Global Information Assurance Certification Paper

Active Defense via a Labyrinth of Deception

GIAC (GCIA) Gold Certification

Author: Nathaniel Quist, nathanielquist1@gmail.com
Advisor: Adam Kliarsky
Accepted: November 19th, 2016

Abstract

A network baseline allows for the identification of malicious activity in real time. However, a baseline requires that every listed action is known and accounted, presenting a nearly impossible task in any production environment due to an ever-changing application footprint, system and application updates, changing project requirements, and not least of all, unpredictable user behaviors. Each obstacle presents a significant challenge in the development and maintenance of an accurate and false positive free network baseline. To surmount these hurdles, network architects need to design a network free from continuous change including, changing company requirements, untested systems or application updates, and the presence of unpredictable users. Creating a static, never-changing environment is the goal. However, this completely removes the functionality of a production network. Or does it? Within this paper, I will detail how this type of static environment, referred to as the Labyrinth, can be placed in front of a production environment and provide real time defensive measures against hostile and dispersed attacks, from both human actors and automated machines. I expect to prove the Labyrinth is capable of detecting changes in its environment in real time. It will provide a listing of dynamic defensive capabilities like identifying attacking IP addresses, rogue-process start commands, modifications to registry values, alterations in system memory and recording the movements of an attacker's tactics, techniques, and procedures. At the same time, the Labyrinth will add these values to block list, protecting the production network lying behind. Successful accomplishment of these goals will prove the viability and sustainability of a Labyrinth defending network (Revelle, 2011) environments.

1. Introduction

The Labyrinth is a virtual network housed within a hypervisor architecture. Hypervisors are often called Virtual Machine Monitors (VMMs) and allow one physical machine to manage the functionality of multiple Operating Systems (OSs). Don Revelle, the author of the whitepaper Hypervisors and Virtual Machines, states that for a single machine to operate multiple virtual OSs, "The hypervisor must work with minimal overhead and maintain supervisory privileges over the entire machine at all times" (Revelle, 2011). The Labyrinth manages the functionality of a network designed to perform in a similar manner to a production network. Each system within the Labyrinth has a role within the environment. Depending upon the available resources provided by the hypervisor, several subnets can be configured and maintained, each representing an aspect of an enterprise environment, a DMZ, an internal server bank, user workstations, or even entire departments like IT, or Human Resources departments. The joining of these separated networks into a single virtual medium creates the illusion of a complete enterprise environment.

The Labyrinth adds a hidden layer of security within this virtual network as a built in multifaceted security monitoring system that is employed to record every action and function on each system in the Labyrinth. Anchoring the security layer is a centralized correlation engine used to measure each action within the Labyrinth and compare these values against a listing of Known-Good actions. This listing is a static baseline of normal operations for each system, user, application, and subnet housed within the Labyrinth. CERN defines a baseline as "a set of basic objectives which must be met by any given service or system" (CERN Computer Security, 2010). Within the Labyrinth, the "basic objectives" are the actions of the individual systems, processes, and users, which must follow a prescribed list of values. Table 1, illustrates how baselines are compared against a known good set of values.

Nathaniel Quist, nathanielquist1@gmail.com

Table 1: Baseline: XOR Baseline Checksum

Matching Baselines			Non-Matching Baselines		
Known Good Value	Unknown Value	XOR Checksum	Known Good Value	Unknown Value	XOR Checksum
00001	00001	00001	00001	00001	00001
00010	00010	00010	00010	00010	00010
00011	00011	00011	00011	00011	00011
00100	00100	00100	00100	00101	00101
00101	00101	00101	00101	00101	00101
00110	00110	00110	00110	00110	00110
00111	00111	00111	00111	00111	00111
01000	01000	01000	01000	01111	01111
01001	01001	01001	01001	01001	01001
01010	01010	01010	01010	01010	01010

The advantage of leveraging static baselines against the virtual environment is the identification of any action performed outside of the static baseline. Since the Labyrinth is a static environment, each system, as well as each of its subsequent actions, should only function in a predictable manner. The correlation engine analyzes the values given from the live Labyrinth and compares these values against the Known-Good values listed within the static baseline. If there is a difference both the live Labyrinth and the baseline Labyrinth, it is undeniable evidence that a third party modified the Labyrinth.

2. SIEM Management versus Labyrinth Network

As mentioned within the previous section, the purpose of the Labyrinth is a security monitoring system, which directs every recorded action to a centralized correlation engine. The correlation engine is responsible for collecting network and system data and comparing it against a predetermined baseline to ensure Labyrinth validity. A centralized correlation engine appliance within the computer security industry is known as a Security Information Event Management (SIEM) tool. As illustrated within Figure 1, the SIEM receives data from several security tools like network security appliances providing IDS, Proxy, and NetFlow collection, as well as system specific event logging, process executions, registry, and memory alteration functionality.

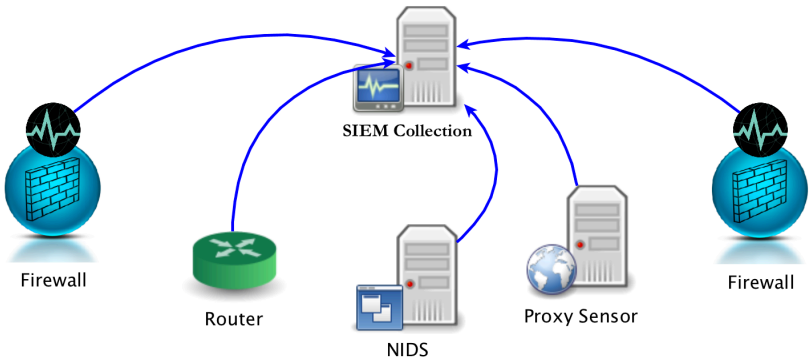

Figure 1: SIEM Topology

The network security layer targets network traffic, specifically Network Intrusion Detection Systems (NIDS), Network Traffic Proxy systems, and Network Flow Analysis. These technologies perform specific network traffic analysis and are essential in building a detailed picture of how an attacker may move and pivot within a network. The Labyrinth architecture should use a Snort integration along with firewall logs, and NetFlow sources to identify the specific types of traffic moving through the Labyrinth. The data from these systems is routed to the SIEM for further contextualization. Where the network data is correlated alongside log data, active directory information, and endpoint detection systems to develop a holistic picture of activity within the Labyrinth.

The security network provides the base architecture for the Labyrinth; as such, the Labyrinth lacks believability from the perspective of a functioning enterprise environment. The core aspects of the network, e.g. the firewalls, servers, endpoint systems, and network connections, still need to be installed and configured. While the security framework contains the structure for the continuity of network traffic, the role of the Labyrinth is designed to be a virtual recreation of a production environment. Honeypots have long been used to emulate live systems because they "simulate the characteristics and vulnerabilities of common operating systems and record all the operations and behaviors" (Du, Zhang, Zhou, & Bai, 2013). The Labyrinth network needs to provide features an attacker would expect from a standard enterprise or business network. The use of honeypots allows the defenders to deceive the attackers by creating the perception of a believable environment. This truly deceptive state will allow "a more active defense ability, learning ability, and dynamic interaction ability, than a traditional

defense system [alone]" (Du, Zhang, Zhou, & Bai, 2013). The Labyrinth takes this a step further by integrating not just one honeypot, but a series of honeypots into what is called a honeynet.

3. Labyrinth Traffic

3.1. Log and Traffic Collection

Detection of potentially malicious traffic begins at the system and device level within the Labyrinth. The placement of sensors throughout the Labyrinth in a variety of formats delivers a unique set of data to the SIEM. Traffic sensors like those employed by firewalls, the Intrusion Detection System (IDS), or the proxy systems, are commonplace within security network architecture. Firewalls provide a robust capability for displaying all inbound and outbound connections within an environment, as Figure 2 shows.

Time	Chain	Iface	Proto	Source Destination	Src Port Dst Port	Country	MAC Address
21:16:30	DROP_INPUT	red0	2	10.0.0.98 224.0.0.1			10:7b:ef:ba:53:48
21:16:15	DROP_NEWNOTSYN	red0	TCP	199.59.148.139 10.0.0.100	443(HTTPS) 51525		10:7b:ef:ba:53:48
21:15:44	DROP_NEWNOTSYN	red0	TCP	17.172.232.11 10.0.0.100	5223 51227		10:7b:ef:ba:53:48
21:15:35	FORWARDFW	green0	TCP	10.1.0.240 199.59.149.232	54950 443(HTTPS)		1a:fd:7b:8f:78:ed
21:15:32	FORWARDFW	green0	TCP	10.1.0.240 104.100.202.15	54961 80(HTTP)		1a:fd:7b:8f:78:ed
21:15:32	FORWARDFW	green0	TCP	10.1.0.240 104.100.202.15	54960 80(HTTP)		1a:fd:7b:8f:78:ed
21:15:32	FORWARDFW	green0	TCP	10.1.0.240 68.177.32.115	54959 80(HTTP)		1a:fd:7b:8f:78:ed

Figure 2: Firewall Connection Panel

Each of these sensors allows network traffic communication protocols and traffic patterns to identify malicious patterns within network traffic, providing the ability to detect malicious network traffic as it happens. The security industry considers IDS to be a dying security tool due to its relative weakness in deciphering between good and bad traffic:"The bane of IDS has been the inability to weed out false positives and false negatives" (Wickman, 2003). Given this currently accepted state, the Labyrinth does employ IDS sensors. The Labyrinth has already created a level baseline, and all action within the Labyrinth are judged by the SIEM, not by the IDS or web proxy devices themselves. IDS logs collected by the SIEM are also held against a baseline to determine legitimate or suspicious behavior.

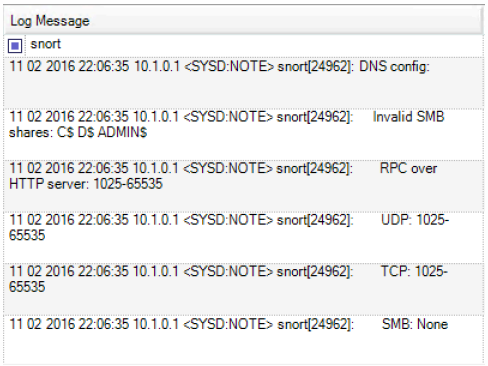

Figure 3: Snort logs displayed within a SIEM

As displayed in the Snort collection logs within Figure 3, the Labyrinth employs additional agent variants designed to capture a holistic threat detection spectrum. The term holistic is most commonly used within the medical industry but is defined "as relating to or concerned with complete systems rather than with individual parts" (Merriam-Webster, 2016). Windows event logs, syslog data, endpoint registry, command, and resource usage events, firewall traffic metadata, netflow traffic metadata, and third-party security appliance collection should all be collected, creating a board spectrum of information. The following graphics display a raw TCPDump traffic dump, Figure 4, which details network packets from and to a Linux system and Figure 5, displaying a Windows Security Log detailing a logon event.

```
20:25:36.070195 IP 46.246.124.91.http > 192.168.1.104.35978: Flags [S.], seq 2744952004, ack 83493226, win 651
60, options [mss 1452,sackOK,TS val 3253416854 ecr 8624004,nop,wscale 11], length 0
20:25:36.070276 IP 192.168.1.104.35978 > 46.246.124.91.http: Flags [.], ack 1, win 229, options [nop,nop,TS va
l 8624044 ecr 3253416854], length 0
```

Figure 4: TCPDump Example

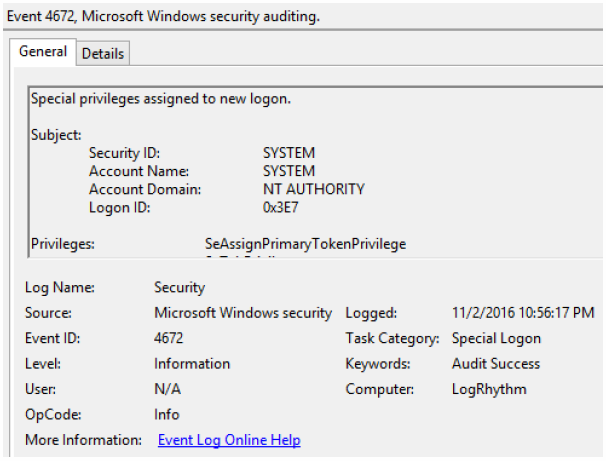

Figure 5: Windows Security Event Log

Nathaniel Quist, nathanielquist1@gmail.com

Each of these data sources should be collected directly from the endpoint systems on which they were created, and delivered to the SIEM for further analysis. Each system within the Labyrinth can give critical information benefiting the network at large. Each of these unique sources needs to be collected by Labyrinth sensors to form a complete picture of an incident. Within the SIEM, each of the unique logs is displayed within a single pane of glass, as seen in Figure 6. This method of viewing network events gives the analyst a deeper understanding of events within a holistic approach to security events and offers correlative capabilities based on specific event types.

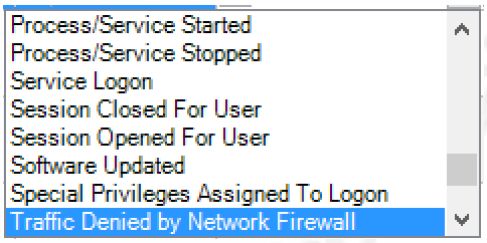

Figure 6: SIEM printout of holistic collection

If an event alters the events designed to take place within the Labyrinth, the SIEM's capabilities to correlate known good actions against the recorded live actions, allowing the SIEM to accurately identify any delta between the whitelist and the live data sets. Should a change be recorded signifying the alteration of the Labyrinth, the SIEM will document the difference between the two states, as illustrated within Figure 7. The delta is instantly written to disk by services in the SIEM, and this information can be used to block the continued actions from the same unknown patterns.

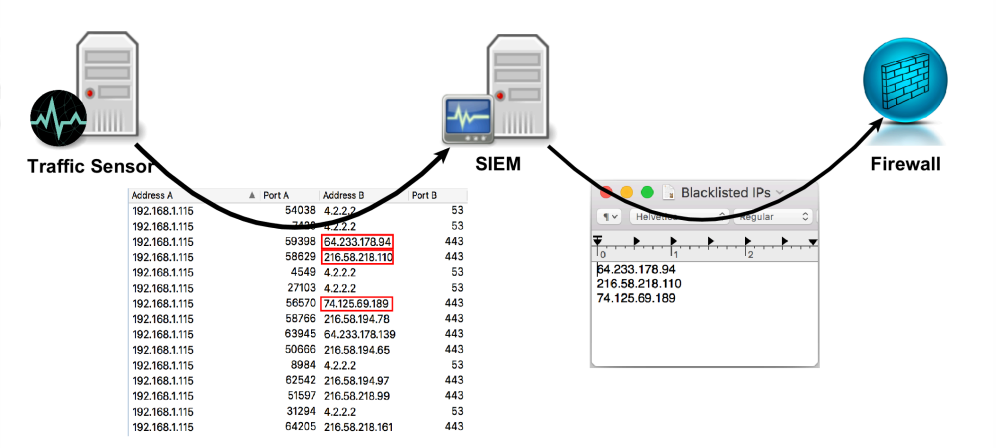

Figure 7: Traffic Detection to Blacklist

Human analysts will be forwarded all post-analyzed data, allowing them to quickly determine the exact anomalous action that took place in the Labyrinth, thus, affording the analyst time to prepare and defend the production environment rather than react to incidents as they occur.

3.1.1 Snort as a Log Source

Snort is an industry standard IDS platform used throughout the network security industry. The usage of Snort within the Labyrinth functions in the same manner as Snort within an enterprise network. Network defenders use Snort to monitor traffic transmissions across the virtual network and employ Snort functionality within a distributed and load balanced architecture. The physical location of Snort sensors is architected into the foundation of the Labyrinth itself, with a sensor located in each of the Labyrinth's subnets, as well as directly behind the Labyrinth's boundary firewall. Each sensor will relay suspect network events to a dedicated Snort manager, which in turn confirms or denies the event and ultimately delivers a positive IDS alert to the Labyrinth's SIEM appliance for further correlation.

The configuration of a Snort platform follows industry standard guidelines. This example of Snort was created on a Linux Debian 7.0 system and used the command 'sudo apt-get install snort' to install the tool. See Figure 8.

```
root@Snort:~# apt-get install snort
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following package was automatically installed and is no longer required:
  python3-urllib3
Use 'apt-get autoremove' to remove it.
The following extra packages will be installed:
  libdaq2 oinkmaster snort-common snort-common-libraries snort-rules-default
Suggested packages:
  snort-doc
The following NEW packages will be installed:
  libdaq2 oinkmaster snort snort-common snort-common-libraries
  snort-rules-default
0 upgraded, 6 newly installed, 0 to remove and 0 not upgraded.
Need to get 1,347 kB of archives.
After this operation, 7,090 kB of additional disk space will be used.
Do you want to continue? [Y/n]
```

Figure 8: Snort Installation

Snort requires configuration for the environment to perform efficiently. The following configurations assist Snort in the task: the home network IP address range, known as HOME_NET, the Snort rule directories, as well as the Snort decoders, and Snort Preprocessors. The configuration of these settings is within the snort.conf file, located in the /etc/Snort directory. Figure 9 presents a graphic display of the Snort configuration file, beginning at the top of the snort.conf file. All configuration changes to the snort.conf file must be performed using Linux's 'nano' or 'vi' editing tools.

```
#--------------------------------------------------
#   VRT Rule Packages Snort.conf
#
#   For more information visit us at:
#     http://www.snort.org                   Snort Website
#     http://vrt-blog.snort.org/    Sourcefire VRT Blog
#
#     Mailing list Contact:      snort-sigs@lists.sourceforge.net
#     False Positive reports:    fp@sourcefire.com
#     Snort bugs:                bugs@snort.org
#
#     Compatible with Snort Versions:
#     VERSIONS : 2.9.6.0
#
#     Snort build options:
#     OPTIONS : --enable-gre --enable-mpls --enable-targetbased --enable-ppm --$
#
#     Additional information:
#     This configuration file enables active response, to run snort in
#     test mode -T you are required to supply an interface -i <interface>
#     or test mode will fail to fully validate the configuration and
#     exit with a FATAL error
#--------------------------------------------------
```

Figure 9: Snort Configuration File

Figure 10 displays the use of the editor to add the Snort Home Network under section 1 of the snort.conf file.

```
# Setup the network addresses you are protecting
#
# Note to Debian users: this value is overriden when starting
# up the Snort daemon through the init.d script by the
# value of DEBIAN_SNORT_HOME_NET s defined in the
# /etc/snort/snort.debian.conf configuration file
#
ipvar HOME_NET 192.168.1.0/24

# Set up the external network addresses. Leave as "any" in most situations
ipvar EXTERNAL_NET any
# If HOME_NET is defined as something other than "any", alternative, you can
# use this definition if you do not want to detect attacks from your internal
# IP addresses:
ipvar EXTERNAL_NET !$HOME_NET
```

Figure 10: Snort Home/External Network Configuration

Configuration of the path location for Snort Rules is seen within Figure 11. This is information is also found within section 1 of the snort.conf file.

```
# Path to your rules files (this can be a relative path)
# Note for Windows users:  You are advised to make this an absolute path,
# such as:  c:\snort\rules
var RULE_PATH /etc/snort/rules
var SO_RULE_PATH /etc/snort/so_rules
var PREPROC_RULE_PATH /etc/snort/preproc_rules
```

Figure 11: Configuring Snort's Rule Directory

Figure 12 illustrates the configuration of decoders used by Snort is performed within section 2 of the snort.conf file.

```
##################################################
# Step #2: Configure the decoder.  For more information, see README.decode
##################################################

# Stop generic decode events:
config disable_decode_alerts

# Stop Alerts on experimental TCP options
config disable_tcpopt_experimental_alerts

# Stop Alerts on obsolete TCP options
config disable_tcpopt_obsolete_alerts

# Stop Alerts on T/TCP alerts
config disable_tcpopt_ttcp_alerts

# Stop Alerts on all other TCPOption type events:
config disable_tcpopt_alerts

# Stop Alerts on invalid ip options
config disable_ipopt_alerts
```

Figure 12: Assigning Snort Decoders

The configuration of each preprocessor implemented by Snort is seen within Figure 13. Configuration of the preprocessors is located under section 4 of the snort.conf file.

```
##################################################
# Step #4: Configure dynamic loaded libraries.
# For more information, see Snort Manual, Configuring Snort - Dynamic Modules
##################################################

# path to dynamic preprocessor libraries
dynamicpreprocessor directory /usr/lib/snort_dynamicpreprocessor/

# path to base preprocessor engine
dynamicengine /usr/lib/snort_dynamicengine/libsf_engine.so

# path to dynamic rules libraries
dynamicdetection directory /usr/lib/snort_dynamicrules
```

Figure 13: Configure Snort's Preprocessors

Once the Snort decoders and preprocessors are configured, enabling the syslog feature will allow for the passage of detected events to flow from Snort to the Labyrinth's SIEM. Located within section 6, illustrated in Figure 14, the addition of string ‘output alert_syslog: host:<SIEM IP>:514, LOG_AUTH LOG_ALERT' directly below the ‘#syslog' header forces Snort to pass all its events to the SIEM via syslog.

```
##################################################
# Step #6: Configure output plugins
# For more information, see Snort Manual, Configuring Snort - Output Modules
##################################################

# unified2
# Recommended for most installs
# output unified2: filename merged.log, limit 128, nostamp, mpls_event_types, v$
output unified2: filename snort.log, limit 128, nostamp, mpls_event_types, vlan$

# Additional configuration for specific types of installs
# output alert_unified2: filename snort.alert, limit 128, nostamp
# output log_unified2: filename snort.log, limit 128, nostamp

# syslog
output alert_syslog: host:192.168.1.235:514, LOG_AUTH LOG_ALERT

# pcap
# output log_tcpdump: tcpdump.log

# metadata reference data.  do not modify these lines
include classification.config
include reference.config
```

Figure 14: Configuring Snort's Syslog Feed

The base configuration of Snort does not include a full set of Snort signatures. Signatures are used by Snort to trigger alerts when malicious traffic passes across a network. The configuration of Snort includes custom Snort signatures used for highly configured settings, or the use of pre-generated signatures typically used in generic environments. For this example, I will demonstrate the process of configuring the Snort signatures using pre-generated rules pulled from the "Registered" version of Snort 2.9. The "Registered" version of Snort Rules v2.9 is an open source list of signatures, and it is available upon approval from Snort.org. This version does require an active membership in which to download the signatures. There is a community version containing an open source collection of Snort signatures, available to any user who wishes to stand up a Snort environment without requiring registration through the Snort website. There is also an additional Snort signature package, called "Subscription" signatures. Subscription signatures are only available through the purchase of this specific package. "Subscription" signature rule sets traditionally contain more targeted sets of signatures, focused up trending threats within the security industry, where the free packages such as the "Community" or the "Registered" signature sets contain only a basic listing of signatures.

Snort v3.0 does contain a listing of Community signature rules by default. However, to install the "Registered" or "Subscription" version of Snort signatures you will need to have a registration code, called an OinkCode, to download specific signatures. To download these signature rules, the user can directly download the rules via a terminal session from the Snort appliance using the following wget command. See Figure 15.

```
root@Snort:/etc/snort# wget https://www.snort.org/rules/snortrules-snapshot-2983.tar.gz?oinkcode=XXXXXXXX
```

Figure 15: wget call for Registered Snort Signatures

After the user has downloaded the rule set, the rules need to be added to the /etc/snort/rules directory, so they are available to Snort. The command to uncompress and move the rules is displayed within Figure 16.

```
root@Snort:/etc/snort# tar -xvfz snortrules-snapshot-2983.tar.gz -C /etc/snort/rules
```

Figure 16: Installing Snort's Registered Signatures

Now that Snort is configured, tuned, and has a complete set of signatures, the user can begin the process of starting Snort. Snort requires the use of the'–s' switch within the Snort start-up command to relay Snort Alerts via syslog to the waiting SIEM appliance. Figure 17 demonstrates what Snort looks like after supplying the Snort start-up command, 'snort –c snort.conf –s –A fast.'

```
        --== Initialization Complete ==--

   ,,_     -*> Snort! <*-
  o"  )~   Version 2.9.6.0 GRE (Build 47)
   ''''    By Martin Roesch & The Snort Team: http://www.snort.org/snort/snort-t
eam
           Copyright (C) 2014 Cisco and/or its affiliates. All rights reserved.
           Copyright (C) 1998-2013 Sourcefire, Inc., et al.
           Using libpcap version 1.5.3
           Using PCRE version: 8.31 2012-07-06
           Using ZLIB version: 1.2.8

           Rules Engine: SF_SNORT_DETECTION_ENGINE  Version 2.1  <Build 1>
           Preprocessor Object: SF_MODBUS  Version 1.1  <Build 1>
           Preprocessor Object: SF_DNP3  Version 1.1  <Build 1>
           Preprocessor Object: SF_DCERPC2  Version 1.0  <Build 3>
           Preprocessor Object: SF_POP  Version 1.0  <Build 1>
           Preprocessor Object: SF_SSH  Version 1.1  <Build 3>
           Preprocessor Object: SF_SSLPP  Version 1.1  <Build 4>
           Preprocessor Object: SF_SMTP  Version 1.1  <Build 9>
           Preprocessor Object: SF_SDF  Version 1.1  <Build 1>
           Preprocessor Object: SF_REPUTATION  Version 1.1  <Build 1>
           Preprocessor Object: SF_DNS  Version 1.1  <Build 4>
           Preprocessor Object: SF_SIP  Version 1.1  <Build 1>
           Preprocessor Object: SF_GTP  Version 1.1  <Build 1>
           Preprocessor Object: SF_IMAP  Version 1.0  <Build 1>
           Preprocessor Object: SF_FTPTELNET  Version 1.2  <Build 13>
Commencing packet processing (pid=38112)
```

Figure 17: Running Snort

Upon seeing this screen, Snort will not display further messages in the terminal screen, as all Snort events are written via syslog and sent to the Labyrinth's SIEM. Snort messages are used as another layer of monitoring and detection, providing network data to the SIEM. Correlating this data against Known-Good baseline network data, the SIEM can identify acceptable and unacceptable connections inside the Labyrinth. Figure 18 details a Snort event from Labyrinth system, 192.168.1.168, attempting to perform and RDP connection to another Labyrinth system, 192.168.1.235.

```
root@Snort:/var/log/snort# cat alert | grep MS
TCP Options (8) => MSS: 1460 NOP WS: 5 NOP NOP TS: 597877841 0 SackOK EOL
TCP Options (8) => MSS: 1460 NOP WS: 5 NOP NOP TS: 597877850 0 SackOK EOL
11/10-20:14:10.170777  [**] [1:1448:12] MISC MS Terminal server request [**] [Cl
assification: Generic Protocol Command Decode] [Priority: 3] {TCP} 192.168.1.168
:56076 -> 192.168.1.235:3389
11/10-20:17:04.013695  [**] [1:1448:12] MISC MS Terminal server request [**] [Cl
assification: Generic Protocol Command Decode] [Priority: 3] {TCP} 192.168.1.168
:56149 -> 192.168.1.235:3389
root@Snort:/var/log/snort#
```

Figure 18: Snort example of an RDP attempt

Since Snort is sending data to the SIEM, defenders can also witness the same event within the SIEM. See Figure 19. In this case, Snort had initially flagged the event as suspicious, and the SIEM has maintained that same correlation.

Log Source Type	Classification	Common Event
Flat File - Snort Fast Alert..	Reconnaissance	Ping Sweep
Flat File - Snort Fast Alert..	Reconnaissance	Ping Sweep
Flat File - Snort Fast Alert..	Reconnaissance	Ping Sweep
Flat File - Snort Fast Alert..	Reconnaissance	Port Scan
Flat File - Snort Fast Alert..	Activity	ICMP Echo Reply
Flat File - Snort Fast Alert..	Reconnaissance	Port Scan
Flat File - Snort Fast Alert..	Reconnaissance	Port Scan
Flat File - Snort Fast Alert..	Reconnaissance	Port Scan
Flat File - Snort Fast Alert..	Suspicious	Suspicious Activity
Flat File - Snort Fast Alert..	Activity	ICMP: Port Unreachable
Flat File - Snort Fast Alert..	Reconnaissance	Port Scan
Flat File - Snort Fast Alert..	Reconnaissance	Port Scan
Flat File - Snort Fast Alert..	Activity	ICMP: Port Unreachable

Figure 19: Listing of all Snort events within the SIEM

Diving further, it is inferred that the SIEM is capable of displaying both the raw Snort log itself, as well as any accompanying metadata values associated with such an event. Within Figure 20, the raw data is displayed, while Figure 21 shows the metadata that is associated with the collected raw data.

Log Source	Log Message	Classification	Common Event
		=	sus
Snort Snort Fast Alert Log	11/10-20:14:10.170777 [**] [1:1448:12] MISC MS Terminal server request [**] [Classification: Generic Protocol Command Decode] [Priority: 3] {TCP} 192.168.1.168:56076 -> 192.168.1.235:3389	Suspicious	Suspicious Activity

Figure 20: Drill down on suspicious Snort event

Field	Value
Vendor Message ID	1448
Entity (Origin)	
Entity (Impacted)	
IP Address (Origin)	192.168.1.168
Known Host (Impacted)	
IP Address (Impacted)	192.168.1.235
Hostname (Impacted)	
Known Application	RDP - Remote Desktop Protocol
TCP/UDP Port (Origin)	56076
TCP/UDP Port (Impacted)	3389
Protocol	TCP
Object	MISC MS Terminal server request

Figure 21: Listing of all known metadata from the Snort event

The power of integrating Snort data into the Labyrinth is that each packet within the Labyrinth can be collected and shipped to the centralized SIEM where it is integrated with data from other systems and network devices. The SIEM performs correlation against the Known-Good network traffic from the live network traffic, providing immediate understanding as to whether the Labyrinth itself generated that event, or if a third party performed the action, indicating malicious intent. If this example were live, the SIEM would flag the originating IP address, 192.168.1.168, and flag all traffic from that system as suspicious. The SIEM could also perform several automated actions against that system, such as capturing a live snapshot of memory to be used for forensic investigation, isolating and extracting a currently running process, and generating a list of user accounts currently logged into that system.

3.2. Firewalls

As in production environments, firewalls hold an important role within the Labyrinth. The Labyrinth assumes the role as the boundary firewall, serving as both an external and internal boundary for traffic entering and leaving the production network. Like edge firewalls within a production environment, the Labyrinth's firewall is subject to large quantities of connection attempts from the Internet like network or system vulnerability probing, potential large-scale denial of service attacks, and malformed packet attacks. Each attempt is designed to either compromise, overwhelm, or penetrate network boundary defenses. These types of events bombard external firewalls with

verbose quantities of data and consistently test the network's firewall defensive capabilities. As Figure 22 shows, the Labyrinth is placed in the direct line of attack from these types of attacks and events.

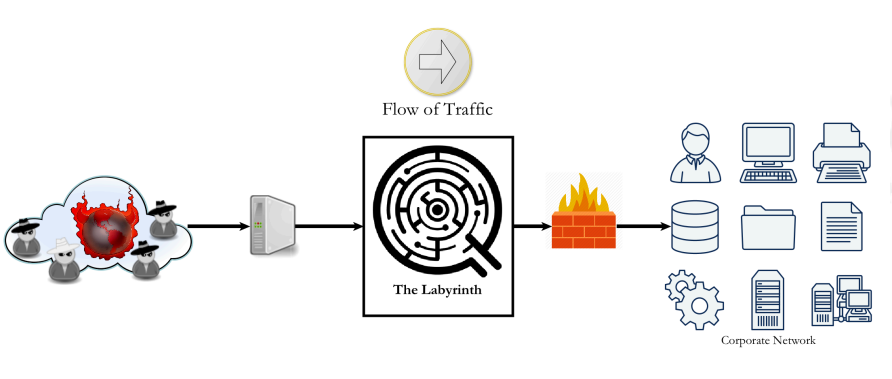

Figure 22: Labyrinth Placement

The Labyrinth is not immune to these malicious patterns of traffic, and precautions need to be taken within the Labyrinth to ensure the configuration of the firewall is conducive to boundary firewall functionality. Standard industry practices for firewall administration and maintenance are requirements even within the Labyrinth. Requirements for these two separate firewalls include that industry standard firewall rule configurations must be configured, and that a distinct separation between true boundary firewall and the Labyrinth's boundary firewall must be employed.

A network containing a Labyrinth technically consists of two boundary firewalls. A firewall housed within the Labyrinth, and the external firewall situated in front of the production network. The Labyrinth's firewall essentially mimics the functionality of the external firewall, and the Labyrinth's firewall should not be trusted to the same extent nor given the same firewall rules as the boundary firewall. The Labyrinth firewall's design must allow for resetting, alteration or modification on a dynamic basis. This requirement forces the Labyrinth firewalls to protect the Labyrinth while not making any allowances for the production network. The Labyrinth's virtual landscape is designed to be re-deployed at a moment's notice, if the Labyrinth's firewall contained custom firewall rules for the production network, those rules would be removed when the Labyrinth was re-deployed. Additionally, if the Labyrinth firewall is configured with the same rules as the

true boundary firewall, attackers could glean information detailing the layout of the production environment, or at least the types of services offered within the network.

The Labyrinth contains several internal firewalls, which behave in the same manner as network segment firewalls within a production network. Segment firewalls are the traditional method for elevating a flat network topology into a multilayered topology. The Labyrinth should mimic a production network in almost every facet. By using a tiered topology, the Labyrinth maintains its deception, and "successfully causing the target to accept as true, a specific incorrect version of reality, with the intent of causing the target to act in a way that benefits the deceiver" (Rowe & Custy, 2007). Successful network deception will make the Labyrinth more believable to attackers and will assist defenders in the identification and analysis of an attacker's actions. Figure 23 illustrates the flow of traffic through the Labyrinth, as data traverses each of the Labyrinth's subnets before continuing its passage to the true boundary firewall.

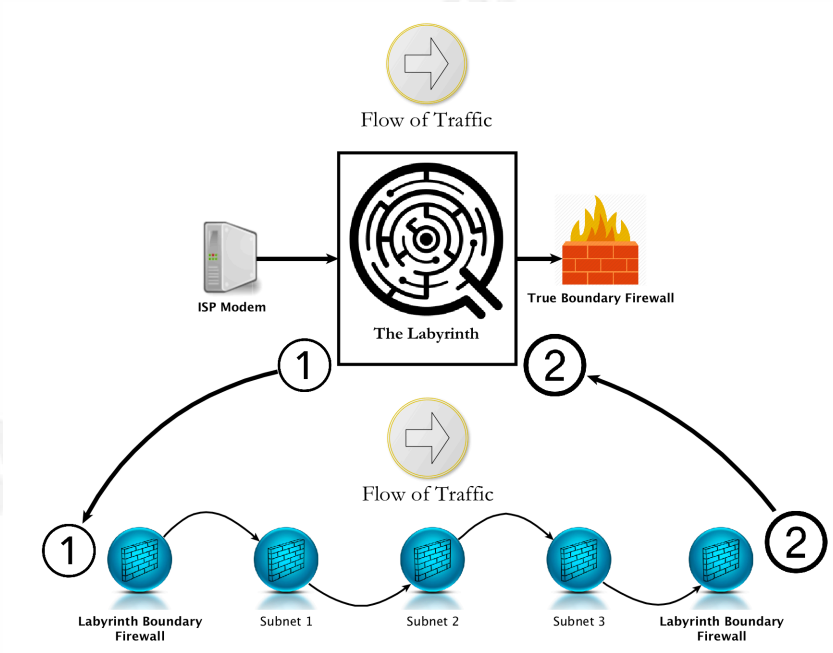

Figure 23: Firewall Data Flow

The internal firewalls within the Labyrinth are not expected to process large volumes of traffic. Only the virtual systems of the Labyrinth will be producing traffic, and in a static Labyrinth, this will be the only traffic crossing the network boundary. As such, these firewalls should not be the targets of heavy network traffic or the recipients of

a sustained bombardment of attacks. If the firewalls do experience this style of traffic, the Labyrinth will detect the event. The internal firewalls will not require a significant amount of resources to provide functionality and virtual firewall applications like those provided by the open-source providers, pfSense, IPFire, and OPNsense, provide many of the features required to process and detect any suspicious action.

As stated above, the production network still maintains its boundary firewall, which employs the originally designed function of protecting the internal network. Even with the presence of the Labyrinth, the boundary firewall still maintains the functionality by blocking external communications, which could find a way through the Labyrinth. Due to the presence of the Labyrinth, the quantity of data the boundary firewall must now process should now be much smaller due to the Labyrinth detecting and blocking the vast majority of unwarranted communications.

The Labyrinth also provides the boundary firewall an extra set of eyes which detect and prevent unwarranted activity. Due to the Labyrinth's heavily censored and static environment, it can quickly and accurately uncover potentially malicious connections and forward any identified external IP addresses to the production boundary firewall for blocking. Any communication to or from that same external IP address will instantly be flagged as suspicious and blocked from communication for all production endpoints. In essence, the boundary firewall is given an early warning detection system via the Labyrinth, allowing for the real-time blocking of malicious traffic before endpoint systems connect to the suspicious external systems.

3.3. Attaching Third Party Appliances

The design of the Labyrinth contains several segmented networks used to alter the network topology and aid in the believability of the Labyrinth. These network segments provide defenders an opportunity to increase their data analytic surface area by connecting physical third-party security appliances. Third party security appliances can provide additional insight into both suspicious and legitimate network traffic traversing the Labyrinth by performing both inline and tapped network analytics pulled directly from the Labyrinth's own network choke points. These devices include Anti-Virus detection sensors, web proxy sensors, deep packet analytic devices, as well as Intrusion

Detection and Prevention Systems. The appliances use the Labyrinth's structured network as an intermediate monitoring zone, analyzing data before any suspicious traffic can pass through into the production network or exit out of the Labyrinth environment.

This integration with the Labyrinth enhances the functionality of these appliances by providing a single routable pathway in which to collect data, and assists with the management of the appliances' resources in a more structured and efficient manner. This efficiency is achieved by using physical NICs to pass network traffic out of the Labyrinth's hypervisor architecture to physical systems outside of the Labyrinth. The Labyrinth reduces its strain on performing constant network analysis on internal network traffic and allows for the security appliances to maintain a more granular analysis of network traffic entering and leaving the production environment. The diversion of traffic out of the Labyrinth is architected into the Labyrinth's network structure between the virtual network segments. The segments within the Labyrinth provide an avenue for network traffic to be routed out of the Labyrinth to the designated appliances. The network traffic pulled from the Labyrinth is then passed to the specified security appliance where the appliance is able to perform the designed functionality. Upon completion of the analysis, the security appliance sends the traffic back into the Labyrinth to complete its journey. This process is seen in Figure 24:

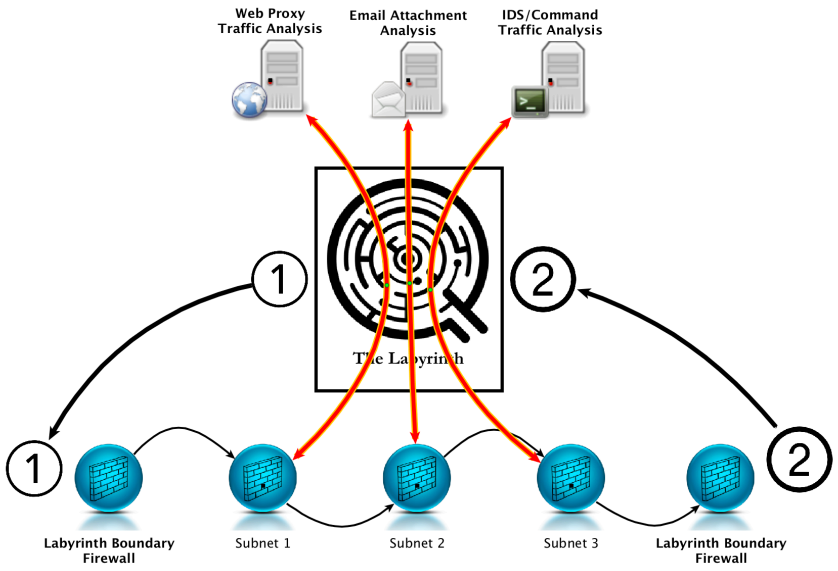

Figure 24: Connection of Third Party Security Appliances

The capability of the Labyrinth to allow third-party appliances to receive traffic from and return traffic to the Labyrinth dramatically increases the security capabilities of the production network. By allowing for a technique called “distributed scanning" the Labyrinth saves resources and distributes the network’s monitoring load to several third-party systems. The offensive equivalent of distributed scanning was explained within the whitepaper “Tracking Darkports for Network Defense” as, “scanning that occurs when multiple systems act in unison using a divide and conquer strategy to scan a network or host of interest” (Whyte, van Oorschot, & Kranaki, 2007). To use this same tactic as a defensive technique, the Labyrinth provides an avenue to perform a higher density of analytic operations, and a means for each packet crossing the Labyrinth to undergo multiple passes across several security sensors, the Labyrinth can bolster the detection capabilities against all traffic moving through the Labyrinth. Security appliances and applications can achieve a more granular analysis of all network traffic by segmenting the analytic requests and by allowing multiple stages of detection from the same analytic system across the length of the Labyrinth.

It may be helpful to think of the style of defense provided by the Labyrinth being akin to the gatehouse within medieval castles. The gatehouse contained two doorways on either side of an enclosed hallway: the entrance being the castle gates and the exit being the portcullis. In the hallway between the two doorways was a long inspection point in which all persons and cargo awaited inspection before access to, or withdraw from, the castle. These hallways were holding zones used to inspect persons and cargo before allowed to continue the journey. Should something be suspected, the portcullis would drop, and the castle gates would close trapping whoever was inside the enclosed hallway. The same concept appears today in data centers and with their use of mantraps. All persons are required to successfully meet the authentication requirements before being allowed to enter or leave the data center. This concept is moved into the realm of the network as the Labyrinth behaves like the mantrap, or rather the castle gatehouse, inspecting, questioning, and authenticating all activities taking place within its boundaries. While it is impossible to prevent and stop all malicious attempts against a network, the Labyrinth provides the gatehouse architecture required to give every packet,

connection, session, and byte, an additional layer of inspection to ensure that best effort analysis of all data occurs.

4. Test Labyrinth

The purpose of the Labyrinth is to detect anomalous activity while still maintaining a deceptive trait of being a fully functional production network. Every action within the Labyrinth is weighed and measured against a baseline of the same type, comparing IP addresses against a legitimate listing of IP addresses, system processes weighed against a known good process list, and user actions weighed against the known actions of Labyrinth user accounts. The whitelists used as the control list are taken from the legitimate actions themselves before the Labyrinth has "gone live." Otherwise stated as all actions and baselines are taken from a Labyrinth before placed in its final position, the listing of IP addresses, processes, system and networking behaviors, and user activities represent all accepted and normal behavior for that entity. Gordon Fraser, author of 'Creating a baseline of process activity for memory forensics,' explained the procedure for creating a process baseline involves, "needing a basic understanding of the core processes loaded by the operating system upon boot and the processes that are loaded when a user logs on" (Fraser, 2014). The recording of minutia processed by a system as a service starts presents the Labyrinth with the core foundation of the process baseline. This data then becomes available to compare future service starting operations to ensure the service executed as designed or if it had been modified to fulfill a separate purpose.

To achieve the truly dynamic deception qualities, yet still maintain quantitative dataset requirements, all network traffic within the Labyrinth data is required to be pre-recorded and looped to ensure continuous action. The Labyrinth recording needs to be recorded in a way so that it prevents the direct perception of recorded traffic. Imagine viewing a looped animated image, if the first sequence of frames and the last sequence of frames do not perfectly align the video will appear to 'jump.' This same effect should be taken into account when a system or a user account refreshes their looped action. One technique employed to hide the jump effect within the looped action is to stagger the recorded tracks. By beginning each recorded endpoint system and each user account on a

separate cycle, the obscurity of network traffic will provide cover for a recording which contains a jumping event

The actions of the Labyrinth itself, those being the systems, users, and processes, will not need an external stimulus to produce believable production value. Internal file servers will be called upon by virtual user accounts to deliver specified virtual data from requested virtual systems within the virtual network. Data will be deleted, created, and modified on all Labyrinth systems as it normally would within a standard environment. To ensure the deception of the Labyrinth, creating external requests should be made from specified Labyrinth systems to aid in the illusion should anyone witness the network traffic within the Labyrinth. Seeing external web traffic emanating from the Labyrinth itself will assist in maintaining the illusion. Recording and then replaying live collected network traffic from a production environment can assist in creating a more believable network profile.

5. Labyrinth Analytics

The analytics process within the Labyrinth falls into three categories: data collection, baseline comparison, and block list creation. These three categories make up the key aspects for how the Labyrinth will determine malicious actions. Figure 25 illustrates this process. The log sources generate logs, which are collected and shipped to the SIEM. Inside the SIEM, a baseline comparison is performed against the live data to ensure validity. Finally, actions that do not match the baseline will be flagged and written to block lists. The SIEM sends these block lists to the appropriate security appliance responsible for blocking the suspicious traffic within the production environment.

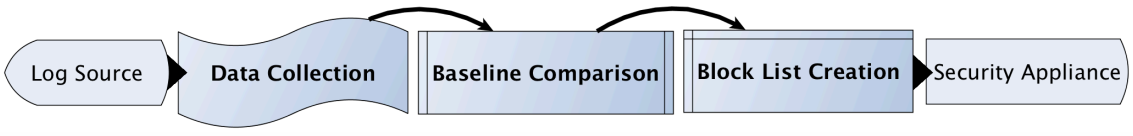

Figure 25: Labyrinth Process Flow

A series of network sensors and data collectors provide data collection through each endpoint system and network device within the Labyrinth. The hypervisor housing the Labyrinth is also capable of collecting data from the Labyrinth systems and shipping

these logs to the SIEM for further analysis. Endpoint systems will collect registry, process, and command-line activities; network devices will collect network traffic, session creations, and connection activities; and the hypervisor collects memory and hard drive resource requests from all virtual system resource requests.

The SIEM compares each activity against a listing of known good actions for that entity category to determine the validity of the action. For example, comparing all current external IP address connections against the listing of known good external IP addresses connections. All collected entity fields follow the same procedure, system processes, services, user command-line and network activities, system registry alterations, application changes or configurations, and network flow statistics, among others. Any action performed within the Labyrinth, which does not contain a correlating entry on the accompanied whitelist, will be written to a blacklist. The blacklist will then have its contents delivered to their respective security appliance, via a scheduled task. The scheduled task is programmed to monitor for changes in the blacklist file. Upon recording a change, the scheduled task will initiate a script performing the action of delivering the contents to the appropriate security appliance.

The following use-case will assist in describing the functionality of the Labyrinths analytics. A malicious actor scans a suspected production environment. They discover a susceptible endpoint. Upon successful exploitation of the endpoint, the actor downloads their Remote Access Tool (RAT) onto the exploited system, successfully hooking the system for continued operations into the network. Given these details, the following pieces of metadata would be collected by the Labyrinth and sent to the internal SIEM.

1. The external IP address(es) performing the reconnaissance scan(s).
2. The external IP address launching the exploit against the internal system.
3. The exploit used to compromise the endpoint.
4. Compromised user accounts used before, during, and after the exploitation.
5. The external system housing the actor's toolkit.
6. The RAT itself, containing the following pieces of metadata:
    a) The malware name.
    b) The malware's hash values: MD5, SHA1, and SHA256.

c) System registry modifications and configurations.
d) Dropper files created by the RAT.
e) Beacon data sent from the RAT to the Command and Control (C2) node.
f) The IP address(es) of the C2 node(s).
g) Specific beacon patterns of the RAT.

7. All subsequent actions by the actors after the placement of the RAT to be identified following these same steps listed above, across other endpoints and network devices within the Labyrinth.

With each attack against the network, specific metadata values can be gathered to identify the action. The Labyrinth allows for each of these values to be analyzed by the SIEM. The Labyrinth then delivers each value to an appropriate security appliance, designed to prevent internal production systems from falling victim to the same attack, which affected the Labyrinth. Following an attack of this nature, the Labyrinth can revert to a golden image. Meaning the Labyrinth is wiped clean of all malicious activities and reverted to a state free from the compromise that just encountered. All objects recovered from the compromise are still on record and activity being blocked by production security appliances, thus leaving the Labyrinth able to detect and identify the next attack on the network.

6. Labyrinth Limitations

The Labyrinth offers many beneficial features for detecting unknown activity by providing indicators of compromise to production security systems. However, the Labyrinth is not fully capable of protecting every aspect of the production network. Perhaps one of the greatest threats facing businesses today is compromised employee accounts. Employee account traffic is, by default, allowed to pass through the Labyrinth unhampered. The function of the Labyrinth is to detect direct action against the static environment, not to analyze traffic leaving the production environment. The Labyrinth does offer the capability to integrate with third-party security appliances to provide a deeper insight into the analysis of legitimate traffic passing across the Labyrinth threshold. The process of analyzing raw network traffic will remain in the same state as it

currently stands within traditional networks, and is subject to the speed and viability of the security products currently performing those actions.

To expound upon how traffic can be sent undetected through the Labyrinth, the Labyrinth functions in the same manner as an NAT'ed environment, meaning that traffic originating from an internal location will have its origin IP address removed from the packet and replaced with an IP address designated by the NAT firewall. This traffic is allowed to cross an untrusted network without fear of exposing the systems or networks behind the firewall. The firewall will then await the return traffic from the packet's original destination. When presented with the return traffic, the firewall compares the packet session IDs, namely the packet's ACK and SYN values. These values are unique to the return packet and will only be accepted by the firewall if they match. If the packets do meet the firewall's criteria, the firewall replaces the original the IP address given to the original outgoing packets with the original source IP address and the return packet is allowed to enter the internal network.

The Labyrinth does not inspect the traffic originating from internal systems, and, as such, is not able to perform baseline operations against this traffic. The door is left open for potentially malicious traffic targeting user accounts to pass through the Labyrinth as if it requested by a legitimate means. Examples of these types of user-focused attacks are, SpearPhishing, a legitimate user clicking on a malicious link from a received email, solutions regarding how to mitigate these attacks range from email and web filtering, payload analysis, Phishing awareness training, and network traffic analysis (Phishlabs, 2015). Waterhole attacks are another style of attack not inherently being blocked by the Labyrinth. Waterhole attacks happen when, "The attacker injects the malicious code by downloading it into the client system, once the client requests for a service from the web server" (Sarala, Kayalvizhi, & Zayaraz, 2014). Web proxy analysis tools would be a more logical choice to prevent this style of attack. Poor downloading habits are another user-based behavior which the Labyrinth will not be able to prevent. Proper user training and vigilant endpoint security monitoring tools would be more suitable tools in the prevention of malicious file downloads. Finally, physical access to user systems from a malicious actor is also not preventable by the Labyrinth, as the

Labyrinth cannot be knowledgeable of the actions from a user account performed by someone other than the account owner.

7. Conclusion

In review, the Labyrinth has the advantage of leveraging a static baseline analysis of pre-recorded actions to determine unwarranted actions against a network. The creation of baseline comes from each Labyrinth user action, process execution, and network connection through a series of scripted actions. Each system within the Labyrinth is a unique data source and feeds data to an internal SIEM. The SIEM compares all collected traffic against a baseline and determines the validity of the action based upon the successful matching of events.

The Labyrinth allows for the integration of third-party security devices to provide a more granular analysis of legitimate ingress and egress traffic. The analytic processes performed within Labyrinth are subsequently reduced due to the integration, leaving additional resources available to the Labyrinth's SIEM to perform more granular baseline comparisons across its environment. The SIEM further reduces analytic resource usage by not having to perform advanced User Behavior Analytics or Network Behavior Analytics, as the SIEM only requires the usage of baseline comparison analysis to detect anomalous activity. With the reduction in required processing, the Labyrinth's ability to detect anomalous activity is better equipped to discover malicious activity in real-time. With the detection of this malicious activity, the Labyrinth can automatically direct the activity to systems capable of preventing the traffic from affecting the production system.

Nathaniel Quist, nathanielquist1@gmail.com

References

CERN Computer Security. (2010, 06 10). *Mandatory Security Baselines*. Retrieved 09 29, 2016, from security.web.cern.ch: https://security.web.cern.ch/security/rules/en/baselines.shtml

Du, J., Zhang, X., Zhou, Y., & Bai, Y. (2013). *Active Defense Security Model in the Application of Network Deception System Design.* Luoyang Electronic Equipment Test Center of China (LEETC). Paris: Atlantis Press.

Fraser, G. (2014). *Creating a Baseline of Process Activity for Memory Forensics.* The SANS Institute, The InfoSec Reading Room. Bethesda: The SANS Institute.

Merriam-Webster. (2016, 10 20). *holistic*. Retrieved 10 20, 2016, from Merriam-Webster: http://www.merriam-webster.com/dictionary/holistic

Phishlabs. (2015). *The CISO's Guide to Spear Phishing Defense.* Phishlabs. Charleston: Phishlabs.

Revelle, D. (2011). Hypervisors and Virtual Machines. *;login;* , 5.

Rowe, N. C., & Custy, E. J. (2007). *Deception in Cyber-Attacks.* U.S. Naval Postgraduate School. Monterey: U.S. Naval Postgraduate School.

Sarala, R., Kayalvizhi, M., & Zayaraz, G. (2014). INFORMATION SECURITY RISK ASSESSMENT UNDER UNCERTAINTY USING DYNAMIC BAYESIAN NETWORKS. *IJRET: International Journal of Research in Engineering and Technology, 3* (7), 4.

Whyte, D., van Oorschot, P. C., & Kranaki, E. (2007). *Tracking Darkports for Network Defense.* Carleton University, School of Computer Science. Ottawa: Carleton University.

Wickman, T. D. (2003). *INTRUSION DETECTION IS DEAD. LONG LIVE INTRUSION PREVENTION!* The SANS Institute. Bethesda: The SANS Institute.

Nathaniel Quist, nathanielquist1@gmail.com

Appendix

The Labyrinth used for testing throughout this paper was created on a Dell PowerEdge R710 equipped with two, eight-core 2.53 GHz Intel Xeon Processors, 24GB RAM, 1 TB of hard disk space, and four NetXtreme II Gigabit Network Interface Cards (NICs). The hypervisor software running on the server is XenServer 7.0.1 64-bit, using Dell 6.4.0 BIOS.

The Labyrinth is connected to the ISP gateway via a standalone modem, set to the bridge setting. This setting forces the modem to provide ISP connectivity but will not interfere with any traffic moving across the device by routing or altering any of the packets. The Labyrinth is then directly attached to the modem via a standard Cat5e cable and connected to the eth0 NIC. The Labyrinth connects to an integrated firewall/router device, via eth1, distributing network connectivity to a small home office. Finally, the management port, eth3, is connected to a standard 5-port switch, which allows for the control of the Labyrinth's hypervisor architecture via an external laptop. The management port is also connected to the firewall/router providing an avenue for the SIEM to send blacklisted IP addresses to the firewall.

The hypervisor architecture used within the test Labyrinth is XenServer 7, which currently manages the following virtual systems and devices:

Table 2: Listing of Managed Virtual Machines

Firewalls	Versions	CPU	RAM	Harddrive	Purpose
IPFire	1.2	1	1GB	5GB	External Firewall
IPFire	1.2	1	1GB	5GB	Internal Firewall
IPFire	1.2	1	1GB	5GB	Network Segment Firewall
Desktops	**Versions**	**CPU**	**RAM**	**Harddrive**	**Purpose**
Windows	8.1	2	2GB	15GB	Network Desktop
Ubuntu	16.04 Workstation	2	2GB	15GB	Network Desktop
Ubuntu	16.04 Workstation	2	2GB	15GB	DNS Server
Ubuntu	16.04 Workstation	2	2GB	15GB	Web Server
SIEM Solutions	**Versions**	**CPU**	**RAM**	**Harddrive**	**Purpose**
Windows	8.1	4	4GB	200GB	Windows Based SIEM
Ubuntu	16.04 Server	4	4GB	200GB	Linux Based SIEM

I used two different SIEM deployments to test the functionality of the dynamic list creation for the Labyrinth. The Windows deployment used LogRhythm version 7.1.7

with no special modifications, and the Ubuntu deployment used an ELK stack (ElasticSearch, LogStash, Kibana) paired with the Watcher module. Both SIEM deployments functioned as expected after the creation of the appropriate whitelist baselines. At this point, there is no reason to say one SIEM is more apt in performing whitelist correlation functions as LogRhythm, ELK stack, Splunk, and IBM's QRadar all support this functionality.

To test the Labyrinth's external IP detection functionality, I used a paid VPN proxy service connected via a mobile hotspot simulating external IP addresses attempting to connect to the Labyrinth. The IP addresses in which I connected were as follows:

Table 3: Listing of Chosen External IP Addresses

IP Address	Country	City/State
184.75.220.154	Canada	Toronto
5.254.96.158	Romania	Bucharest
46.246.124.91	Sweden	Stockholm
179.43.133.114	United States	Michigan
96.44.188.98	United States	California

Both SIEM appliances were successful in identifying the external connections from two of the Labyrinth's data sources, the external firewall, and one of the Ubuntu systems configured as a Web Server. Both SIEM solutions successfully wrote the external IP addresses to their respective blacklists. My firewall currently uses iptables, so I was not able to make a direct direction to the firewall using Window's PowerShell. Plink.exe could be an option to make this functional. However, my Ubuntu machine was able to use a bash script to execute the IP Block Script via a cron job. Scripts for Cisco ASA, Juniper, and Palo Alto firewalls can also be utilized via the command line.

BLOCK_IP_Script.sh

```
#!/bin/bash

# Ingest the Blacklist IPs
for i in $(cat BLOCK_External_IP.txt)

do

# Check to see if the IPs are already blocked
    /sbin/iptables -L -n -v | grep -q "${i}"
    RETVAL=$?
    if [ $RETVAL -ne 0 ]; then

# Write the IPs to the blocklist
     /sbin/iptables -A INPUT -s "${i}" -j DROP

done
```

Figure 26: Block IP Bash Script