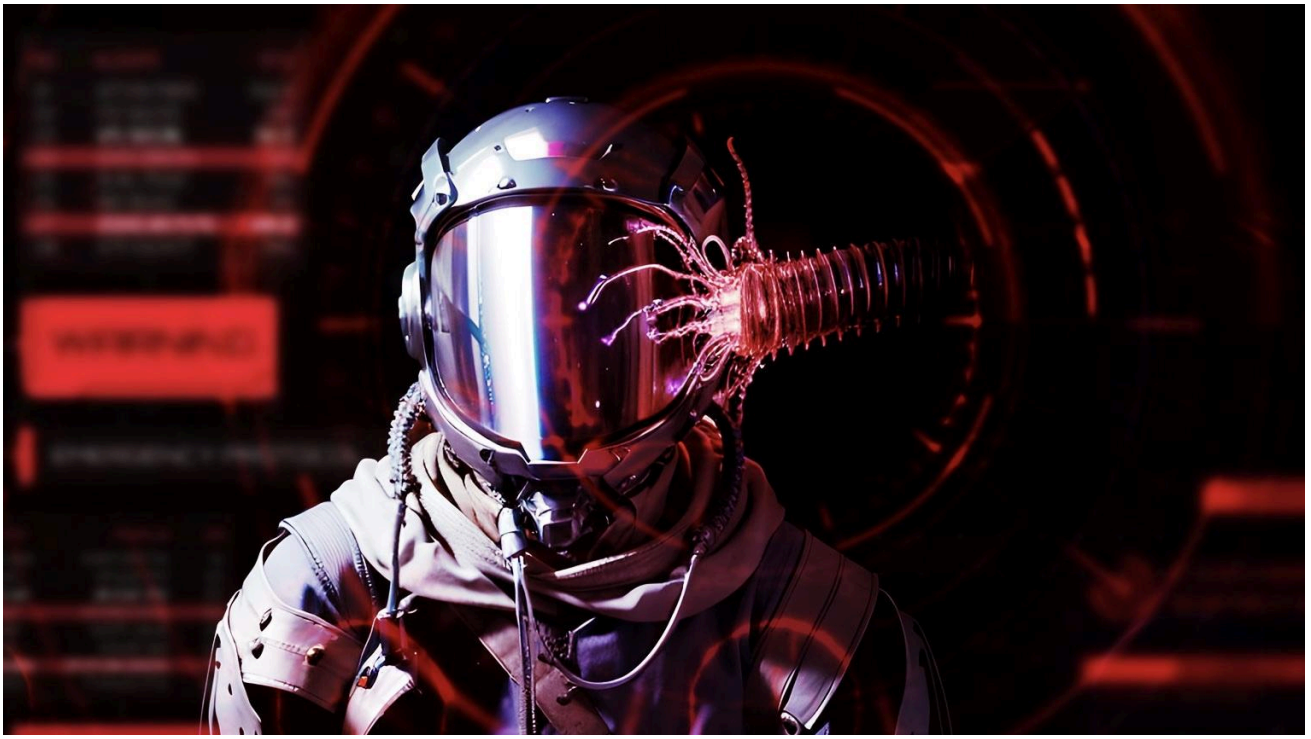
MANUAL AND AUTOMATIC DETECTION OF C&C CHANNELS



December 4, 2025

"The one where we automate all the things"

Credits

Content: Sebastian Garcia, Veronica Valeros, Maria Rigaki
Martin Řepa, Lukáš Forst, Ondřej Lukáš, Muris Sladić
Illustrations: Fermin Valeros
Introduction theme video: Art: Fermin Valeros, Production: Veronica 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

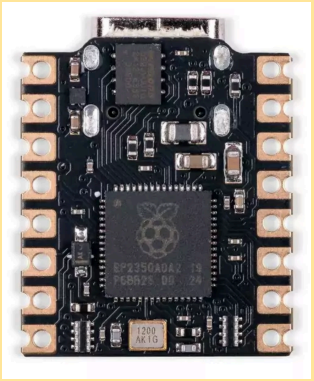
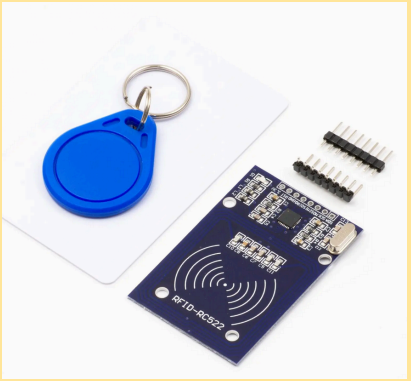
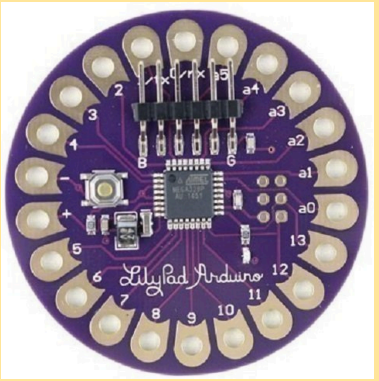
CLASS DOCUMENT	https://bit.ly/BSY2025-11
WEBSITE	https://cybersecurity.bsy.fel.cvut.cz/
MATRIX	https://matrix.bsy.fel.cvut.cz/
CTFD (CTU STUDENTS)	https://ctfd.bsy.fel.cvut.cz/
PASSCODE FORM (MOOC STUDENTS)	https://bit.ly/BSY-MOOCPasscode
FEEDBACK	https://bit.ly/BSY-Feedback
LIVESTREAM	https://bit.ly/BSY-Livestream
INTRO Animation	https://bit.ly/BSY-IntroTheme Or you can give us a like on YouTube 😉 Official Intro Theme — CTU Introduction to Se...
VIDEO RECORDINGS PLAYLIST	https://bit.ly/BSY-Recordings

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

How was the pace of the class?

19 responses

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

1st Place	2nd Place	3rd Place
		
Pimoroni Tiny 2350	RFID reader RC522	Arduino Lilypad 328P
Ondřej (vasaton3)	**Max (hollmmax)**	**David (seredda1)**

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

- **CTU Students' exam dates are confirmed as follows:**
    - Tuesday, January 20th. From 15:00 - 17:00. KN-107
    - Thursday, January 22rd, From 14:30 - 16:30. KN-107
    - Thursday, January 29th. From 14:30 - 16:30. KN-107
    - Thursday, February 5th. From 14:30 - 16:30. KN-107
    - You will have to register for the exams in KOS. Limited space per date.
- CTU Bonus Assignment:
    - It will open on **December 19th, 2025, at 21:00 CET.**
    - The deadline is **January 7th, 2026, at 23:59:59**.
    - You will know by December 18th the latest if you unlocked the report
    - **Bonus report submissions are final and cannot be edited**. Please review the feedback on your unlock reports beforehand to avoid recurring mistakes.
    - Yes, even if you submit the bonus report before the deadline, the submission is final.
    - More details to come soon. Next class.
- **MOOC students**: click 'Start' in the StratoCyberLab dashboard for Class 11

Class outline

- Manual Detection of C&C Channels in the Network
- Traffic Analysis with Zeek
- Automatic Detection of Command and Control and attacks with Machine Learning
- Slips IDS

Manual Detection of Command and Control (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 (often the only confirmation).
- Knowing if the malware is **autonomous** or **controlled** (which is much worse).
- Determining whether the malware was successful or not.
- Determining whether the malware is active or dormant.

However, detecting C&C communications is hard because they sometimes look too similar to real, benign communications. Experience is key. Context is critical.

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

Download the PCAP **class11-capture1.pcapng** on your computer:

1. CTU Students: Host Computer / MOOC Students: Host Computer
    a. Download from MEGA
    b. Download from Google Drive

Open **class11-capture1.pcapng** in Wireshark. You should see something like this:

No.	Time	Source	sPort	Destination	dPort	Protocol	Length	Host	Info
1		192.168.1.196	59934	104.248.160.24	80	TCP	74		59934 → 80 [SYN] Seq=0 Win=29200 Len
2	1.079387	192.168.1.196	59934	104.248.160.24	80	TCP	74		[TCP Retransmission] 59934 → 80 [SYN
3	0.024982	104.248.160.24	80	192.168.1.196	59934	TCP	74		80 → 59934 [SYN, ACK] Seq=0 Ack=1 Wi
4	0.000250	192.168.1.196	59934	104.248.160.24	80	TCP	66		59934 → 80 [ACK] Seq=1 Ack=1 Win=293
5	0.000250	192.168.1.196	59934	104.248.160.24	80	HTTP	215	104.24…	GET /ntpd HTTP/1.1
6	0.030483	104.248.160.24	80	192.168.1.196	59934	TCP	66		80 → 59934 [ACK] Seq=1 Ack=150 Win=1
7	0.001509	104.248.160.24	80	192.168.1.196	59934	TCP	2962		80 → 59934 [ACK] Seq=1 Ack=150 Win=1
8	0.000239	192.168.1.196	59934	104.248.160.24	80	TCP	66		59934 → 80 [ACK] Seq=150 Ack=1449 Wi
9	0.000249	192.168.1.196	59934	104.248.160.24	80	TCP	66		59934 → 80 [ACK] Seq=150 Ack=2897 Wi
10	0.010748	104.248.160.24	80	192.168.1.196	59934	TCP	1514		80 → 59934 [ACK] Seq=2897 Ack=150 Wi
11	0.000247	192.168.1.196	59934	104.248.160.24	80	TCP	66		59934 → 80 [ACK] Seq=150 Ack=4345 Wi
12	0.011744	104.248.160.24	80	192.168.1.196	59934	TCP	1514		[TCP Previous segment not captured]
13	0.000248	192.168.1.196	59934	104.248.160.24	80	TCP	78		[TCP Dup ACK 11#1] 59934 → 80 [ACK]

💡 You can review Class 2 for a refresher on how to configure the columns in Wireshark: Class 2 - Finding computers, scanning and basic network analysis [...

This capture contains 1,174 packets (Menu → Statistics → Capture File Properties):

Statistics

Measurement	Captured	Displayed	Marked
Packets	1174	1174 (100.0%)	—
Time span, s	459.311	459.311	—

Analysis Methodology: Clean, Rinse, Repeat

In most analysis we need to review everything on the capture to make sure all the behaviors are identified. We will go packet by packet[1].

To make sure we review everything, we will follow the clean, rinse, repeat methodology 🧽:

- 🕵🏽 Identify the first network connection
- 🔎 Analyse if it is benign or not
- 👜 Filter out what we know (benign or malicious)
- ↩ Repeat the process until we have analysed all the network connections

We will use **Wireshark streams** to guide our analysis:

- Easy identification of every connection
- Very clear to share the analysis with colleagues
- Let's add a new custom column to see the stream each packet belongs to:
    - Right-click on the columns bar → 'Column Preferences' → 'Add new column' → Change name to Stream → Change Type to 'custom' → Change field to tcp.stream or udp.stream
    - Move (drag and drop) the new column below the column 'Protocol'.
    -

Source	sPort	Destination	dPort	Protocol	Stream	Length
192.168.1.196	59934	104.248.160.24	80	TCP	0	74
192.168.1.196	59934	104.248.160.24	80	TCP	0	74
104.248.160.24	80	192.168.1.196	59934	TCP	0	74
192.168.1.196	59934	104.248.160.24	80	TCP	0	66
192.168.1.196	59934	104.248.160.24	80	HTTP	0	215
104.248.160.24	80	192.168.1.196	59934	TCP	0	66
104.248.160.24	80	192.168.1.196	59934	TCP	0	2962
192.168.1.196	59934	104.248.160.24	80	TCP	0	66

Step-by-Step Analysis

By the end of this analysis, we will have reviewed all 1,174 packets! Let's go!

[1] Review Class 1 for understanding known protocols: Class 1 - Introduction to the Class, Security and Networking [2025.09.25]

1. **Identify:** tcp.stream==0
    - **Analyse**
        - What is this connection?
            - 192.168.1.196 ←→ 104.248.160.24 (80/TCP)
            - Seems an HTTP download
        - Anything suspicious about the content? Follow stream
            - GET /ntpd HTTP/1.1
            - User-Agent: Wget/1.18 (linux-gnueabihf)
            - Content-Length: 119167
            - .ELF.....@.....4
        - Export objects, check hash of the file:
            - Export objects → HTTP → ntpd → Save → stream0.donotclick
            - In your terminal:
                - md5sum stream-0.donotclick
                - 3be82bc4abd7fc0436463dac83c146c8
            - Check hash on Virus Total
            - Malicious file download
    - **Filter out**
        - Let's filter this out and continue: `!tcp.stream eq 0`
    - **Repeat**
        - Let's analyse the next stream
2. **Identify:** tcp.stream==1
    - **Analyse**
        - What is this connection?
            - 192.168.1.196 ←→ 104.248.160.24 (80/TCP)
            - A connection attempt, not established
        - Anything suspicious about the content? Follow stream
            - 1 SYN packet
            - 6 OS automatic retransmissions
        - We will ignore it:
            - No content transferred
            - No established connection
            - We already saw the dstIP on the previous stream
    - **Filter out**
        - Let's filter this out and continue: `!tcp.stream lt 2`
            - `lt ⇒ lower than`
            - Filters out TCP stream 0 and 1
            - Only use it if you are analyzing sequentially!
    - **Repeat**
        - Let's analyse the next stream
3. **Identify:** udp.stream==0

- **Analyse**
  - What is this connection?
    - 192.168.1.196←→ 192.168.1.1 (53/UDP)
    - Query for: 0.debian.pool.ntp.org
  - Anything suspicious about the content?
    - NTP → Network Time Protocol
    - OS Debian
    - OS is doing automatic time synchronization
  - We will ignore it:
    - Normal NTP behavior
    - Trusted NTP domain
- **Filter out**
  - Let's filter this out and continue:
    - `!tcp.stream lt 2 && !udp.stream lt 1 &&! dns.qry.name contains "ntp.org"`
      - Filters out TCP stream 0 and 1
      - Filters out UDP stream 0
      - Filters out all DNS query names to ntp.org
- **Repeat**
  - Let's analyse the next stream

4. **Identify:** arp (there are no streams!)
   - **Analyse**
     - What are these connections?
       - 192.168.1.1 is at 78:8a:20:43:93:d5
       - 192.168.1.196 is at b8:27:eb:8b:b1:2c
     - Anything suspicious about the content?
       - We see consistent mapping of IP and MAC addresses
       - There are no signs of tempering
     - We will ignore it:
       - No MAC claims multiple IPs
       - No IPs have multiple MAC claimants
       - No gratuitous ARPs or unsolicited announcements
   - **Filter out**
     - Let's filter this out and continue:
       - `!tcp.stream lt 2 && !udp.stream lt 1 && !dns.qry.name contains "ntp.org" && !arp`
   - **Repeat**
     - Let's analyse the next stream

5. **Identify:** udp.stream==1
   - **Analyse**
     - What is this connection?

        - 192.168.1.196 ←→ 89.221.210.188 (123/UDP)
        - NTP (Network Time Protocol) traffic
    - Anything suspicious about the content?
        - Normal request/respond pattern
        - Periodic pooling. Expected
    - We will ignore it:
        - No suspicious behavior observed
- **Filter out**
    - Let's filter this out and continue:
        - `!tcp.stream lt 2 && !udp.stream lt 2 && !dns.qry.name contains "ntp.org" && !arp && !ntp`
- **Repeat**
    - Let's analyse the next stream

6. **Identify:** tcp.stream==2 → TRY TO DO THE ANALYSIS ON YOUR OWN
    - **Analyse**
        - What is this connection?
            - 192.168.1.196 ←→ 104.248.160.24 (80/TCP)
            - Seems an HTTP download
        - Anything suspicious about the content? Follow stream
            - GET /sshd HTTP/1.1
            - User-Agent: Wget/1.18 (linux-gnueabihf)
            - Content-Length: 119167
            - .ELF.....
        - Should we flag it or not?
            - We already analysed one download and it was malicious
            - We can mark this as suspicious file download
    - **Filter out**
        - Let's filter this out and continue:
            - `!tcp.stream lt 3 && !udp.stream lt 2 && !dns.qry.name contains "ntp.org" && !arp && !ntp`
    - **Repeat**
        - Let's analyse the next stream

7. **Identify:** tcp.stream==3 → TRY TO DO THE ANALYSIS ON YOUR OWN
    - **Analyse**
        - What is this connection?
            - 192.168.1.196 ←→ 104.248.160.24 (80/TCP)
            - A connection attempt, not established
        - Anything suspicious about the content? Follow stream
            - 1 SYN packet
            - 6 OS automatic retransmissions
        - We will ignore it:

        - No content transferred
        - No established connection
        - We already saw the dstIP on the previous stream
    - **Filter out**
        - Let's filter this out and continue:
            - `!tcp.stream lt 4 && !udp.stream lt 2 && !dns.qry.name contains "ntp.org" && !arp && !ntp`
    - **Repeat**
        - Let's analyse the next stream

8. **Identify:** tcp.stream==4 → TRY TO DO THE ANALYSIS ON YOUR OWN
    - **Analyse**
        - What is this connection?
            - 192.168.1.196 ←→ 104.248.160.24 (80/TCP)
            - Seems an HTTP download
        - Anything suspicious about the content? Follow stream
            - GET /openssh HTTP/1.1
            - User-Agent: Wget/1.18 (linux-gnueabihf)
            - Content-Length: 82844
            - .ELF.....
        - Should we flag it or not?
            - We already analysed one download and it was malicious
            - We can mark this as suspicious file download
    - **Filter out**
        - Let's filter this out and continue:
            - `!tcp.stream lt 5 && !udp.stream lt 2 && !dns.qry.name contains "ntp.org" &&`
    - **Repeat**
        - Let's analyse the next stream

9. **Identify:** tcp.stream==5
    - **Analyse**
        - What is this connection?
            - 192.168.1.196 ←→ 104.248.160.24 (80/TCP)
            - Seems an HTTP download
        - Anything suspicious about the content? Follow stream
            - GET /bash HTTP/1.1
            - User-Agent: Wget/1.18 (linux-gnueabihf)
            - Content-Length: 88523
            - .ELF.....>
        - Should we flag it or not?
            - We already analysed one download and it was malicious
            - We can mark this as suspicious file download

- **Filter out**
  - Let's filter this out and continue:
    - `!tcp.stream lt 6 && !udp.stream lt 2 && !dns.qry.name contains "ntp.org" && !arp && !ntp`

10. **Identify:** tcp.stream==6
    - **Analyse**
      - What is this connection?
        - 192.168.1.196 ←→ 104.248.160.24 (80/TCP)
        - Seems an HTTP download
      - Anything suspicious about the content? Follow stream
        - GET /tftp HTTP/1.1
        - User-Agent: Wget/1.18 (linux-gnueabihf)
        - Content-Length: 117425
        - .ELF.....
      - Should we flag it or not?
        - We already analysed one download and it was malicious
        - We can mark this as suspicious file download
    - **Filter out**
      - Let's filter this out and continue:
        - `!tcp.stream lt 7 && !udp.stream lt 2 && !dns.qry.name contains "ntp.org" && !arp && !ntp`
11. **Identify:** tcp.stream==7
    - **Analyse**
      - What is this connection?
        - 192.168.1.196 ←→ 104.248.160.24 (80/TCP)
        - A connection attempt, not established
      - Anything suspicious about the content?
        - 1 SYN packet
        - 5 OS automatic retransmissions
      - We will ignore it:
        - No content transferred
        - No established connection
        - We already saw the dstIP on the previous stream
    - **Filter out**
      - Let's filter this out and continue:
        - `!tcp.stream lt 8 && !udp.stream lt 2 && !dns.qry.name contains "ntp.org" && !arp && !ntp`
    - **Repeat**
      - Let's analyse the next stream

12. **Identify:** tcp.stream==8
    - **Analyse**

        - What is this connection?
            - 192.168.1.196 ←→ 104.248.160.24 (80/TCP)
            - A connection attempt, not established
        - Anything suspicious about the content?
            - 1 SYN packet
            - 6 OS automatic retransmissions
        - We will ignore it:
            - No content transferred
            - No established connection
            - We already saw the dstIP on the previous stream
    - **Filter out**
        - Let's filter this out and continue:
            - `!tcp.stream lt 9 && !udp.stream lt 2 && !dns.qry.name contains "ntp.org" && !arp && !ntp`
    - **Repeat**
        - Let's analyse the next stream

13. **Identify:** tcp.stream==9
    - **Analyse**
        - What is this connection?
            - 192.168.1.196 ←→ 104.248.160.24 (23/TCP)
            - A connection attempt, not established
        - Anything suspicious about the content?
            - 1 SYN packet
            - 5 OS automatic retransmissions
        - We will ignore it:
            - No content transferred
            - No established connection
            - We already saw the dstIP on the previous stream
    - **Filter out**
        - Let's filter this out and continue:
            - `!tcp.stream lt 10 && !udp.stream lt 2 && !dns.qry.name contains "ntp.org" && !arp && !ntp`
    - **Repeat**
        - Let's analyse the next stream

14. **Identify:** tcp.stream==10
    - **Analyse**
        - What is this connection?
            - 192.168.1.196 ←→ 104.248.160.24 (23/TCP)
            - A connection attempt, not established
        - Anything suspicious about the content?
            - 1 SYN packet

        - 5 OS automatic retransmissions
    - We will ignore it:
        - No content transferred
        - No established connection
        - We already saw the dstIP on the previous stream
- **Filter out**
    - Let's filter this out and continue:
        - `!tcp.stream lt 11 && !udp.stream lt 2 && !dns.qry.name contains "ntp.org" && !arp && !ntp`
- **Repeat**
    - Let's analyse the next stream

15. **Identify:** tcp.stream==11
- **Analyse**
    - What is this connection?
        - 192.168.1.196 ←→ 104.248.160.24 (23/TCP)
        - TELNET connection
    - Anything suspicious about the content? Follow stream
        - Open connection, but not closed
        - Plain text content
        - Seems there are some instructions
    - Is this a C&C channel?
        - It appears that way, but we need to confirm
        - The infected machine is reporting a status
        - The remote server is answering with what appears to be an instruction
        - Does the infected machine follow the instruction?
- **Filter out**
    - Let's filter this out and continue:
        - `!tcp.stream lt 12 && !udp.stream lt 2 && !dns.qry.name contains "ntp.org" && !arp && !ntp`
- **Repeat**
    - Let's analyse the next stream

16. **Identify:** tcp.stream gt 11
- **Analyse**
    - What are these connections?
        - 192.168.1.196 ←→ ~165 dstIP addresses
        - Connection attempt to 23/TCP
        - An attempt, not established connection
    - Anything suspicious about the content?
        - 1 SYN packet per destination IP
        - The destination IPs seem 'programmed'

            - Each octet is increased by 1 with every new connection
            - All 165 connections happen in less than 1 second
        - This is malicious behavior:
            - Aggressing scanning to 23/TCP TELNET
            - Correlation with previously received instruction
            - This confirms the previous connection was the C&C
    - **Filter out**
        - Let's filter this out and continue:
            - `!tcp.stream lt 177 && !udp.stream lt 2 && !dns.qry.name contains "ntp.org" && !arp && !ntp`
    - **We have finished! No packet left without analysis!**

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 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 several connection attempts to an HTTP server (104.248.160.24 80/TCP); server unavailable
    - malicious several malicious file downloads from HTTP server (104.248.160.24 80/TCP); malicious ELF files
    - suspicious several connection attempts to an TELNET server (104.248.160.24 23/TCP); server unavailable
    - malicious TELNET (C&C) connections to server 104.248.160.24
    - malicious network service discovery (telnet) through horizontal scanning on dPort 23.
- Conclusion:
    - With the current information we know that host 192.168.1.196, running possibly Debian, got infected with malware.

- It is unclear how the first infection happened, the traffic capture shows the host was already infected.
- The malicious behavior matches classic IoT/Linux malware: download of ELF files, execution of the malicious file, contacting the C&C server, receiving instructions, carrying out instructions.
- OSINT reveals it is *probably*[2] a variant of BASHLITE malware: https://honeytarg.cert.br/honeypots/docs/papers/honeypots-sensors19.pdf

Recap: Manual analysis is a crucial skill. It is hard to learn, but this is the deepest you can go. Some malware/attacks can only be detected in this manner. Be methodical and take notes.

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

Please note that after the break, you will need a Google Account to access the Google Colab environment.

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

Although the network analysis with Wireshark was effective, it took us a considerable amount of time. And there are still additional findings to be made. How to 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."*[3,4]

What is a network flow? (15:50, 2m)

A network flow is a text record with the summarised information of all packets in the same communication or stream. One record per stream.

[2] Wikipedia, Words of Estimative Probability. https://en.wikipedia.org/wiki/Words_of_estimative_probability (accessed Dec 02, 2025)
[3] 'The Zeek Network Security Monitor', Zeek. https://zeek.org/ (accessed Dec. 05, 2024).
[4] Zeek Cheat Sheet, Corelight, https://f.hubspotusercontent00.net/hubfs/8645105/Corelight_May2021/Pdf/002_CORELIGHT_080420_ZEEK_LOGS_US_ONLINE.pdf

From Packets

```
1970-01-01 02:01:21.987838  IP 10.0.2.15.49170 > 54.72.9.51.80: Flags [S],   seq 2522050395                , length 0
1970-01-01 02:01:22.028551  IP 54.72.9.51.80 > 10.0.2.15.49170: Flags [S.],  seq 320001, ack 2522050396  , length 0
1970-01-01 02:01:22.028838  IP 10.0.2.15.49170 > 54.72.9.51.80: Flags [.],   ack 1                        , length 0
1970-01-01 02:01:22.029069  IP 10.0.2.15.49170 > 54.72.9.51.80: Flags [P.],  seq 1:269, ack 1             , length 268: HTTP: GET /login.php HTTP/1.1
1970-01-01 02:01:22.029518  IP 54.72.9.51.80 > 10.0.2.15.49170: Flags [.],   ack 269                      , length 0
1970-01-01 02:01:42.179969  IP 54.72.9.51.80 > 10.0.2.15.49170: Flags [P.],  seq 1:757, ack 269           , length 756: HTTP: HTTP/1.1 504 Gateway Time-out
1970-01-01 02:01:42.184855  IP 10.0.2.15.49170 > 54.72.9.51.80: Flags [P.],  seq 269:537, ack 757         , length 268: HTTP: GET /login.php HTTP/1.1
1970-01-01 02:01:42.186291  IP 54.72.9.51.80 > 10.0.2.15.49170: Flags [.],   ack 537                      , length 0
1970-01-01 02:02:02.291733  IP 54.72.9.51.80 > 10.0.2.15.49170: Flags [P.],  seq 757:1485, ack 537        , length 728: HTTP: HTTP/1.1 504 Gateway Time-out
1970-01-01 02:02:02.293802  IP 54.72.9.51.80 > 10.0.2.15.49170: Flags [F.],  seq 1485, ack 537           , length 0
1970-01-01 02:02:02.294067  IP 10.0.2.15.49170 > 54.72.9.51.80: Flags [.],   ack 1486                     , length 0
1970-01-01 02:02:02.294244  IP 10.0.2.15.49170 > 54.72.9.51.80: Flags [F.],  seq 537, ack 1486           , length 0
1970-01-01 02:02:02.294915  IP 54.72.9.51.80 > 10.0.2.15.49170: Flags [.],   ack 538                      , length 0
```

To Flows

ts	id.orig_h	id.orig_p	id.resp_h	id.resp_p	proto	duration	orig_bytes	resp_bytes	conn_state	orig_pkts	resp_pkts
81.987838	10.0.2.15	49170	54.72.9.51	80	tcp	40.306406	536	1484	SF	6	7

💡 There are many standards of network flows, among the most well known are NetFlow v5 (Cisco)[5], NetFlow v9 (Cisco)[6], IPFIX[7] and Zeek Flows[8].

Using Zeek (15:08, 15m)

Zeek interprets the traffic and produces rich, structured logs about what actually happened on the network. What can Zeek do?

- ***Capture*** live traffic directly from a network interface.
- ***Process*** offline pcaps just like the ones we used in Wireshark.
- ***Parse protocols*** (HTTP, DNS, SSL/TLS, SSH, FTP, SMTP, etc.)
- ***Reconstruct*** files transferred over the network.
- ***Run custom scripts*** to add detections, alerts, and threat intelligence.

MOOC Students: SCL HackerLab

- *You clicked 'Start Class 11' on the SCL dashboard*
- SSH to the victim FTP Server container (IP `172.20.0.101`; Password: `admin`)
    - `ssh -o UserKnownHostsFile=/dev/null -o StrictHostKeyChecking=no root@172.20.0.101`

MOOC Students: SCL HackerLab | MOOC Students: Class 11 Container

1. Create a special folder for the logs: you don't want all the files getting mixed.
    1. `mkdir zeek_logs`
    2. `cd zeek_logs`

5 NetFlow. Wikipedia.org. https://en.wikipedia.org/wiki/NetFlow (accessed Dec 02, 2025)

6 NetFlow. Wikipedia.org. https://en.wikipedia.org/wiki/NetFlow (accessed Dec 02, 2025)

7 What Is IPFIX and How Does It Work? FS.com. https://www.fs.com/blog/what-is-ipfix-and-how-does-it-work-8276.html (accessed Dec 02, 2025)

8 'The Zeek Network Security Monitor', Zeek. https://zeek.org/ (accessed Dec 02, 2025)

i. `zeek -Cr /data/class11-capture1.pcapng`

- -C → If PCAP checksums are bad, fix it and continue.
- -r → read a PCAP file

2. Let's check the logs created in zeek_logs
    1. `ls -alh ~/zeek_logs`
3. What are the log files?[9]
    1. conn.log[10]: TCP/UDP/ICMP connections flows[11] (no ARP)
        i. `cat conn.log | zeek-cut -d -M| column -t | less -S`
        - `zeek-cut -d`: Print human dates.
        - `zeek-cut -M`: Include all format header blocks in the output in minimal view.
    2. dns.log[9]: DNS activity
        i. `cat dns.log | zeek-cut -d -M| column -t | less -S`
    3. files.log[9]: File analysis results (e.g.: files downloaded and hashes)
        i. `cat files.log | zeek-cut -d -M| column -t | less -S`
    4. http.log[9]: HTTP requests and replies
        i. `cat http.log | zeek-cut -d -M| column -t | less -S`
    5. ntp.log[9]: 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[12]!

[9] 'Log Files — Book of Zeek (v5.1.0)'. https://docs.zeek.org/en/current/script-reference/log-files.html (accessed Dec. 01, 2025).
[10] conn.log, http.log, ssl.log - Book of Zeek, https://docs.zeek.org/en/current/scripts/base/protocols/conn/main.zeek.html (accessed Dec. 01, 2025).
[11] 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, 2025).
[12] Example: https://mcfp.felk.cvut.cz/publicDatasets/CTU-Maldisplays the entire history of state changes forware-Capture-Botnet-334-1/bro/

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.

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[13]:

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?

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

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 the Src IP and Dst IP (and ports) may be swapped. It indicates that what you believe is the **source IP (Src IP) is the destination IP (Dst IP)** and vice versa.

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

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

Extra: 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** has 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 number 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 print the destination IP ($5) and destination port ($6) in the matching lines.
   ```

g. Sort them, obtain unique combinations, and sort them by the number of times they appeared

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

Zeek has a beautiful programming language based on Lua[14] that is Turing-complete [15]. There is a package manager called zkg[16] that allows you to install many scripts created by the community.

1. Zeek can run various scripts to perform additional tasks. 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/class11-capture1.pcapng /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!**

Automatic Detection of Command and Control and attacks with Machine Learning (16:26, 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

[14] Lua: about. (2025). Lua.org. https://www.lua.org/about.html (accessed Dec. 01, 2025).
[15] Turing completeness, Wikipedia. https://en.wikipedia.org/wiki/Turing_completeness (accessed Dec. 01, 2025).
[16] Zeek.org. (2024). Zeek Package Manager. https://packages.zeek.org/ (accessed Dec. 01, 2025).

analyst labels.

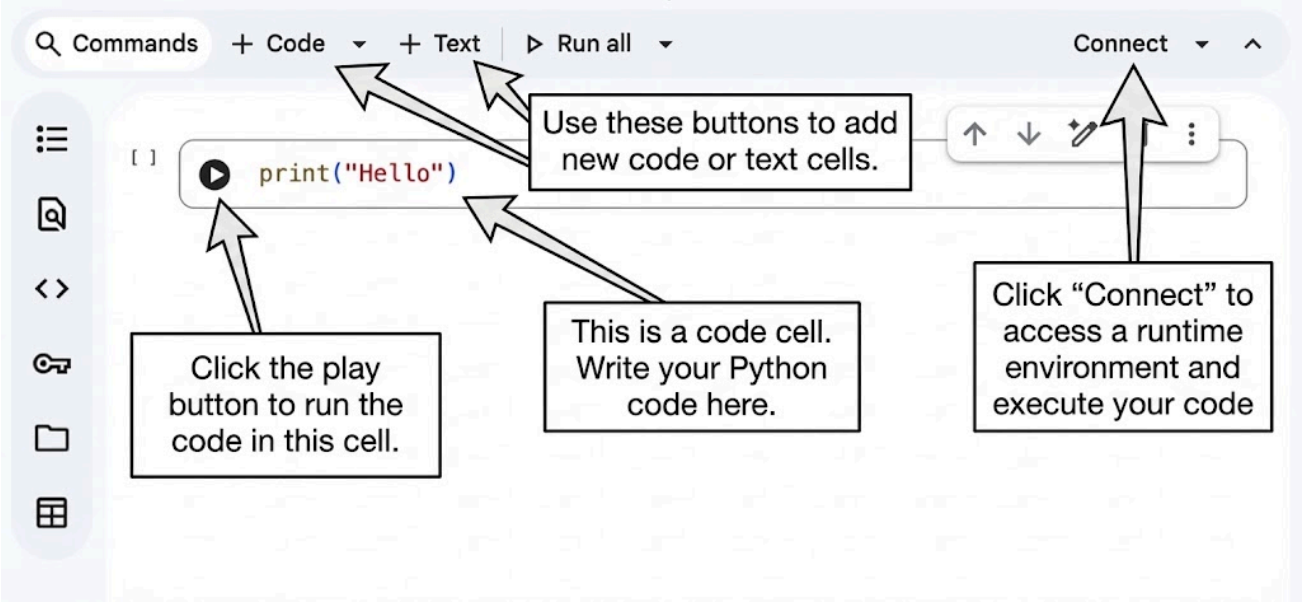
Google Colab

For this section, we will work with Python and Google Colab. You need a Google account to log in. CTU students can use their university accounts.

What is Google Colab? Short for Colaboratory, Google Colab is an online tool that lets you write and run code on Google's computers instead of your own.

"Colab is a hosted Jupyter Notebook[17] service that requires no setup to use and provides free access to computing resources, including GPUs and TPUs. Colab is especially well-suited to machine learning, data science, and education." - Google[18]

What does a Google Colab look like?

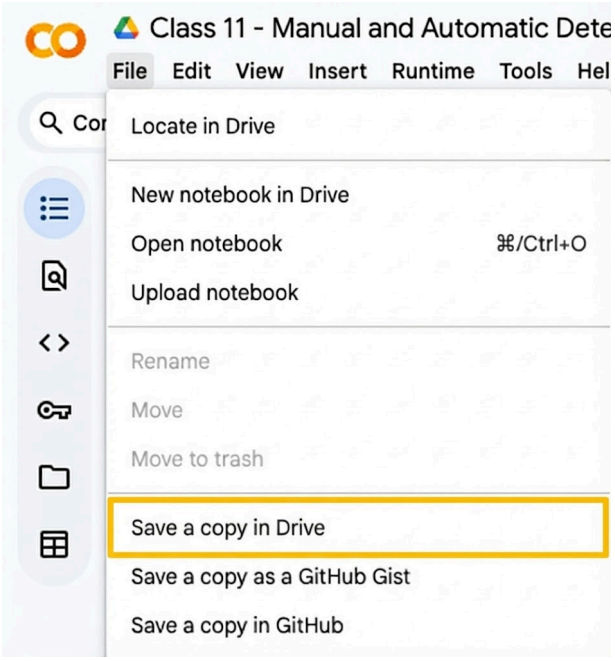


We have created a Google Colab Notebook for your use. We need you to **make a copy of this notebook. First, read through the instructions below and then follow the instructions:**

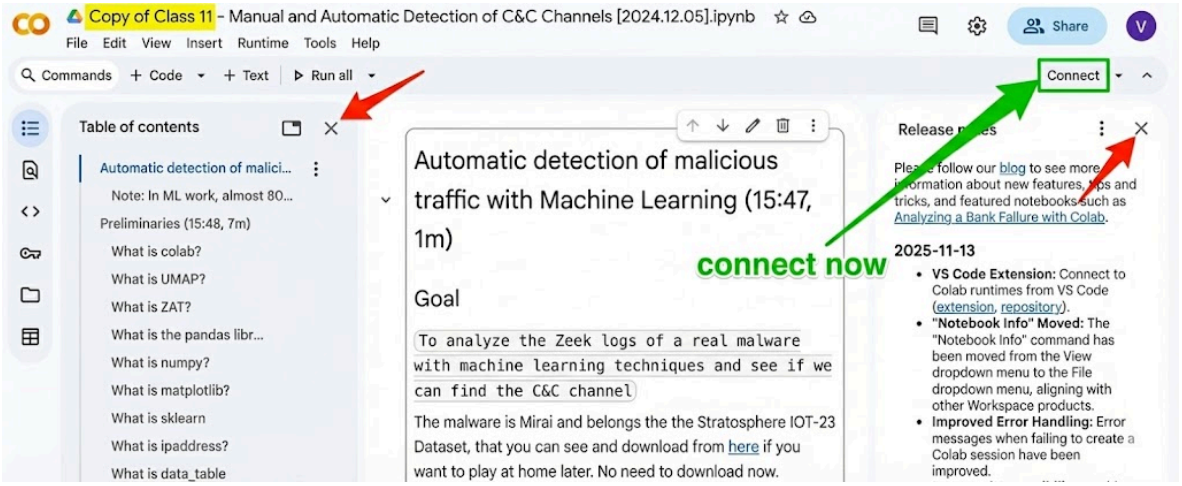
1. Access the notebook at:
   Class 11 - Manual and Automatic Detection of C&C Channels [2025.12.04].i...

[17] Project Jupyter. (2025). Jupyter.org. https://jupyter.org/ (accessed Dec. 01, 2025).
[18] colab.google. (2025). Colab.google. https://colab.google/ (accessed Dec. 01, 2025).

2. Go to Menu File → Save a copy in Drive:



3. The new copy will open automatically. Click 'connect' on the top right:



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

Slips IDS (17:27, 1m)

Slips[19] is a Python-based machine-learning tool created by our Stratosphere laboratory. It does not rely on signatures; instead, it analyzes network behaviors to detect malicious activities. It is the first free software machine learning-based IDS, which you can download and contribute to on GitHub:

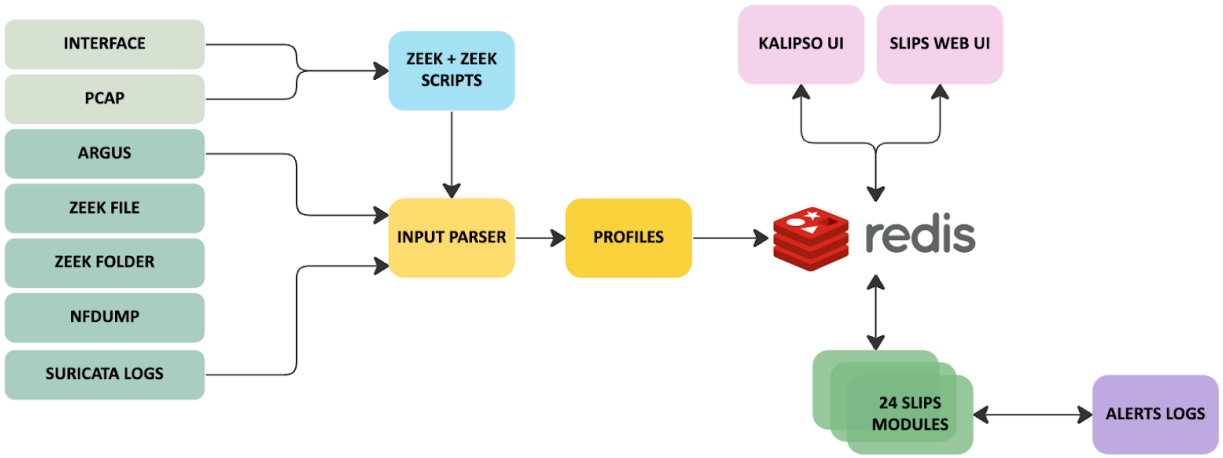
- https://github.com/stratosphereips/StratosphereLinuxIPS

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

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

All of this is possible today thanks to the collaboration of Alya Gomaa, the main Slips developer at 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:

ARP	BLOCKING	CESNET	CYST	DAEMON MODE	DETECT SMPT BRUTEFORCE	DETECT ICMP SCANS	GET RDNS OF IPS
ENSEMBLING	EXPORTING ALERTS	FLOW ALERTS	FLOW ML DETECTION	DETECT SSH BRUTEFORCE	DETECT DATA EXFILTRATION	DETECT INVALID CERTIFICATES	DETECT CC CHANNELS
HTTP ANALYZER	IP INFO	KALIPSO	LEAK DETECTOR	DETECT CONN. TO UNKNOWN PORTS	DETECTS DGA	ONLINE AND OFFLINE MACS RESOLUTION	EXPORT ALERTS TO SLACK
NETWORK DISCOVERY	P2P TRUST	PROGRESS BAR	RISK IQ	DETECT MULTIPLE USER AGENTS	DETECT SUCCESSFUL SSH ACCESS	DETECTS INCOMPATIBLE USER AGENTS	DOWNLOADS J3S FEEDS
RNN CC DETECTION	THREAT INTELLIGENCE	TIMELINE	UPDATE MANAGER	SUPPORTS ALLOWLISTS/W HITELISTS	DETECTS DOH FLOWS	DETECTS DNS WITHOUT RESOLUTION	DETECT BAD SMPT LOGINS
VIRUS TOTAL	IRIS	FIDES	SLIPS WEB UI	DETECTS YOUNG DOMAINS	DETECTS CONNECTIVITY CHECKS	DETECTS HORIZONTAL PORT SCANS	DETECTS VERTICAL PORT SCANS

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

Slips is already installed in the CTU dockers and SCL. These instructions are here so that you can reproduce later if you need:

- Shallow clone of the Slips repository:
  - `git clone --depth 1 https://github.com/stratosphereips/StratosphereLinuxIPS /StratosphereLinuxIPS`
  - `cd /StratosphereLinuxIPS`
- Install Zeek. Follow instructions from official source:
  - echo 'deb https://download.opensuse.org/repositories/security:/zeek/xUbuntu_22.04/ /' | sudo tee /etc/apt/sources.list.d/security:zeek.list
  - curl -fsSL https://download.opensuse.org/repositories/security:zeek/xUbuntu_22.04/Release.key | gpg --dearmor | sudo tee /etc/apt/trusted.gpg.d/security_zeek.gpg > /dev/null
  - sudo apt update
  - sudo apt install zeek-8.0
- Install Linux dependencies:
  - cat install/apt_dependencies.txt | xargs apt-get -y install
- Install Python dependencies:
  - python3 -m venv venv
  - source venv/bin/activate
  - `python3 -m pip install -r install/requirements.txt`

Lets Test that Slips is Running on your Containers

MOOC Students: SCL HackerLab

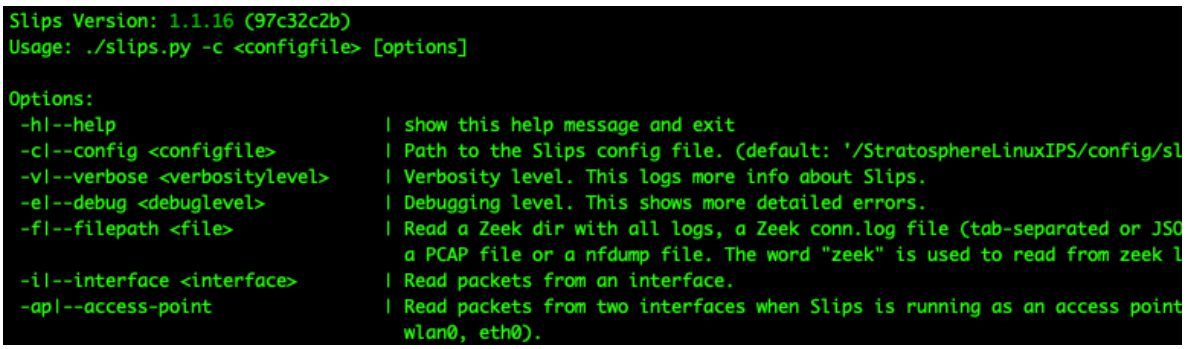
- *You clicked 'Start Class 11' on the SCL dashboard*
- SSH to the Class 11 Container (IP `172.20.0.101`; Password: `admin`)

- `ssh -o UserKnownHostsFile=/dev/null -o StrictHostKeyChecking=no root@172.20.0.101`

CTU Students: Docker container. MOOC Students: Class 11 Container

Let's check if Slips is working:

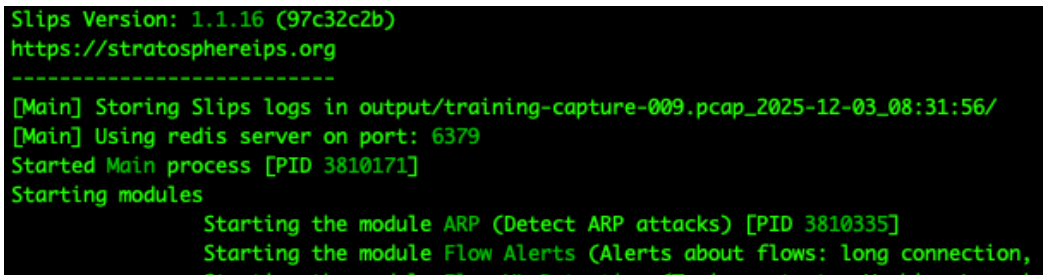
- `cd /StratosphereLinuxIPS`
- `source venv/bin/activate`
- `./slips.py -h`



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

Let's use Slips on the capture **/data/training-capture-009.pcap** available both on the CTU and SCL dockers.

- Let's change the default detection threshold to be more sensitive since the capture is small.
  - `sed -i 's/: 0\.25/: 0\.08/g' config/slips.yaml`
- To keep disk usage low, we have not installed tensorflow (~2GB). So we need to disable one of Slips modules that depends on it:
  - `sed -i "/disable:/,/]/ s/\]/, rnnccdetection]/" config/slips.yaml`
- Let's run Slips (it should take ~3 minutes or less in normal conditions):
  - `./slips.py -f /data/training-capture-009.pcap`
    - -f: The input file
    - Slips can ingest pcap, Zeek conn.log, Zeek folder, nfdump, Suricata JSON, and more.



- Checking the Slips output:
  - `less -S output/training-capture-009.pcap_2025-12-*/alerts.log`
  - If you don't have logs, you can use these copies:
    - CTU Students: Docker container.
      - `/data/slips-logs/training-capture-009`
    - MOOC Students: Host Computer
      - https://drive.google.com/file/d/1FfE4BPMgbzBnURhyavemzngUuHBeSFsl/view?usp=sharing

- **Extra:** You can summarise the logs if you are in the same folder as alerts.log:
  - `cat output/training-capture-009.pcap_2025-12-*/alerts.log | awk -F 'Detected' '{print $2}'|grep "threat level: high" |sort |uniq -c |sort -rn`
  - You can put this line in a script:
    - `bash /data/slips-alerts-sort.sh output/training-capture-009.pcap_2025-12-*/alerts.log`
- **Extra:** If Slips is still running, you can use the watch command to show the summary every 1 second:
  - `bash /data/slips-alerts-sort-monitor.sh output/training-capture-009.pcap_2025-12-*/alerts.log`

Why do Slips take long to stop?

Slips is a very complex tool, 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.

Conclusions

We have moved from raw packets in Wireshark, to flows and structured metadata in Zeek, and finally to automated analysis and threat detection with Slips.

Slips lets us automate the analysis further: instead of manually digging through packets or logs, we get high-level alerts, correlations, and threat classifications. It doesn't replace Wireshark or Zeek, it builds on them.

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

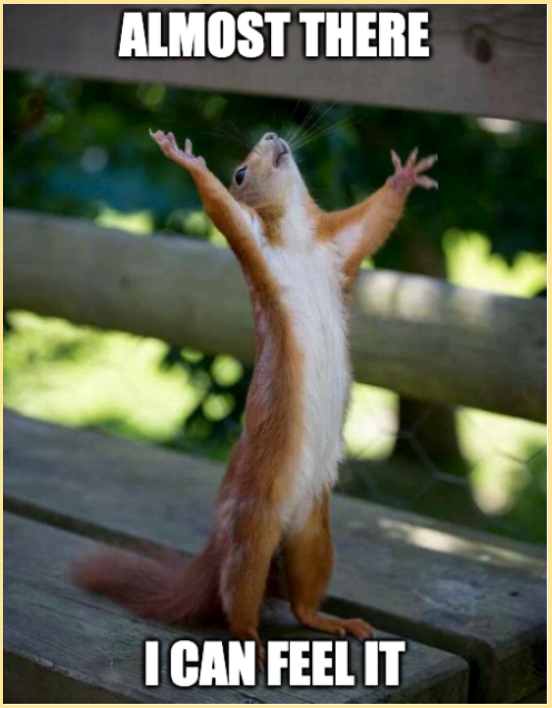
Part I:

1. Download the 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

DEADLINE: 07/01/2026 23:59:59



Class Feedback

By providing us with feedback after each class, you can help us make the next class even better!

https://bit.ly/BSY-Feedback