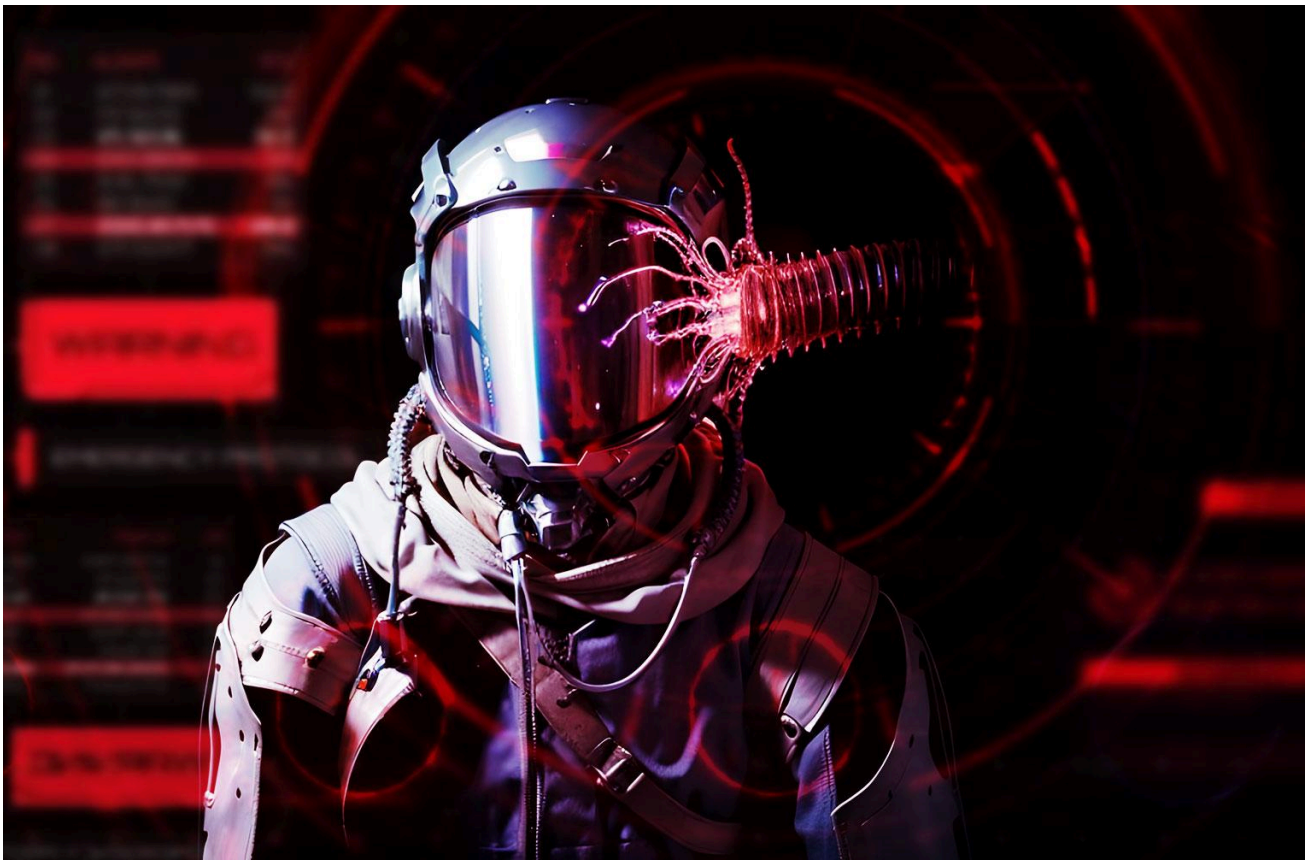
MANUAL AND AUTOMATIC DETECTION OF C&C CHANNELS



"The one where we automate all the things"

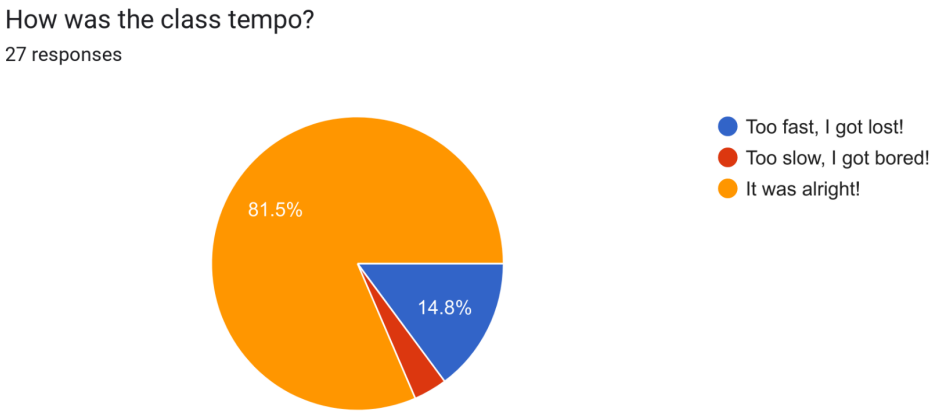
December 5th, 2024

Credits

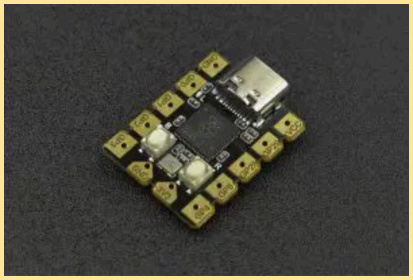
Content: Sebastian Garcia, Veronica Valeros, Alya Gomaa, Ondřej Lukáš, Martin Řepa, Lukáš Forst, Muris Sladić, Maria Rigaki
Illustrations: Fermin Valeros
Design: Veronica Garcia, Veronica Valeros, Ondřej Lukáš
Music: Sebastian Garcia, Veronica Valeros, Ondřej Lukáš
CTU Video Recording: Jan Sláma, Václav Svoboda, Marcela Charvatová
Audio files, 3D prints, and Stickers: Veronica Valeros

CLASS DOCUMENT	https://bit.ly/BSY2024-11
WEBSITE	https://cybersecurity.bsy.fel.cvut.cz/
CLASS MATRIX	https://matrix.bsy.fel.cvut.cz/
CLASS CTFD (CTU STUDENTS)	https://ctfd.bsy.fel.cvut.cz/
CLASS PASSCODE FORM (ONLINE STUDENTS)	https://bit.ly/BSY-VerifyClass
FEEDBACK	https://bit.ly/BSYFEEDBACK
LIVESTREAM	https://bit.ly/BSY-Livestream
INTRO SOUND	https://bit.ly/BSY-Intro
VIDEO RECORDINGS PLAYLIST	https://bit.ly/BSY2024-Recordings
CLASS AUDIO	https://audio.com/stratosphere

Results from the survey of the last class (14:32, 1m)



Pioneer Prize for Assignment 8 (14:33, 1m)

1st Place	2nd Place	3rd Place
		
DFRobot Beetle RP2040 Mini Development Board	USB 2.0 isolator	Solderless Zero Dongle for Raspberry Pi Zero / Zero W
Tomáš (veselt27)	**David (kostada2)**	**Lukáš (kaufmlu1)**

Parish notices, Exam & Bonus Announcements (14:34, 3m)

- **CTU Students exam dates are confirmed as follows:**
  - Tuesday, January 21st. From 14:30 - 16:30. KN-107
  - Thursday, January 23rd, From 14:30 - 16:30. KN-107
  - Thursday, January 30th. From 14:30 - 16:30. KN-107

    - Thursday, February 6th. From 14:30 - 16:30. KN-107
- Bonus Assignment will open on **December 20th, 2024 at 21:00 CET**
    - More details to come soon. Next class

Class outline

1. Manual Detection of C&C Channels in the Network
2. Traffic Analysis with Zeek
3. Automatic Detection of Command and Control and attacks with Machine Learning
4. Slips IDS

Manual Detection of C&C Channels in the Network (14:37, 1m)

Goal: to learn how to detect command-and-control channels in the network manually

Detecting command and control (C&C) channels is very important for:

- Obtaining **confirmation** that there is malware (sometimes the only confirmation).
- Knowing if the malware is **autonomous** or **controlled** (which is much worse).
- Knowing if the malware **worked** or not.
- Knowing if the malware is **alive** or dormant.

However, detecting C&C communications is hard because they sometimes look too similar to real benign communications. In this case, experience is key, and context is critical.

Let's detect some Command and Control (14:38, 0m)

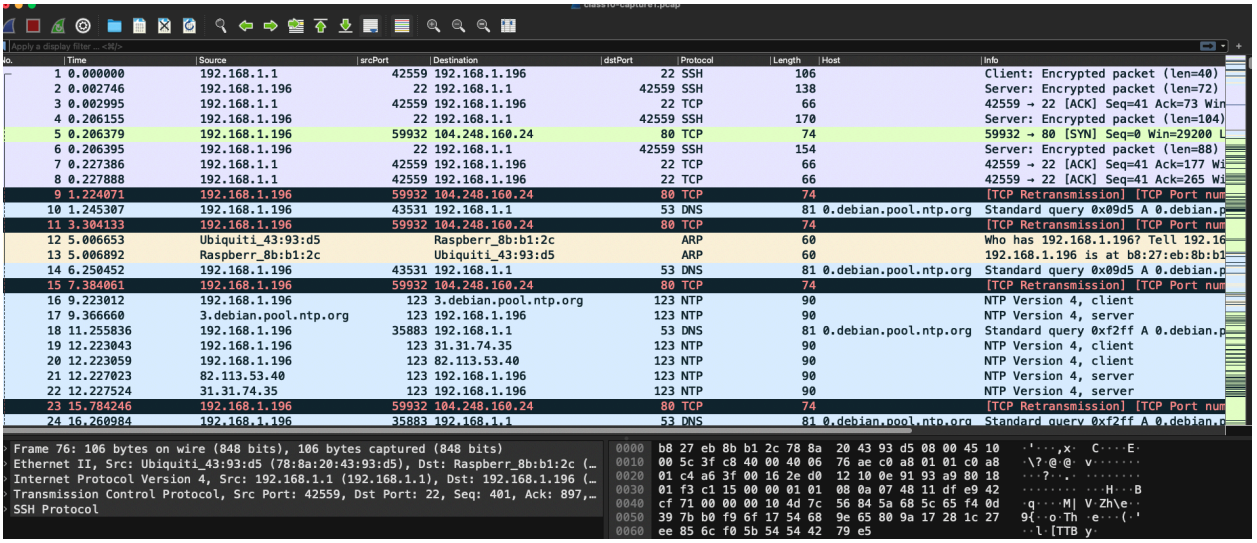
All: Host computer. Not in dockers.

Download the PCAP **class10-capture1.pcap** to your computer

1. https://mega.nz/file/o9FSnBhR#ZHXWGtpi2lSj1JA0jAqXSIrPMfF1u1dT26JAE-3f7hI
2. Or: https://drive.google.com/file/d/1iozPavBt3uc2CxSExwlxxtMWb2WGcj19/view?usp=sharing
3. It is also in /data/class10-capture1.pcap in your dockers CTU docker SCL class11-lab docker

Analyzing the C&C with Wireshark (14:38, 25m)

Open class10-capture1.pcap in Wireshark. You should see something like this:



Clean, Rinse, Repeat Analysis Methodology

- This capture contains 3,000 packets. Do you see the counter?
- We are going to go packet by packet.
- To leave an idea of the process in this document, we are going to:
  - 🕵🏽 Identify each connection
  - 🔎 See if it is benign or not for us
  - 👜 Filter out what we know is benign

- 🏳️‍🌈 Colorize the conversations that we are interested in to recognize them fast
- ↩ Continue until we identify all the conversations

Step-by-Step Analysis

- Before starting, we like to add a new custom column to see the packets that belong to the same stream: tcp.stream
  - As we did before: Right-click on the columns bar → 'Column Preferences' → 'Add new column' → Change name to TCPStream → Change Type to 'custom' → Change field to tcp.stream
  - Move (drag and drop) the new column below the column 'host'.
- 192.168.1.1 -> 192.168.1.196 (dPort 22)
  - What is this connection? → follow TCP stream
    - Look at the protocol column. Wireshark identifies the protocol by the packet's content.
  - We know it is SSH, and from the default GW, most probably.
  - This can be suspicious. Since we did the capture, we confirm this is how we got into the computer.
  - We can continue then. Forget this connection. Put this in the filter.[1]
    - ! tcp.stream eq 0
- 192.168.1.196 -> 104.248.160.24 (dPort 80)
  - What is this connection?
  - Follow TCP stream
    - A connection attempt.
    - Suspicious that no RST was back.
    - Also very few packets.
  - Filter out

[1] We know is tcp.stream 0 because when doing the tcp stream Wireshark identifies the stream as stream 0

        - `! tcp.stream eq 0 && ! tcp.stream eq 1`
- Let's forget **ARP**, it doesn't seem to be a local thing.
    - `! tcp.stream eq 0 && ! tcp.stream eq 1 && not arp`
- 192.168.1.196 -> 192.168.1.1 (dPort 53) → follow UDP stream
    - What is this connection?
        - DNS to an NTP server. Debian.
        - Benign
    - Forget all of them
        - `! tcp.stream eq 0 && ! tcp.stream eq 1  && not arp && ! dns.qry.name contains "ntp.org"`
        - See that we are not forgetting all the DNS, only this type.
- 192.168.1.196 -> 89.221.210.188 (dPort 123) NTP connection -
    - What is this?
        - NTP. Network Time Protocol.
    - Use the filter ntp to filter all NTP traffic in the capture. It looks like normal NTP. Slow. Not large.
    - `Forget them: ! tcp.stream eq 0 && ! tcp.stream eq 1  && not arp && ! dns.qry.name contains "ntp.org" && ! ntp`
- 192.168.1.196 -> 104.248.160.24 (dPort 80) → follow TCP stream
    - This looks suspicious
        - Downloads a file called 'ntpd'.
        - Is a binary
        - Is a Linux executable.
        - No domain in the host.
        - User-Agent is wget.
    - Let's investigate all the other connections from this IP.
    - ip.addr == 104.248.160.24 && tcp.port == 80 && http → check all HTTP connections to this external server

        - Confirmation of downloads.
    - ip.addr == 104.248.160.24 && tcp.port == 80 && http && (http.request.method == "GET" || http.request.method == "POST") → to see only the downloads.
    - Take notes and continue.
    - Forget it:
        - `! tcp.stream eq 0 && ! tcp.stream eq 1  && not arp && ! dns.qry.name contains "ntp.org" && ! ntp && !(ip.addr == 104.248.160.24 && tcp.port == 80) && !(ip.addr == 104.248.160.24 && tcp.port == 23)`
- 192.168.1.196 -> 104.248.160.24 (dPort 23) → follow TCP stream
    - ip.addr == 104.248.160.24 && tcp.port == 23 → check all the traffic to this external server on port 23
        - Seems a connection attempt that the port is filtered.
    - This is stream 9. Continue looking at all the next streams.
        - tcp.stream eq 10
        - Which is the first that looks like it worked?
    - What about tcp.stream eq ?
    - What is this traffic?
        - Follow the TCP stream.
        - Looks like connections to a TELNET server and some data are being sent.
        - It looks like a command and control sending an order.
        - This means that the binaries were probably executed, and they seem to have worked (at least one).
    - If you keep looking at streams, do you notice something suspicious in the destination IPs?
        - See all of them tcp.port == 23
        - To see all of the port 23 with data: tcp.port == 23 and tcp.len > 0

        - To see all the TELNET (independently of the port): telnet
    - Take note and forget all these Telnets.
        - `! tcp.stream eq 0 && ! tcp.stream eq 1  && not arp && ! dns.qry.name contains "ntp.org" && ! ntp && !(ip.addr == 104.248.160.24 && tcp.port == 80) && !(ip.addr == 104.248.160.24 && tcp.port == 23)  && !tcp.port == 23`
- No more packets! done.

This is the full Wireshark filter. After applying the full filter, you should not see any packets missing, as we have managed to explain all the behaviors:

! tcp.stream eq 0 && ! tcp.stream eq 1 && not arp && ! dns.qry.name contains "ntp.org" && ! ntp && !(ip.addr == 104.248.160.24 && tcp.port == 80) && !(ip.addr == 104.248.160.24 && tcp.port == 23) && !tcp.port == 23

Analysis Summary (15:03, 2m)

This capture contains the traffic of a host, 192.168.1.196, which seems to be infected with malware. We identified the following key behaviors:

- Benign behavior:
    - benign device management traffic from the router to 192.168.1.196, port 22
    - benign DNS traffic from 192.168.1.196 to resolve ntp.org servers
    - benign ARP traffic in the local network
    - benign NTP traffic from 192.168.1.196
- Malicious behavior:
    - suspicious connection attempt to an HTTP server; server unavailable
    - malicious file downloads on server 104.248.160.24; malicious ELF files
    - malicious TELNET (C&C) connections to server 104.248.160.24
    - malicious network service discovery (telnet) through horizontal scanning on dPort 23.

Recap: Manual analysis is very important as a skill. It is hard to learn, but this is the deepest you can go. Some malware/attacks can only be found like this. Be methodological and take notes.

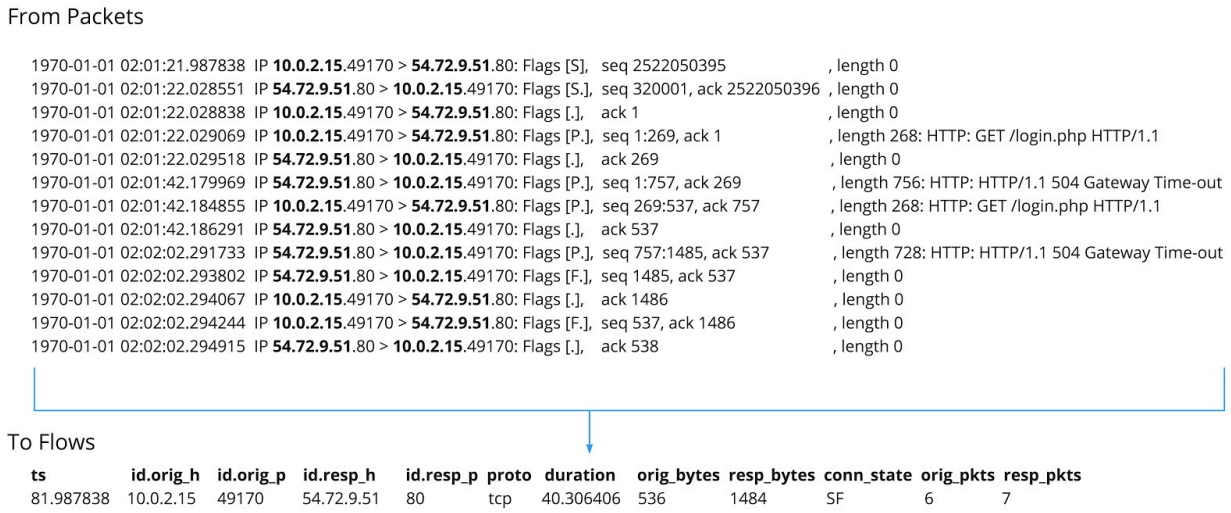
Traffic Analysis with Zeek (15:05, 1m)

Even though the network analysis with Wireshark was good, there are still more things to find. However, for a small capture, it took us a long time. How can we do more?

By using network flows. In this case, Zeek.

Zeek (formerly Bro) "*is a network monitor that quietly and unobtrusively observes network traffic and interprets what it sees to create compact, high-fidelity transaction logs, and file content.*"[2][3]

What is a network flow? (15:06, 2m)



[2] 'The Zeek Network Security Monitor', Zeek. https://zeek.org/ (accessed Dec. 05, 2024).
[3] Zeek Cheat Sheet, Corelight, https://f.hubspotusercontent00.net/hubfs/8645105/Corelight_May2021/Pdf/002_CORELIGHT_080420_ZEEK_LOGS_US_ONLINE.pdf

Using Zeek (15:08, 15m)

CTU docker SCL class11-lab docker

1. Create a special folder for the logs: you don't want all the files getting mixed.
    1. `cd`
    2. `mkdir zeek_logs`
    3. `cd zeek_logs`
        i. `zeek -Cr /data/class10-capture1.pcap`
            1. -C → If PCAP checksums are bad, fix it and continue.
            2. -r → read a PCAP file
2. If you can not do it, the Zeek log files are also here for you:
    1. CTU Dockers: /data/bsy_class_11_zeek_logs/
    2. Online: https://drive.google.com/file/d/18MKHnnLQS60sYuV52xGX3DTp8FqQchZP/view?usp=sharing
    3. Online ZIP with Password (bsyrocks!): https://drive.google.com/file/d/1k5lW5o7TP-scKo8nKArPBeGS98yONF1E/view?usp=sharing
3. Let's check the logs created in zeek_logs
    1. `ls -alh ~/zeek_logs`
4. What are the log files?[4]
    1. conn.log[5]: TCP/UDP/ICMP connections flows[6] (no ARP)
        i. Connections or flows
        ii. `less -S conn.log`
        iii. Or better

[4] 'Log Files — Book of Zeek (v5.1.0)'. https://docs.zeek.org/en/current/script-reference/log-files.html (accessed Dec. 01, 2022).
[5] conn.log, http.log, ssl.log - Book of Zeek, https://docs.zeek.org/en/current/scripts/base/protocols/conn/main.zeek.html. Accessed on 06/24/2022.
[6] M. Hayes, 'What is a Network Traffic Flow?', Bits 'n Bytes, Sep. 26, 2018. https://mattjhayes.com/2018/09/26/what-is-a-network-traffic-flow/ (accessed Dec. 01, 2022).

```
1. cat conn.log |column -t |less -S
```

iv. Or better use the custom zeek-cut program from Zeek.

```
1. cat conn.log | zeek-cut -d -M| column -t | less -S
```

a. -d: Print human dates.

b. -M: Include all format header blocks in the output in minimal view.

2. dns.log: DNS activity
   i. `cat dns.log | zeek-cut -d -M| column -t | less -S`
3. files.log: File analysis results (e.g.: files downloaded and hashes)
   i. `cat files.log | zeek-cut -d -M| column -t | less -S`
4. http.log[6]: HTTP requests and replies
   i. `cat http.log | zeek-cut -d -M| column -t | less -S`
5. ntp.log[6]: NTP Protocol
   i. `cat ntp.log | zeek-cut -d -M| column -t | less -S`
   ii. Notice the field 'ref_id' to know where the time data comes from!
6. There are many others[7]!

Zeek States (15:23, 4m)

The states are the most important part of recognizing what a connection did. The field is called **'conn_state'** in the conn.log.

[7] Example: https://mcfp.felk.cvut.cz/publicDatasets/CTU-Malware-Capture-Botnet-334-1/bro/

SF	Connection established. Connection terminated.	SHR	Responder sent a SYN ACK followed by a FIN. No SYN from the originator.
S0	Connection attempt seen. No reply.	REJ	Connection attempt rejected.
S1	Connection established. Not terminated.	RSTO	Connection established. Originator aborted with a RST.
S2	Connection established. Close attempt by originator. No reply from responder.	RSTR	Responder sent a RST.
S3	Connection established. Close attempt by responder. No reply from originator.	RSTOS0	Originator sent a SYN followed by a RST. No SYN-ACK from the responder.
SH	Originator sent a SYN followed by a FIN. No SYN ACK from the responder.	RSTRH	Responder sent a SYN ACK followed by a RST. No SYN from the originator.
OTH	No SYN seen, just midstream traffic. Example of this is a "partial connection" that was not later closed.		

Zeek history

Zeek history is a special field in conn.log that shows all the history of **state** changes in that flow. For example:

```
1545402975.921971   Cb0wv5SToIz5VV4r1   192.168.1.196  59934
104.248.160.24 80   tcp   http   2.248716   149   119443 SF   T   F   0
ShADadttfF   88   5737   85   125319 -
```

The explanation is as follows[8]:

Zeek History

Orig UPPERCASE, Resp lowercase, compressed

S	A **S**YN without ACK bit set	**C**	Packet with bad **c**hecksum
H	A SYN-ACK ("**h**andshake")	**I**	**I**nconsistent packet (both SYN & RST)
A	A pure **A**CK	**Q**	Multi-flag packet (SYN & FIN or SYN + RST)
D	Packet with payload ("**d**ata")	**T**	Re**t**ransmitted packet
F	Packet with **F**IN bit set	**W**	Packet with zero **w**indow advertisement
R	Packet with **R**ST bit set	**^**	Flipped connection

So, for **ShADadttfF**, what is it?

Remember that **capitalized** letters indicate an action by the connection **originator**. **Lowercase** letters indicate an action by the **responder**.

The important part of this history is the **flipped** connection ^. This means that ***some*** packets may have been **lost** and that the **Src IP and Dst IP (and ports) may be flipped**. It indicates that what you believe is the **Src IP** is the **Dst IP** and vice versa.

[8] Source: https://f.hubspotusercontent00.net/hubfs/8645105/Corelight_May2021/Pdf/002_CORELIGHT_080420_ZEEK_LOGS_US_ONLINE.pdf

Imagine, for example, if the first SYN packet is lost!

~~~~ 💗 First Break! 💗~~~~ (15:27, 10m)

Remember that after the break, you will need a Google Account to use the Google Colab environment.

Example threat hunting question: With whom did the client connect? (15:37, 5m)

This is a simple threat-hunting question. We want to find the destination IP and port of all the hosts with which **the client** had established connections. But we only want those connections that transfer data.

1. Take your time and try to solve it by yourself first.
2.
```
cat conn.log | awk -F'\t' '{if ($3=="192.168.1.196" && ($12=="S1" || $12=="S2" || $12=="S3" || $12=="RSTO" || $12=="RSTR" || $12=="SF") && $10>0 ) print $0}' | awk -F'\t' '{print $5" "$6}' | sort | uniq -c| sort -r
```
    a. The field separator is a TAB (-F '\t')
    b. Field $3 is the source IP.
    c. Field $12 is the state. The state should be established (see below)
    d. Field $10 is the amount of bytes sent by the source. We are searching for connections with data (> 0)
    e. Field $0 is the complete line. This means that if the condition is true, print the whole matching line.
    f. Then, only in the matching lines print the destination IP ($5) and destination port ($6).
    g. Sort them, obtain unique combinations, and sort them by the amount of times they appeared

Zeek Scripts: Extract Files (15:42, 5m)

Zeek has a beautiful programming language based on Lua that is Turin-complete[9]. There is a package manager called zkg[10] that allows you to install the many scripts created by the community.

1. Zeek can run different scripts to do more things. Many more.
    1. Check /opt/zeek/share/zeek/policy/
    2. And /opt/zeek/share/zeek/policy/frameworks/files/
2. Special script to extract all files in the PCAP and store them
    1. `cd zeek_logs`
    2. `zeek -r /data/class10-capture1.pcap /opt/zeek/share/zeek/policy/frameworks/files/extract-all-files.zeek`
    3. `cd extract_files`
    4. `file *`

```
name-HTTP-uid:  ELF 32-bit LSB executable, MIPS, MIPS-I version 1 (SYSV), statically lin...
name-HTTP-uid:  ELF 32-bit LSB executable, Renesas SH, version 1 (SYSV), statically lin...
name-HTTP-uid:  ELF 64-bit LSB executable, x86-64, version 1 (SYSV), statically linked, ...
name-HTTP-uid:  ELF 32-bit LSB executable, ARM, EABI4 version 1 (SYSV), statically link...
```

    5. Do you want to know the commands done by the malware?
        a. `strings -n 10 extract-1545403174.025504-HTTP-* |head -n 1`
        b. Why did this command work?
    6. Be careful! These are real Linux malware! **Do not execute!**

Recap: Flows are critically important in most organizations because pcaps are expensive in storage. Flows can detect many malicious behaviors.

[9] Turing completeness, Wikipedia, https://en.wikipedia.org/wiki/Turing_completeness. Accessed on 12/05/2024
[10] Zeek.org. (2024). Zeek Package Manager: Home. [online] Available at: https://packages.zeek.org/ [Accessed 5 Dec. 2024].

Automatic Detection of Command and Control and attacks with Machine Learning (15:47, 5m)

To show an alternative way to analyze traffic data automatically. Learn to use machine learning to cluster the traffic and compare our model with the human analyst labels.

Google Colab

Today, we are going to work with Python and Google Colab. You need a Google account to log in, get ready. CTU students can use their university accounts

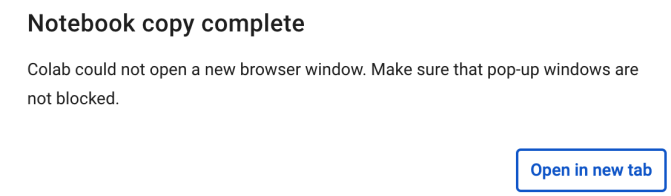
Google Colab (short for Colaboratory) is an online tool created by Google. It lets you write and run code, especially **Python** code, on Google's computers instead of your own.

We created a Google Colab Notebook for you to use. We need you to **make a copy of this notebook**:

- Access the notebook at:
  Class 10 - Manual and Automatic Detection of C&C Channels [2023.12.14].i...
- Go to Menu File

- Select "Save a copy in Drive"

- Select "Open in new tab"



- **Use this COPY. This is YOUR copy, and you can do whatever you want.**

Slips IDS (17:27, 1m)

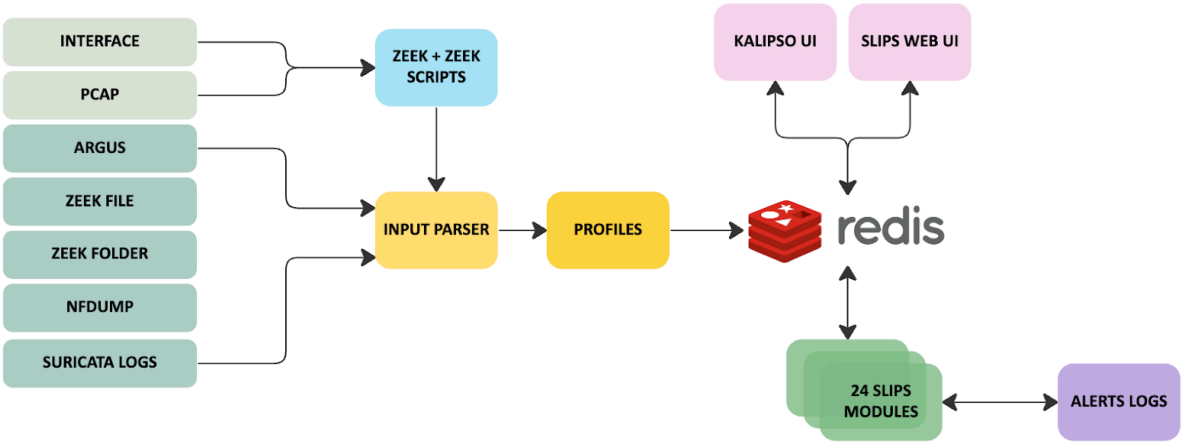
Slips[11] is a Python-based machine-learning tool created by our Stratosphere laboratory. It does not rely on signatures and instead, it analyses network behaviors to detect malicious activities. It is the first free software machine learning-based IDS, you can download and contribute on GitHub:

- https://github.com/stratosphereips/StratosphereLinuxIPS

We are going to use Slips to automatically see how ML can make some detections and how it is important to make the difference between evidence and alerts.

All of this is possible to do today thanks to the collaboration of Alya Gomaa, the main Slips developer in Stratosphere. You can thank her in Matrix.

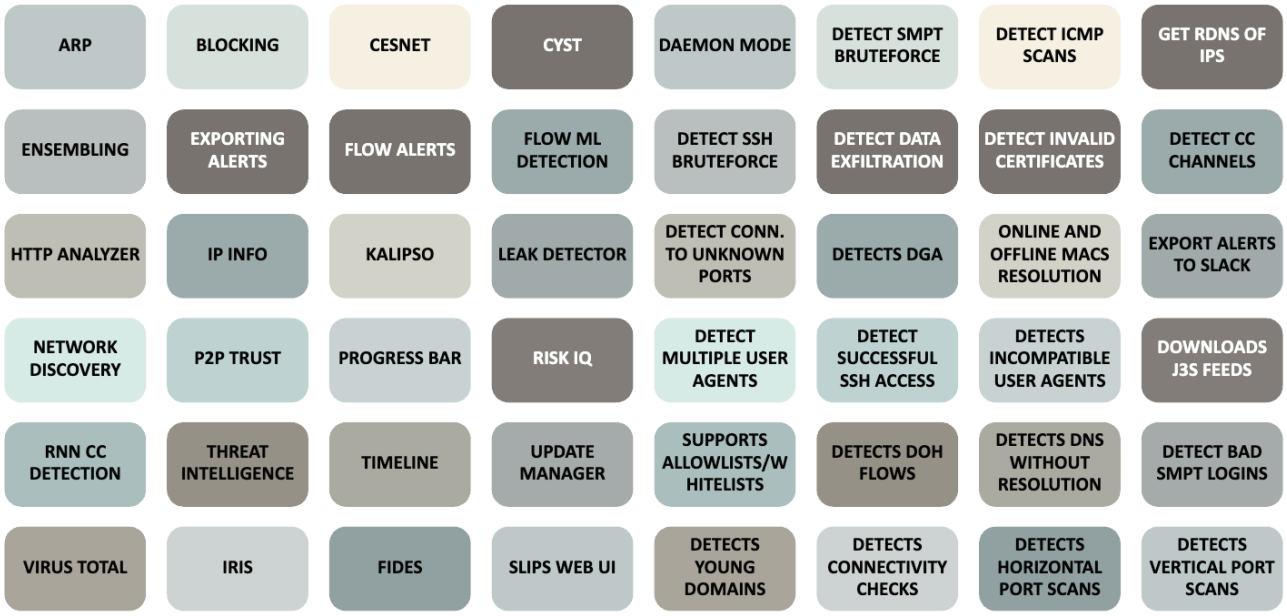
Architecture of Slips (17:28, 1m)



Modules in Slips (17:29, 1m)

Slips has many modules which enhance his capabilities. Here are some of them:

[11] Slips — Slips v1.1.4 documentation, https://stratosphereLinuxips.readthedocs.io/en/develop/. Accessed on 04/12/2024.



For home: Install Slips in your containers (17:30, 1m)

CTU Students.

- Everything is done in your dockers. But in case you want to know how to install
- Shallow clone of the Slips repository:
  - `git clone --depth 1 https://github.com/stratosphereips/StratosphereLinuxIPS /StratosphereLinuxIPS`
  - `cd /StratosphereLinuxIPS`
- Install the dependencies:
  - `python3 -m pip install -r install/requirements.txt`

Online Students:

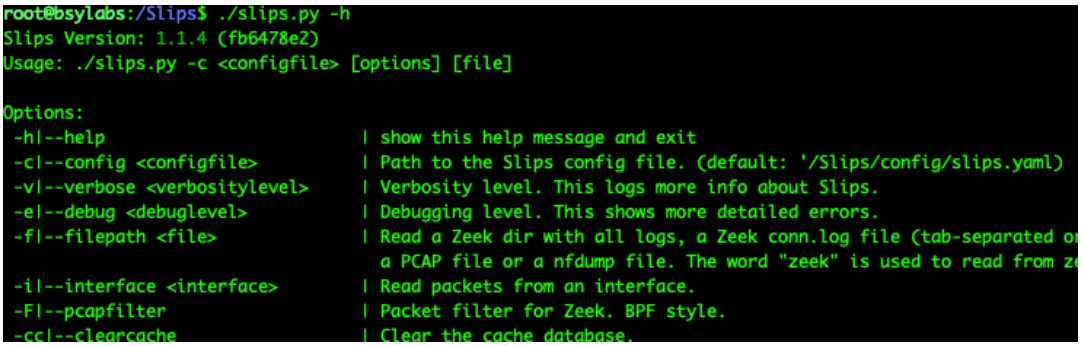
- All is done in the docker called SCL class11-lab docker. So connect there using the password `'admin'`
  - `ssh root@172.20.0.101`

All Students:

Let's check it works.

CTU docker SCL class11-lab docker

- `cd /StratosphereLinuxIPS`
- `./slips.py -h`



Execution of Slips in Malware Captures (17:31, 10m)

Let's use Slips on the capture **`training-capture-009.pcap`**

The file is in /data in CTU and SCL dockers. But if you want a copy, it is here.

CTU docker · SCL class11-lab docker

- `cd /StratosphereLinuxIPS`

Let's change the default detection threshold to be more sensitive since the capture is small. **This is already done for CTU students:**

- `sed -i 's/: 0\.25/: 0\.08/g' config/slips.yaml`

Let's run Slips (it should take ~3 minutes or less in normal conditions):

- `tmux new -t slips`
- `wget "https://drive.usercontent.google.com/download?id=1HJA3GBjjNZDev83c7yfCVRInNxlUuUJP&export=download&authuser=0" -o training-capture-009.pcap`
- `./slips.py -f /data/training-capture-009.pcap`
  - `-f: The input file (pcap, Zeek conn.log, Zeek folder, nfdump, Suricata JSON, etc.)`

How to see the detection logs:

- `cd output/training-capture-009.pcap_2024-12-05*`
- `less -S alerts.log`

If you don't have **logs**, you can use these copies:

- In CTU Dockers
    - `/data/slips-logs/training-capture-009`
    - cp -a /data/slips-logs/training-capture-009 output/
- Online
    - https://drive.google.com/file/d/1FfE4BPMgbzBnURhyavemzngUuHBeSFsl/view?usp=sharing

You can summarise the logs if you are in the same folder as alerts.log:

- `cat alerts.log | awk -F 'Detected' '{print $2}'|sort |uniq -c |sort -rn`

You can put this line in a script, although it needs to be in the same directory that alerts.log:

- `/data/slips-alerts-sort.sh`

If Slips is still running, you can use the watch command to show the summary every 1 second:

- `/data/slips-alerts-sort-monitor.sh`

Slips is very complex, including downloading more than **45 threat intelligence feeds** in real-time from the Internet. But this is done in the 'background' by default.

So when Slips finishes analyzing a capture, it may stay working for some minutes to download all the files. The files are stored in Redis, so the next time you run it, you don't have to download them (but they will be updated every couple of days).

You can wait for this, or you can stop it with CTRL-C. You can check with **ps afx** to ensure no Slips' instances are running.

Extra: Output of Slips in the Web (17:41, 1m)

The output of slips can also be seen using the *Kalipso Nodejs* interface in the console or the web interface (in beta). Apart from exporting to TAXII servers, Slack, CESNET Warden, CYST, sending emails, and doing pop-ups on your screen.

The web server can be started with the **-w** parameter, but you can also start it later by running ./webinterface.sh.

In SCL, the docker with Slips we installed is the 'light' version, so it is smaller (done by Alya specifically for this class), so you don't have the web interface there. But you can get the full docker from here.

If the webserver is up, you will see port 55000 open **externally** in your docker (which means everyone in the local network will see it).

Check that the port 55000 is open:

CTU docker

- `netstat -anp|less`

To see the web of Slips, connect to your docker with a tunnel to the local port

CTU Students:

All: Host computer. Not in dockers.

- You need to do this from your computer. SSH to your docker again with a tunnel.
- `ssh -L 55000:localhost:55000 root@147.32.80.36 -p <YourPort>`

Online Students

All: Host computer. Not in dockers.

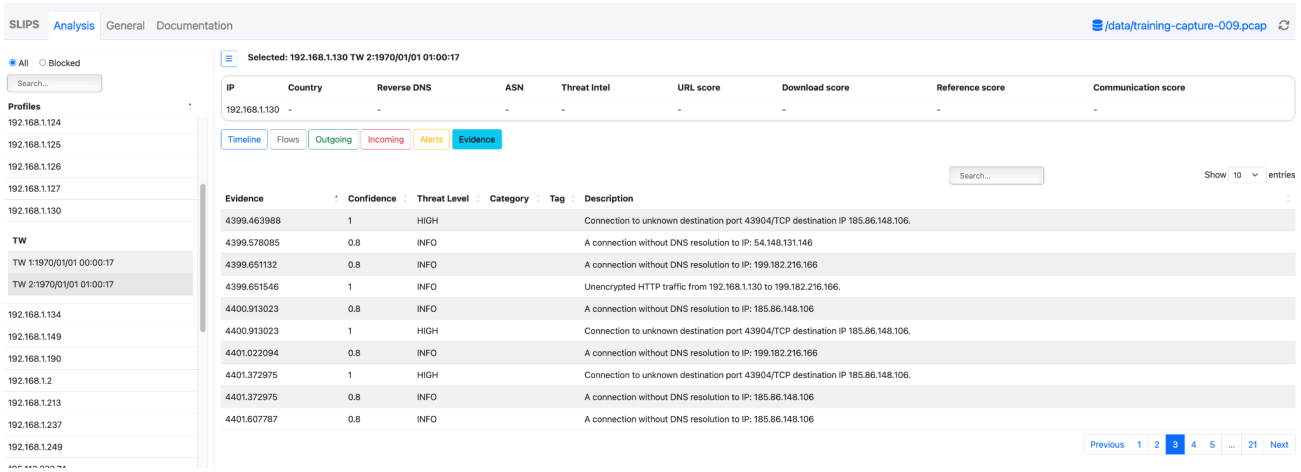
- If you installed the web interface later.
- You need to do this from your **host/main computer**, meaning from outside SCL to your **hackerlab** docker.
- `ssh -L 55000:localhost:55000 root@172.20.0.2 -p <YourPort>`

All students

To access the webpage, use a browser in your main computer on your personal computer to go to:

- `http://127.0.0.1:55000/`

Here is an example output of Slips Web UI:



Stop the web server

- `ps afx|grep webinterface`
- `kill -9 <pid>`

Assignment 9 (5 Points) (17:42, 3m)

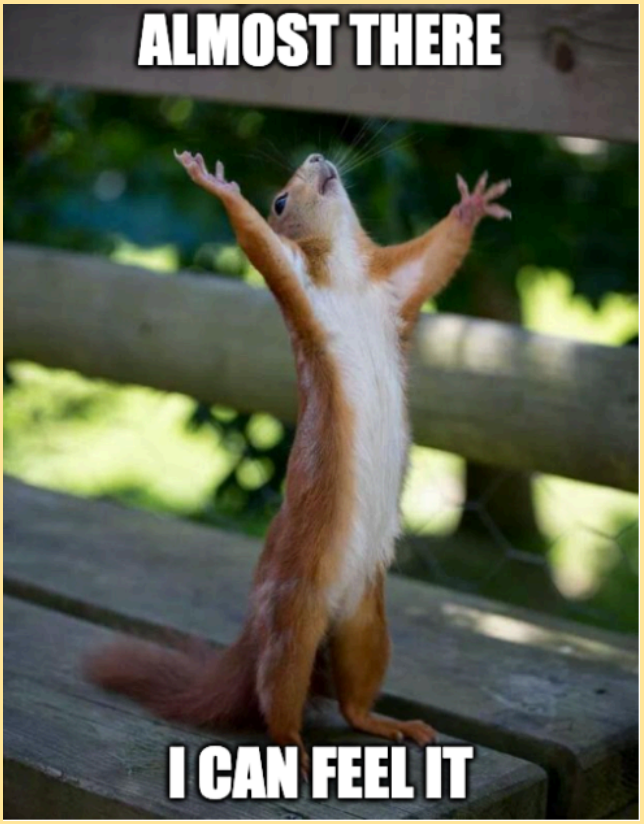
The assignment 9 has two parts:

Part I:

1. Download a PCAP from CTFd
2. You need to answer the questions in CTFd

Part II:

3. Go to CTFd
4. Read the challenge for the second part
5. Modify the ML part in the class Colab according to the instructions.
6. Go to CTFd and answer the question



Class Feedback

By giving us feedback after each class, we can make the next class even better!

bit.ly/BSYFeedback