collecting data, parsing said data, creating cryptographic hashes using two or more functions, scanning of event logs, and local detection of vulnerabilities. All agent events are sent to the host-based collection server to determine if the event should generate an alerted. A single agent event or multiple events may generate an alert on the host-based collection server. The agent runs as a low privilege user (created during installation) and inside a chroot jail, isolated from the system. Most of the agent configuration is pushed from the manager, with some parameters stored locally on each agent. If these local options are changed, the manager will receive this information and generate an alert. The agent will run on various operating systems, including Linux, MacOS, Solaris, HP-UX, AIX, Android, and Windows.

D. *Traffic Shaping DummyNet/Kaunet System*

The traffic shaper server is used to simulate/enforce queue and bandwidth limitations, delay, packet loss, and multipath effects. The traffic shaper server also implements a variant of weighted fair queuing called WF2Q+. DummyNet works by intercepting packets (selected by ipfw rules; ipfw is a FreeBSD software-based firewall) within the protocol stack, and passing them through one or more queues and pipes, which simulate the effects of bandwidth limitation, propagation delay, bounded-size queues, packet loss, and multipath routing.

Consider the following examples as to what you can do with DummyNet, either using it on your workstation or putting a PC with two Ethernet cards between your network and the outside world:

- Limit the total incoming TCP traffic to 2Mbit/s, and UDP to 300Kbit/s
  ipfw add pipe 2 in proto tcp
  ipfw add pipe 3 in proto udp
  ipfw pipe 2 config bw 2Mbit/s
  ipfw pipe 3 config bw 300Kbit/s
- Limit incoming traffic to 300Kbit/s for each host on network 10.1.2.0/24.
  ipfw add pipe 4 src-ip 10.1.2.0/24 in
  ipfw pipe 4 config bw 300Kbit/s queue 20 mask dst-ip 0x000000ff
- Simulate an ADSL link to a very distant location:
  ipfw add pipe 3 out
  ipfw add pipe 4 in
  ipfw pipe 3 config bw 128Kbit/s queue 10 delay 1000ms
  ipfw pipe 4 config bw 640Kbit/s queue 30 delay 1000ms

While previous work on network testbeds have discussed their architecture, this level of detail as to how to implement and deploy the software components has been sorely lacking.

E. *Researchers Workstation*

The researcher's workstation is used to generate test vulnerabilities, collect packets, generate statistics, and graph network data using Wireshark. Using this workstation, the researcher can control the experiment by sending crafted vulnerabilities to the agents and using Wireshark's packet capture capability to monitor network activity. The researcher has the ability to control the remote agents CPU and disk loads using CLIF as well as replay packet from experiments and generate network loads to stress the agent's network and software.

III. ONBIT HARDWARE AND SOFTWARE

A. *Hardware Components*

Our current hardware base is show in Figure 1. Experiments run on six real systems, each of which has dual core Intel(R) Xeon(R) CPUs @ 3.00GHz with 8 gigabytes of ram. The switch is a CentreCom FS716 Ethernet Switch. The server and UNIX agents run Redhat 2.6.18-194.el5 kernel and Microsoft Vista is installed on one of the agents.

B. *Software Components*

The software packages and libraries have been developed to support network, host-based applications and protocol evaluations. These software packages include open source libraries such as pcap and software packages for traffic generation, measurement, calibration, and collection. The software is flexible enough to enable researchers the ability to generate almost all known vulnerabilities as well as to create new vulnerabilities. Researchers may develop their own applications or add additional tools if needed. To give an idea of the flexibility that the software needs to provide, consider the following sample experiment scenarios.

Researchers may want to determine how a particular agent reacts to traffic stress testing and CPU loading; with the goal of measuring the packet loss, packet delays, etc. The researcher may want to send one of Metasploits known exploits and determine if the host-based collection server detects the alert. These types of network tools are provided by default.

More advanced researchers may want to design their own exploits or modify the OSSEC server or agents to behave in a particular manner. Using open source software allows the researcher the ability of modify the testbed to meet their particular needs. As an example, the researcher may require the host-based collection server to generate a custom formatted alert to fit a particular database or backend application. This would require changing the OSSEC server code that is written in the c programming language.

The open source software tools and configuration capabilities installed fall into the following categories:

- Bandwidth Measurement and Network Traffic Loading
- Exploit and Vulnerability Generation
- Bandwidth Limitation and Delay Testing
- Random Packet Loss
- Dynamic Queue Creation
- KauNet: Testing OSSEC Agents
- Load and CPU Testing

1. Bandwidth Measurement and Network Traffic Loading

Wireshark is used to measure, collect, and graph network traffic. Wireshark is a network packet analyzer. A network packet analyzer will capture network packets to display with packet data as detailed as possible. A convent feature in Wireshark is the ability to graph the collected data. Wireshark testing features include:

1. Show statistics summary. The statistics summary shows many details about the traffic over the entire test.
2. Show packets being collected. Typically, it will not be possible to display the payload information since it is compressed and encrypted. However, it is possible to display the communication between the OSSEC server and the OSSEC agent.
3. Show statistics in I/O graphs. These graphs are generated and loaded into the scientific data collector application.

Two standard networking tools, tcprewrite and tcpreplay, are used to playback recorded Wireshark raw data. This allows us to playback experiments for verification and repeatability. For instance, to replay the data stored in the file wireshark_udp.dat onto the network associated with eth0:

```
tcpreplay -i eth0 wireshark_udp.dat
```

We employ two traffic generation tools, trafclient and trafserver, to generate TCP and UDP traffic to emulate a real network environment and stress the network operations of the OSSEC agent. This allows out small testbed configuration to exhibit the network traffic of a complete network setup. A command must be issued on both the server

```
trafserver -sn basic -rn echo tcp 11111 udp 11111 tcp 11112 udp 7001
```

and the client:

```
trafclient -dn tcp -dp 11111 -da 10.10.10.2 -pn flxed -pmax 1000 -cn fixed -cmax 1 -n 1
```

in order for the network traffic generation to be fully initiated. More specifically, to simulate 16 megabits of traffic the following client command can be used:

```
trafclient -dn tcp -dp 11112 -da 10.10.10.2 -pn random -pmin 100 -pmax 1500 -cn fixed -cmax 5 -n 5
```

This second example demonstrates 4.5 megabits of data:

```
trafclient  -dn udp -dp 11111 -da 10.10.10.2 -pn random -pmin100 -pmax 550 -cn fixed -cmax 3 -n 3
```

We can also generate traffic using the ping command. The –A and –s options to ping provide the relevant packet control. –A forces only a single unanswered packet to be on the network at a time. –s specifies the number of data bytes to be injected into the ICMP payload. The following command generated 5.8 Megabits per second of traffic:

```
ping -A -s 1500 10.10.10.2
```

2. Exploit and Vulnerability Generation

We can use both the command line and Metasploit to generate OSSEC alerts. To generate attacks using the command line on the attack box we can use commands as simple as ssh to initiate a root login to the OSSEC agent box and force a root access deny. This generates a root deny access alert. The OSSEC agent detects the update to the messages file and sends the alert to the OSSEC server for processing. The second type of alert generation uses Metasploit. Generating alerts using Metasploit, msfgu, allows us to reproduce almost any known exploit with a payload and test it against the agent system running OSSEC.

3. Bandwidth Limitation and Delay Testing

Dummynet: Installed and being used for testing agents.

A DummyNet bridge is used to exhibit network delays and bandwidth limitations on the network. Such DummyNet bridges can be configured as follows:

```
brctl addif br0 eth1
brctl addif br0 eth2brctl addif br0 eth1
ip link set br0 up
brctl show
brctl addif br0 eth1
brctl addbr br0
route add default gw 10.1.1.1 dev br0
```

DummyNet can be initially installed and preliminarily configured to simulate/enforce queue and bandwidth limitations, delays, packet losses, and multipath effects. More specifically, the commands to control bandwidth and propagation delay to the OSSEC agent being tested are as follows:

```
ipfw3
insmod dummynet2/ipfw_mod.ko
ipfw add pipe 1 ip from any to any via br0
ipfw pipe 1 config bw 57Kbit/s delay 300ms
ipfw pipe 1 config bw 1000Mbits  delay 0
```

To limit the queue size of pipe 1 to 100 Kbytes:

```
ipfw pipe 1 config queue 100KByte/s
```

To set the propagation delay to 100ms:

```
ipfw pipe 1 config delay 100ms
```

4. Random Packet Loss

The packet loss in a network can also be simulated in DummyNet. The command plr X, where X is a floating-point number between 0 and 1, causes packets to be dropped at random to simulate packet loss, where 0 indicates no loss and 1 indicates 100%packet loss. For instance, 50% packet loss would be generated with:

```
ipfw pipe 1 config plr 0.5
```

5. KauNet: Testing OSSEC Agents

KauNet is an extension to the well-known DummyNet network emulator. The KauNet emulation system provides additional pattern-based emulation not available in DummyNet. KauNet enables bit precise placement of bit-errors, exact and repeatable packet loss, delays, and bandwidth variations.

6. Load and CPU Testing

CLIF is being used to monitor and load our system running the OSSEC agent. CLIF supports the concept of "Probes" which monitor agent resources and "Injectors" which load the agents with controlled and emulated user activity. CLIF allows the central console the ability to monitor system resources on the system running the Host based agent. The console displays real-time graphs of the agents CPU, memory, and disk utilization during the tests. CLIF also allows for loading the agents memory, disk activity, and CPU by emulating user activity. CLIF is fully programmable to allow for user emulation flexibility. To start CLIF on a system, on the OSSEC agent:

```
cd /root/clif-2.0.5-server
ant -f build.xml server
```

On the monitoring system:

```
clif/clif-2.0.5-eclipseconsole
./clif-console
```

Following the directions in the console will allow the user to deploy a CPU emulation test plan.

IV. EXAMPLE EXPERIMENTAL SETUP USING ONBIT.

This example experiment exemplifies how simulated attacks can be generated to evaluate the effectiveness of intrusion detection systems. This experiment employs DummyNet to control bandwidth, delays, packet loss, and multipath effects on a wired network. This experiment also uses network load generators to attempt to emulate a real world environment. The experiment includes packet degradation, i.e., packet loss, to affect the overall alert performance.

The experiment collects IO, disk, memory, and cpu measurements during the tests on the agents. The experiment will collect bandwidth measurements, generate packet and byte transfer graphs, and store the results in the data collector. The experiment will record the client agent sessions for playback of results. The playbacks may be used for future verification of the experiment. Each experiment is recorded and repeatable.

The system was configured to be vulnerable to a well-known reverse shell exploit that takes advantage of vulnerability in the distccd daemon on a UNIX Agent. When the distccd vulnerability is exploited, using the researcher's workstation, the experiment is configured to generate five OSSEC alerts and five network Snort alerts. Before conducting the experiment, Snort and OSSEC rules were preconfigured to generate the desired alerts.

On the client, the following command starts the vulnerable distcc daemon.

```
/usr/bin/distccd -p 33237 --allow 10.10.10.2
```

From the researcher's workstation, Metaspolit is executed to attack the vulnerable distccd daemon. The following are the used Metasploit commands:

```
msf > use unix/misc/distcc_exec
msf exploit(distcc_exec) > set rhost 10.10.10.3
rhost => 10.10.10.3
msf exploit(distcc_exec) > set rport 33237
rport => 33237
msf exploit(distcc_exec) > set payload cmd/unix/bind_perl
payload => cmd/unix/bind_perl
msf exploit(distcc_exec) > exploit
[*] Started bind handler
[*] Command shell session 1 opened (10.10.10.2:36490 ->
10.10.10.3:4444) at 2011-08-08 11:34:21 -0400
msf exploit(distcc_exec) > cp /etc/passwd /tmp
msf exploit(distcc_exec) > /usr/sbin/tcpdump
msf exploit(distcc_exec) > ls
distcc_1d02ca55.stderr
hsperfdata_root
jdk-6u20-linux-x64.bin
keyring-qBz1rK
libaprutil-1.so.0
libaprutil-1.so.0.3.4
orbit-tracker
pulse-iMrqdAG4wz26
ssh-hvfMGe1452
virtual-tracker.sdoTqM
msf exploit(distcc_exec) > pwd
/tmp

Abort session 1? [y/N]
msf exploit(distcc_exec) > y
```

The Metaspolit exploit execs the following reverse shell command on the 10.10.10.3 UNIX Agent:

```
perl -MIO -e '$p=fork();exit,if$p;$c=new
IO::Socket::INET(LocalPort,#{datastore['LPORT']},Reuse,1,Listen)-
>accept;$~->fdopen($c,w);STDIN->fdopen($c,r);system$_ while<>
```

On the network IDS collection sensor, OSSEC rules have been preconfigured to generate events created by the

Metaspolit reverse shell attack. The following are the OSSEC generated events:

2011 Aug 10 17:25:14 Rule Id: 1002 level: 2
Location: (tracker-desktop) 10.10.10.3->/var/log/messages
Unknown problem somewhere in the system.
Aug 10 17:39:48 tracker-desktop kernel: [21614464.721451] type=1503 audit(1313012388.268:95): operation="open" pid=2704 parent=2700 profile="/usr/sbin/tcpdump" requested_mask="::r" denied_mask="::r" fsuid=65534 ouid=0 name="/dev/bus/usb/"
2011 Aug 10 17:23:52 Rule Id: 803 level: 10
Location: (tracker-desktop) 10.10.10.3->/var/log/syslog
distccd
Aug 10 17:38:26 tracker-desktop distccd[13968]: sh main.c on localhost completed ok
2011 Aug 10 17:23:52 Rule Id: 806 level: 10
Location: (tracker-desktop) 10.10.10.3->/var/log/syslog
distccd
Aug 10 17:38:26 tracker-desktop distccd[13968]: (dcc_r_file_timed) 10 bytes received in 0.000027s, rate 362kB/s
2011 Aug 10 17:23:52 Rule Id: 804 level: 10
Location: (tracker-desktop) 10.10.10.3->/var/log/syslog
distccd
Aug 10 17:38:26 tracker-desktop distccd[13968]: compile from main.c to main.o
2011 Aug 10 17:23:52 Rule Id: 802 level: 10
Location: (tracker-desktop) 10.10.10.3->/var/log/syslog
distccd
Aug 10 17:38:26 tracker-desktop distccd[13968]: (dcc_check_client) connection from 10.10.10.2:58754

A. *Snort Identified Definitive Attack*

The following are the network-based Snort generated alerts caused by the Metasploit attack. The alerts are generated from the Snort alerts configured and processed on the network IDS collection sensor.

08/10/11-21:25:52.960396 [**] [1:20110394:1] distcc_exec connection warning [**] [Priority: 0] {TCP} 10.10.10.2:58754 -> 10.10.10.3:33237
08/10/11-21:25:52.964643 [**] [1:20110398:1] Known reverse shell command perl -MIO [**] [Priority: 0] {TCP} 10.10.10.2:58754 -> 10.10.10.3:33237
08/10/11-21:25:59.203671 [**] [1:20110395:1] Known Hostile distccd Command cd /boot [**] [Priority: 0] {TCP} 10.10.10.2:44044 -> 10.10.10.3:4444
08/10/11-21:26:54.719778 [**] [1:20110396:1] Known Hostile distccd Command passwd [**] [Priority: 0] {TCP} 10.10.10.2:44044 -> 10.10.10.3:4444
08/10/11-21:27:14.738138 [**] [1:20110397:1] Known Hostile distccd Command tcpdump [**] [Priority: 0] {TCP} 10.10.10.2:44044 -> 10.10.10.3:4444

The experiment is repeated and recorded as higher rates of packet loss are introduced.

B. *Random Packet Loss*

The packet loss in a network can be simulated in DummyNet. The ipfw argument plr X, where X is a floating-point number between 0 and 1 that causes packets to be dropped at random simulates packet loss, where 0 is for no loss and 1 is for 100%packet loss.

ipfw pipe 1 config plr 0.5

C. *Correlation of OSSEC and Network-based alerts*

This experiment shows the correlation of network-based and host-based alerts. The correlation of OSSEC alerts and Snort alerts was accomplished by altering the OSSEC server code to generate alerts in a format that can be processed by the network IDS collection sensor. The OSSEC/Snort formatted alert is time sequenced with the approximate time of the network alert and displayed on the web portal using a single correlated network and host based interface page. The network researcher views the OSSEC alert and Snort alerts together on a single web portal page.

OSSEC Generated Alert:

8011 Aug 10 17:23:52 Rule Id: 802 level: 10
Location: (tracker-desktop) 10.10.10.3->/var/log/syslog
distccd
Aug 10 17:38:26 tracker-desktop distccd[13968]: (dcc_check_client) connection from 10.10.10.2:58754

Converted to formatted alert:

08/10/11-21:25:52..960396 [**] [1:13968:1] OSSEC Alert (dcc_check_client) connection from 10.10.10.2:58754[**] [Priority:10] {TCP} 10.10.10.2:58754-> 10.10.10.3:33237

Corresponding Sensor Generated Snort Alert:

08/10/11-21:25:52.960396 [**] [1:20110394:1] distcc_exec connection warning [**] [Priority: 0] {TCP} 10.10.10.2:58754 -> 10.10.10.3:33237

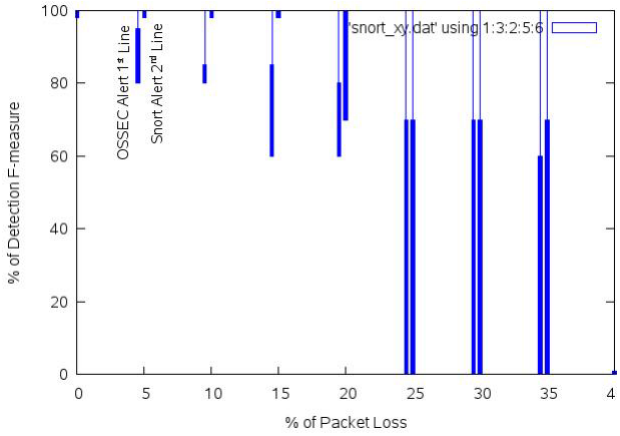

Figure 2: Results of correlating network and host based IDSs in the presence of packet loss.

Figure 2 exemplifies the results of correlating network and host based IDSs in the presence of packet loss. Packet loss rates greater than 20% are improved by combining the two detection rule sets of OSSEC and Snort and using the packets that reach the collection sensor more effectively. We observed from testing that with less than 20% packet loss that Snort accurately detected the intrusion. In this particular experiment, the higher detection rate was largely due the accuracy of the Snort rules and not network quality.

V. CONCLUSIONS

In this paper, we presented the design of a novel open network and host-based intrusion detection testbed (ONBIT) that is intended to facilitate a broad range of experimental research on the next-generation of network security applications and security vulnerabilities. We have also explained how to conduct a typical experiment and provided a sample experiment as a proof-of-concept validation of the testbed design. We were able to show how the experiment is conducted to maintain repeatability to enforce the validity of the architecture and design.

The testbed also provides researchers with the ability to conduct experiments using exploits and potentially vulnerable code on a private isolated network. For most experiments, remote access to the testbed is supported from the Internet. Network security measures, such as an IDS to monitor network activity, and a firewalled access point to the testbed has been deployed. When physical isolation for a particular experiment is required, the researcher can remove the cat-5 cable input from the Internet and conduct the experiment from the researcher's workstation.

A testbed is crucial in continuing cutting- edge wireless technology research and development since physical and MAC layers in mobile wireless communication have many parameters pertinent to performance that are often not tractable in analysis or simulation alone. The next generation of the ONBIT testbed would benefit in having a wireless MAC and physical layer-testing environment. The benefit would be two fold, we would gain insight into the world of wireless vulnerabilities and provide a testing and development environment for tablet and smart phone technologies. Physical layer wireless is required to accurately load applications onto these types of devices. We can emulate wireless devices, but testing is very limited. A proposed wireless testbed could be used to develop an understanding of wireless threats and help to expose new wireless vulnerabilities.

REFERENCES

[1] J. Barceló, C. Macáo, J. Infante, M. Oliver, and A. Sfairopoulou, "Barcelona's open access network testbed," in Proceedings of IEEE Tridentcom. IEEE, 2006.

[2] Terry Benzel, Bob Braden, Ted Faber, Jelena Mirkovic, Steve Schwab, Karen Sollins, and John Wroclawski. 2009. Current Developments in DETER Cybersecurity Testbed Technology. In Proceedings of the 2009 Cybersecurity Applications \& Technology Conference for Homeland Security (CATCH '09). IEEE Computer Society, Washington, DC, USA, 57-70.

[3] Shashi Guruprasad, Robert Ricci, and Jay Lepreau. 2005. Integrated Network Experimentation using Simulation and Emulation. In Proceedings of the First International Conference on Testbeds and Research Infrastructures for the DEvelopment of NeTworks and COMmunities (TRIDENTCOM '05). IEEE Computer Society, Washington, DC, USA, 204-212.

[4] A. Hay, D Cid, R. Bray, *OSSEC Host-Based Intrusion Detection Guide*. 2008, Syngress: Burlington.

[5] Horst Hellbrück, Max Pagel, Alexander Kröller, Daniel Bimschas, Dennis Pfisterer, Stefan Fischer: Using and operating wireless sensor network testbeds with WISEBED. Med-Hoc-Net 2011: 171-178.

[6] Nikola Knežević;, Simon Schubert, and Dejan Kostić;. 2010. Towards a cost-effective networking testbed. SIGOPS Oper. Syst. Rev. 43, 4 (January 2010), 66-71.

[7] Mahindra, R. , Bhanage, G. , Hadjichristofi, G. , Ganu, S. , Seskar, I. and Raychaudhuri, D. (2009) Architecture for federating heterogeneous networking testbeds. http://www.orbit-lab.org/raw-attachment/wiki/Orbit/Documentation/Publications/plab_orbit.pdf

[8] Luigi Rizzo. 1997. Dummynet: a simple approach to the evaluation of network protocols. SIGCOMM Comput. Commun. Rev. 27, 1 (January 1997), 31-41.

[9] Jang-Ping Sheu, Chia-Chi Chang, and Wei-Sheng Yang. 2010. A distributed Wireless Sensor Network testbed with energy consumption estimation. Int. J. Ad Hoc Ubiquitous Comput. 6, 2 (July 2010), 63-74.

[10] Geoffrey Werner-Allen, Patrick Swieskowski, and Matt Welsh. 2005. MoteLab: a wireless sensor network testbed. In Proceedings of the 4th international symposium on Information processing in sensor networks (IPSN '05). IEEE Press, Piscataway, NJ, USA, , Article 68 .

[11] DETER supported project list, http://www.isi.deterlab.net/projectlist.php3

[12] The Honeynet Project, http://www.honeynet.org

[13] The Honeynet Project, Know your Enemy, 2nd edition, Addison-Wesley Profeesional, May 2004.

[14] OSSEC is an Open Source Host-based Intrusion Detection System. http://www.ossec.net/main/