ZEEK INTRUSION DETECTION SERIES

Lab 7: Advanced Zeek Scripting for Anomaly and Malicious Event Detection

Document Version: 02-01-2020

Award 1829698
“CyberTraining CIP: Cyberinfrastructure Expertise on High-throughput Networks for Big Science Data Transfers”
Contents

Overview .................................................................................................................................................. 3
Objectives .................................................................................................................................................. 3
Lab topology .............................................................................................................................................. 3
Lab settings .............................................................................................................................................. 3
Lab roadmap .............................................................................................................................................. 4
1 Zeek’s default anomaly detection scripts .......................................................................................... 5
   1.1 Zeek scan-event .............................................................................................................................. 5
   1.2 Zeek bruteforce-event ..................................................................................................................... 6
2 Generating customized malicious network traffic .............................................................................. 7
   2.1 Starting a new instance of Zeek ..................................................................................................... 7
   2.2 Setting up the Bro2 machine for live network capture ................................................................. 8
   2.3 Using the Bro1 machine to launch customized TCP-based scans .............................................. 9
3 Applying Zeek scripts to filter network traffic ................................................................................. 11
   3.1 Applying the ZeekScanDetection filter ......................................................................................... 12
   3.2 Applying the ScanFilter filter .................................................................................................... 14
   3.3 Closing the current instance of Zeek ............................................................................................ 18
References .................................................................................................................................................. 19
Overview

This lab covers Zeek’s scripting language and introduces more advanced scripting capabilities. This lab simulates a new zero-day scanning technique and explains a Zeek script that captures this new event. The lab is designed to further highlight the customization properties of Zeek scripting.

Objectives

By the end of this lab, students should be able to:

1. Use precompiled Zeek scripts for identifying network traffic anomalies.
2. Develop a Zeek script for identifying and organizing specific malicious traffic events.
3. Generate customized malicious traffic to be used for testing purposes.

Lab topology

Figure 1 shows the lab workspace topology. This lab primarily uses the Zeek2 machine for offline Zeek script development and offline packet capture processing and analysis.

![Figure 1. Lab topology.](image)

Lab settings

The information (case-sensitive) in the table below provides the credentials to access the machines used in this lab.
Table 1. Device credentials for lab workspace.

<table>
<thead>
<tr>
<th>Virtual Machine</th>
<th>IP Address</th>
<th>Account</th>
<th>Password</th>
</tr>
</thead>
<tbody>
<tr>
<td>Zeek1</td>
<td>192.168.1.2</td>
<td>admin</td>
<td>password</td>
</tr>
<tr>
<td>DTN</td>
<td>192.168.1.3</td>
<td>root</td>
<td>password</td>
</tr>
<tr>
<td>Client</td>
<td>192.168.3.2</td>
<td>root</td>
<td>@dmin123</td>
</tr>
<tr>
<td>Zeek2</td>
<td>192.168.2.2</td>
<td>admin</td>
<td>password</td>
</tr>
<tr>
<td></td>
<td>192.168.3.1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Router</td>
<td>192.168.1.1</td>
<td>root</td>
<td>password</td>
</tr>
<tr>
<td></td>
<td>192.168.2.1</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>203.0.113.2</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 2. Shell variables and their corresponding absolute paths.

<table>
<thead>
<tr>
<th>Variable Name</th>
<th>Absolute Path</th>
</tr>
</thead>
<tbody>
<tr>
<td>$ZEEK_INSTALL</td>
<td>/usr/local/zeek</td>
</tr>
<tr>
<td>$ZEEK_TESTING_TRACES</td>
<td>/home/vlab/Zeek/testing/btest/Traces/</td>
</tr>
<tr>
<td>$ZEEK_PROTOCOLS_SCRIPT</td>
<td>/home/vlab/Zeek/scripts/policy/protocols/</td>
</tr>
<tr>
<td>$ZEEK_LABS</td>
<td>/home/vlab/Zeek-Labs-Workspace/</td>
</tr>
</tbody>
</table>

Lab roadmap

This lab is organized as follows:

1. Section 1: Zeek’s default anomaly detection scripts.
2. Section 2: Generating customized malicious network traffic.
3. Section 3: Applying Zeek scripts to filter network traffic.
1 Zeek’s default anomaly detection scripts

Zeek’s scripting language can be used to identify and report network anomalies by using event-driven functions. This section introduces two default Zeek script filters that are installed by default after Zeek installation.

While these default Zeek scripts might not correctly identify every unique anomaly, they provide a comprehensive starter code that can be customized further for anomaly-based detection.

1.1 Zeek scan-event

The first default Zeek script is the scan.zeek script. More information on this script can be found in Zeek’s documentation pages.


The file has been copied into the Zeek lab workspace directory and renamed to ZeekScanDetection.zeek for ease of access and name-reference clarity.

This Zeek script is used to identify scan-related traffic. Internet scanning can be split into three main categories:

1. Vertical Scanning: an attacker scans many ports on a single destination host address.
2. Horizontal Scanning: an attacker scans a single port on many destination host addresses.
3. Block Scanning: an attacker interweaves vertical and horizontal scanning techniques to increase complexity and become harder to track.

The script shown in the figure below list the first few lines of the ZeekScanDetection.zeek file.

```
1 #!TCP Scan detection
2 # .Authors: Sheharbano Khattak
3 # Seth Hall
4 # All the authors of the old scan.bro
5 @load base/frameworks/notice
6 @load base/frameworks/sumstats
7 @load base/utils/time
```

As shown in the figure above, loading other scripts is done through the @load statement with the following format:

```
@load <zeekscriptfile>
```

Lines 5, 6 and 7 include the functionalities found within the export blocks of the respectively included Zeek scripts.
The script leverages thresholds to determine if scan-like activities are present when processing network capture. If all the thresholds are exceeded, traffic is inferred to be scan-related.

For real time deployment, these thresholds will need to be modified dependent on the network size. For instance, a smaller network containing less IP addresses will need a lower threshold of scan packets to identify a scan-event. However, modifying these thresholds may result in an increase of false positives and true negatives, so it highly recommended to simulate and test network traffic before modification.

The figure above shows the thresholds in the `ZeekScanDetection.zeek` file. The thresholds are explained as follows. Each number represents the respective line number:

28. `const addr_scan_interval = $min & redef;` threshold to check a source IP address for varying destination IP address scan-related traffic. The default interval is 5 minutes.
32. `const port_scan_interval = $min & redef;` threshold to check a source IP address for varying destination port scan-related traffic. The default interval is 5 minutes.
35. `const addr_scan_threshold = 25.0 & redef;` threshold of unique destination IP addresses that a single host attempts to contact. The default threshold is 25 unique destination IP addresses.
38. `const port_scan_threshold = 15.0 & redef;` threshold of unique destination ports that a single host attempts to contact. The default threshold is 15 unique destination ports.

1.2 Zeek bruteforce-event

The second default Zeek script is the `detect-bruteforcing.zeek` script. More information on this script can be found in Zeek’s documentation pages.


The file has been copied into the Zeek lab workspace directory and renamed to `ZeekBruteforceDetection.zeek` for ease of access and name-reference clarity.
This Zeek script is used to identify brute-force password attacks. Brute-force attacks can be identified by several failed login attempts. This denotes that an attacker is attempting to systematically submit credentials until the correct credentials are found. The motivation behind this attack is to gain authorized access to an account, machine or server.

The script leverages the following thresholds to determine if scan-like activities are present when processing network capture. During real time deployment, these thresholds should be modified depending on the network size. The number of failed login attempts (or duration) should be modified to increase the script’s accuracy.

```
1 #! FTP brute-force detector, triggering when too many rejected usernames or
2 #! failed passwords have occurred from a single address.
3 #! load base/protocols/ftp
4 #! load base/frameworks/sumstats
5 #! load base/utilis/time
6 module FTP;
7  export {
8    redef enum Notice::Type += {
9      # Indicates a host bruteforcing FTP logins by watching for too
10      # many rejected usernames or failed passwords.
11      Bruteforcing
12    };
13    # How many rejected usernames or passwords are required before being
14    # considered to be bruteforcing.
15    const bruteforce_threshold: double = 20 &redef;
16    # The time period in which the threshold needs to be crossed before
17    # being reset.
18    const bruteforce_measurement_interval = 15mins &redef;
19 }
```

The thresholds are explained as follows. Each number represents the respective line number:

15. `const bruteforce_threshold`: threshold for the number of failed authentications attempts a source IP address can make. The default value is 20 failed attempts within the related time interval threshold.

18. `const bruteforce_measurement_interval`: threshold for the time to check a source IP address for failed authentication attempts. The default interval is 15 minutes.

## 2 Generating customized malicious network traffic

This section introduces creating and using a new Zeek script, tailored to react to more specific events.

### 2.1 Starting a new instance of Zeek

**Step 1:** On the top of the lab workspace, click on the *Bro2* icon as shown below to enter the *Bro2* machine.
Step 2: On the left side of the Bro2 desktop, click on the Terminal button as shown below.

Step 3. Start Zeek by entering the following command on the terminal. This command enters Zeek's default installation directory and invokes `zeekctl` tool to start a new instance. To type capital letters, it is recommended to hold the `Shift` key while typing rather than using the `Caps` key. When prompted for a password, type `password` and hit `Enter`.

```
cd $ZEEK_INSTALL/bin && sudo ./zeekctl start
```

A new instance of Zeek will now be active, and we can proceed to the next section of the lab.

### 2.2 Setting up the Bro2 machine for live network capture

Step 1: Navigate to the lab workspace directory and enter the `TCP-Traffic` directory. To type capital letters, it is recommended to hold the `Shift` key while typing rather than using the `Caps` key.

```
cd $ZEEK_LABS/TCP-Traffic
```
Step 2. Start live packet capture on interface *ens33* and save the output to a file named *ntraffic.pcap*. Take notice that the 0 in the following command is the number zero. If necessary, type `password` as the password.

```
sudo tcpdump -i ens33 -s 0 -w ntraffic.pcap
```

The *Bro2* machine is now ready to begin collecting live network traffic. Next, we use the *Bro1* machine to generate unique scan-based network traffic

### 2.3 Using the *Bro1* machine to launch customized TCP-based scans

In this section we use the *nmap* software to generate TCP-based scan traffic. TCP flags are included within the packets’ headers.

We will replicate a *zero-day* exploit within our subnet’s firewall. By including specific TCP flag combinations, packets can pass the subnet’s firewall unobstructed.

**Step 1.** On the top of the lab workspace, click on the *Bro1* button as shown below to enter the *Bro1* machine.

![Bro1 button](image)

**Step 2.** On the left side of the *Bro1* desktop, click on the Terminal button as shown below.
Step 3. Launch a TCP connect scan against the Client machine. If necessary, type `password` as the password.

```
sudo nmap -sT 192.168.3.2
```

Step 4. Launch a scan against the Client machine with the SYN, FIN and RST flags set. We will label this scan as Case1.

```
sudo nmap --scanflags SYN,FIN,RST 192.168.3.2
```

By specifying the `--scanflags` option, we can control which TCP flags are included in the packet header.
Step 5. Launch a scan against the Client machine with the SYN, RST and ACK flags set. We will label this scan as Case2.

```bash
sudo nmap --scanflags SYN,RST,ACK 192.168.3.2
```

Step 6. Return to the Bro2 machine and use the `Ctrl+c` key combination to stop the live traffic capture session.

The capture session’s results show that 6050 packets were captured. We will now process the `nttraffic.pcap` file using Zeek while including Zeek filters to organize and identify these new zero-day exploits. Note that the number of packets captured may vary per session and for the purpose of this lab, it is okay to continue.

### 3 Applying Zeek scripts to filter network traffic
Now that we have collected traffic containing the zero-day exploits, we will process the packet capture file using Zeek.

3.1 Applying the ZeekScanDetection filter

**Step 1:** Process the *nttraffic.pcap* packet capture file while in the TCP-Traffic directory. To type capital letters, it is recommended to hold the Shift key while typing rather than using the Caps key. Note that using the tab key to autofill the directory may cause errors in finding the ZeekScanDetection.zeek filter and it is recommended to manually type the entire file path.

```
zeek -C -r ntraffic.pcap /$ZEEK_LABS/ZeekScanDetection.zeek
```

**Step 2:** Display the contents of the notice.log file using the `cat` command.

```
cat notice.log
```

Scan::Port Scan 192.168.1.2 scanned at least 15 unique ports of host 192.168.3.2
Within the *notice.log* file, we can see the *Bro1* machine has been identified for creating scan-based network traffic and exceeding the 15-ports threshold configured earlier.

**Step 3:** Display the contents of the *conn.log* file using the following command.

```
head -n 25 conn.log | zeek-cut ts id.orig_h id.orig_p id.resp_h id.resp_p history
```

The Terminal command is explained as follows:

- **`head -n 25 conn.log`**: returns the top 25 rows of the *conn.log* file, specified by the `-n` option.
- **`| zeek-cut ts id.orig_h id.orig_p id.resp_h id.resp_p history`**: uses the `zeek-cut` utility to return the specified columns and remove padding.

The `history` column (last column in the figure above) contains information regarding which TCP flags were found within a packet header:

- `S`: SYN flag.
- `H`: SYN+ACK flags.
- `A`: ACK flag.
- `F`: FIN flag.
- `R`: RST flag.
- `U`: URG flag.
- `Q`: Multiple flags set.

The event is attributed to the host when the flag letter is uppercase; otherwise, it is attributed to the receiver.
3.2 Applying the ScanFilter filter

**Step 1:** Navigate to the lab workspace directory.

```bash
cd $ZEEK_LABS
```

**Step 2:** Display the contents of the `ScanFilter.zeek` file using `nl`.

```bash
nl ZeekFilter.zeek
```

The script is explained as follows. Each number represents the respective line number:

1. Declares a new module workspace.
2. Export block allows code to be accessed outside the current module workspace.
3. Creates and appends the `CASE1LOG` to the list of Log files.
4. Creates and appends the `CASE2LOG` to the list of Log files.
6. Block that includes all the columns and features to be included in these new log files. Each will contain a variable type and output location:

   - `ts`: time that the packet was received.
   - `id`: packet identification number.
   - `orig_h`: source IP address.
- orig_p: source port.
- resp_h: destination IP address.
- resp_p: destination port.
- history: string of flag characters.

16. Initialization event.
17. Creates a new log stream using the previously introduced CASE1LOG LOG ID, `outputFormat` column formatting and a file name path.
18. Creates a new log stream using the previously introduced CASE2LOG LOG ID, `outputFormat` column formatting and a file name path.
20. Event triggered when a TCP packet is processed.
21. Creates a local variable `rec` to store the column-related information, using the current packet data, accessed with the `$id$<column>` format.
22. Checks if the SFR flag combination is present in the packet. This relates to the history column, containing SYN-FIN-RST flags.
23. If the SFR flag combination is present, the packet will be written to the CASE1LOG log stream with the packet information passed through the local variable `rec`.
24. Checks if the SRA flag combination is present in the packet. This relates to the history column, containing SYN-RST-ACK flags.
25. If the SRA flag combination is present, the packet will be written to the CASE2LOG log stream with the packet information passed through the local variable `rec`.

**Step 3:** Process the `nttraffic.pcap` packet capture file while in the lab workspace directory.

```
zeek -r TCP-Traffic/nttraffic.pcap ScanFilter.zeek
```
Step 4: List the generated log files in the current directory.

```
ls
```

Note the `Case1.log` and `Case2.log` files generated by including the `ScanFilter.zeek` filter during processing.

Step 5: View the contents of the `Case1.log` file.

```
head -n 25 Case1.log | zeek-cut ts id.orig_h id.orig_p id.resp_h id.resp_p history
```
The Terminal command is explained as follows:

- `head -n 25 Case1.log`: returns the top 25 rows of the `conn.log` file, specified by the `-n` option.
- `| zeek-cut ts id.orig_h id.orig_p id.resp_h id.resp_p history`: uses the `zeek-cut` utility to only return the specified columns, and removes padding.

Unlike the default example, we can see the `history` column contains the exact same flag. Our filter was successful in organizing the traffic related to the `Case1` exploit.

**Step 6:** Display the contents of the `Case2.log` file.

```
head -n 25 Case2.log | zeek-cut ts id.orig_h id.orig_p id.resp_h id.resp_p history
```
The Terminal command is explained as follows:

- `head -n 25 Case2.log`: returns the top 25 rows of the `conn.log` file, specified by the `-n` option.
- `[zeek-cut ts id.orig_h id.orig_p id.resp_h id.resp_p history]`: uses the `zeek-cut` utility to only return the specified columns, and removes padding.

Unlike the default example, we can see the `history` column contains the exact same flag. Our filter was successful in organizing the traffic related to the `Case2` exploit.

### 3.3 Closing the current instance of Zeek

After you have finished the lab, it is necessary to terminate the currently active instance of Zeek. Shutting down a computer while an active instance persists will cause Zeek to shut down improperly and may cause errors in future instances.

**Step 1.** Stop Zeek by entering the following command on the terminal. If required, type `password` as the password. If the Terminal session has not been terminated or closed, you may not be prompted to enter a password. To type capital letters, it is recommended to hold the **Shift** key while typing rather than using the **Caps** key.

```
cd $ZEEK_INSTALL/bin && sudo ./zeekctl stop
```
Concluding this lab, we introduced default frameworks for anomaly-detection scripts. We generated malicious network traffic to simulate a zero-day exploit, and then processed the traffic using a customized a Zeek script. With the resulting Zeek log files, these exploits can be studied for additional analysis and mitigation.

References