Project: Network Intrusion Detection with Suricata and Zeek

Secure Software Engineering and Management

Due date: Monday, April 20, 11:59pm
Format: Custom rules, analysis notes, incident report (all PDF), evasion analysis
Weight: Part of the Projects component (40% total across five projects)

Overview

In this project you will use two complementary network security tools — **Suricata** (a signature-based intrusion detection system) and **Zeek** (a network traffic analyzer) — to investigate a packet capture (PCAP) file containing a mix of normal and malicious network traffic.

The PCAP contains several distinct attack activities. Your job is to:

- run Suricata with its default open-source ruleset and see what it detects,
- use Zeek to build a fuller picture of network activity,
- identify malicious traffic that the default rules *missed*,
- write custom Suricata rules to detect the missed activity,
- produce a network incident report telling the story of what happened, and
- analyze how attackers could evade your custom rules.

This mirrors the workflow of a real Security Operations Center (SOC) analyst: receive alerts, investigate with additional tools, close detection gaps, report findings, and reason about the limits of your detections.

Learning Objectives

By completing this project, you will:

- deploy and configure an open-source IDS (Suricata) and a network analyzer (Zeek),
- analyze real network traffic for signs of compromise,
- understand what signature-based detection can and cannot catch,
- write custom IDS rules targeting specific attack behaviors,
- produce a professional-style incident report, and
- reason about detection evasion — the adversarial limits of signature-based rules.

What You Are Given

You are provided with the following (download from https://mhicks.me/courses/cis-7000-spring2026/downloads/project5-ids.zip):

- a PCAP file (`capture.pcap`) containing mixed benign and malicious network traffic, and
- a Docker Compose configuration that runs Suricata and Zeek against the PCAP in a reproducible environment.

The PCAP contains several kinds of attack activity. The next section describes attack patterns you may encounter and what to look for. The default ET Open ruleset does not provide complete coverage of everything in the capture — finding and explaining those gaps is part of the project.

Background: Attack Types in the PCAP

The PCAP contains traffic from several attack types. You have not studied these attacks in detail in this course, so this section provides the background you need to recognize them in network traffic. You are encouraged to read the linked references for deeper understanding.

Command-and-Control (C2) Beaconing

After malware compromises a host, it needs to communicate with the attacker's server to receive instructions and send stolen data. This communication is called **command-and-control** (C2). A common C2 pattern is **beaconing**: the infected host periodically "checks in" with the C2 server at regular or semi-regular intervals.

What it looks like in network traffic:

- Repeated outbound connections from the same internal host to the same external IP or domain, often at roughly regular intervals (e.g., every 30 seconds, every 5 minutes).
- The connections may use HTTP or HTTPS. HTTP beacons often use GET requests to the same URI path, sometimes with encoded parameters. HTTPS beacons will show up as repeated TLS connections to the same server.
- The responses may be small and uniform in size ("no new instructions") with occasional larger responses ("here is a command").
- The User-Agent string may be unusual, absent, or a known malware default.

What to look for in Zeek logs:

- In `conn.log`: repeated connections from one internal IP to one external IP, with similar durations and byte counts. Sort by source IP and destination IP to spot repetition.
- In `http.log`: repeated requests to the same host and URI path. Look at the `user_agent`, `host`, `uri`, and `resp_body_len` fields.
- In `ssl.log`: repeated TLS connections to the same SNI (Server Name Indication) value.

Further reading:

- MITRE ATT&CK technique T1071 (Application Layer Protocol): https://attack.mitre.org/techniques/T1071/
- MITRE ATT&CK technique T1573 (Encrypted Channel): https://attack.mitre.org/techniques/T1573/
- "Detecting Beaconing Activity" section of the SANS Reading Room paper "Finding Beacons in the Dark": https://www.sans.org/white-papers/36667/

Exploit Kit Traffic

An **exploit kit** is a pre-packaged toolkit hosted on a web server that automatically exploits vulnerabilities in a victim's browser or browser plugins. The typical chain is:

1. The victim visits a legitimate website that has been compromised (or clicks a malicious link).
2. The compromised page contains a hidden redirect (often via an iframe or injected JavaScript) to the exploit kit's **landing page**.
3. The landing page probes the victim's browser to identify vulnerabilities.
4. If a vulnerability is found, the exploit kit serves a **payload** (malware executable).
5. The victim's browser downloads and executes the payload, compromising the host.

What it looks like in network traffic:

- A chain of HTTP requests: first to a legitimate-looking site, then a redirect to an unfamiliar domain, then one or more requests to the exploit kit server.
- The landing page URL often contains long, randomly-generated path components or query parameters.
- A file download (the payload) from the exploit kit server, often an executable (`.exe`) or obfuscated content with a misleading MIME type.
- The referring chain may be visible in HTTP `referer` headers, though in some captures you may need to reconstruct the chain from timing, hosts, URIs, and response metadata instead.

What to look for in Zeek logs:

- In `http.log`: look for sequences of requests from the same host that move from one site to unfamiliar domains and then to executable or payload-like content. If the `referrer` field is populated, use it; otherwise reconstruct the chain from timestamps, hosts, URIs, response sizes, and MIME types.
- In `files.log`: files transferred over HTTP with executable MIME types (`application/x-dosexec`, `application/octet-stream`) or mismatched extensions. The `md5` and `sha256` fields can be checked against threat intelligence databases like VirusTotal.
- In `conn.log`: short-lived connections to IPs or domains that appear nowhere else in the capture, especially if they immediately precede the start of C2 beaconing.

Further reading:

- MITRE ATT&CK technique T1189 (Drive-by Compromise): https://attack.mitre.org/techniques/T1189/
- "Exploit Kit" article on the Malwarebytes glossary: https://www.malwarebytes.com/glossary/exploit-kit

DNS Tunneling

DNS tunneling is a technique that encodes data inside DNS queries and responses, using the DNS protocol as a covert communication channel. Because DNS traffic is almost always allowed through firewalls, attackers use it to exfiltrate data or maintain C2 communication even when other protocols are blocked.

The basic mechanism: the attacker controls a domain (e.g., `evil.com`) and runs a custom DNS server for it. The malware on the victim encodes data as a subdomain label in a DNS query, such as `encoded-stolen-data.evil.com`. The attacker's DNS server receives the query, decodes the data, and can send data back in the DNS response (typically in TXT or CNAME records).

What it looks like in network traffic:

- An unusually high volume of DNS queries from a single internal host.
- DNS queries with abnormally long subdomain labels (legitimate subdomains are typically short; tunneling produces labels of 30–60+ characters of encoded data).
- Many queries for the same parent domain but with constantly changing subdomain prefixes (e.g., `aGVsbG8gd29ybGQ.evil.com`, `dGhpcyBpcyBkYXRh.evil.com`).
- Query types that are unusual for normal browsing, especially TXT records.
- DNS responses that are larger than typical (especially TXT responses carrying encoded return data).

What to look for in Zeek logs:

- In `dns.log`: sort by `query` length. Tunneling queries will be noticeably longer than normal queries (e.g., `www.google.com` is 14 characters; a tunneling query might be 60–100+ characters). Look at the `qtype_name` field for TXT queries. Group queries by the base domain (the last two labels, e.g., `evil.com`) and look for domains with an unusually high number of unique subdomains.
- In `conn.log`: look for a single internal host with hundreds or thousands of DNS connections in a short period.

Further reading:

- MITRE ATT&CK technique T1071.004 (Application Layer Protocol: DNS): https://attack.mitre.org/techniques/T1071/004/
- SANS InfoSec Reading Room, "Detecting DNS Tunneling": https://www.sans.org/white-papers/35302/
- The `iodine` and `dnscat2` tools are commonly used DNS tunneling implementations. Reading their documentation helps understand what the traffic looks like.

Lateral Movement

Once an attacker has compromised one host inside a network, they often try to move to other internal hosts to expand their access. This is called **lateral movement**. Common techniques include:

- Using stolen credentials to authenticate to other machines via SMB (Windows file sharing) or SSH.
- Exploiting vulnerabilities in internal services that are not exposed to the internet.
- Using remote administration tools (PsExec, WMI, PowerShell Remoting) to execute commands on other hosts.

What it looks like in network traffic:

- Internal-to-internal connections on ports associated with remote access: SMB (TCP 445), SSH (TCP 22), RDP (TCP 3389), WinRM (TCP 5985/5986).
- An internal host that was previously only a client suddenly acting as a client to other internal hosts on administrative ports.
- Failed authentication attempts (visible in some protocols) followed by successful ones, suggesting credential guessing or credential reuse.
- File transfers between internal hosts (e.g., copying a malware executable via SMB).

What to look for in Zeek logs:

- In `conn.log`: filter for connections where both `id.orig_h` and `id.resp_h` are internal addresses, especially on ports 445, 22, 3389, 5985. Look for hosts that appear as connection *originators* to many other internal hosts in a short time.
- In `smb_mapping.log` and `smb_files.log` (if present): SMB share access and file operations between internal hosts.
- In `ssh.log` (if present): SSH connections between internal hosts, especially from a host that was the target of earlier attack activity.
- In `files.log`: files transferred over SMB between internal hosts, particularly executables.

Further reading:

- MITRE ATT&CK Tactic TA0008 (Lateral Movement): https://attack.mitre.org/tactics/TA0008/
- MITRE ATT&CK technique T1021.002 (Remote Services: SMB/Windows Admin Shares): https://attack.mitre.org/techniques/T1021/002/

Setup

You will run both Suricata and Zeek inside Docker containers to ensure a consistent environment. You need Docker Desktop (Mac/Windows) or Docker Engine (Linux) installed and running.

Step 1: Get the Project Files

Download the project package from Canvas and extract it. The package contains:

- `capture.pcap` — the network capture to analyze
- `docker-compose.yml` — orchestrates Suricata and Zeek containers
- `suricata/` — Suricata configuration files
- `custom-rules/` — directory where you will place your custom Suricata rules
- `output/` — directory where analysis output will be written

Step 2: Verify Prerequisites

Confirm the following work in a terminal:

```
docker --version
docker compose version
```

If `docker compose` does not work, try `docker-compose` (with a hyphen). Use whichever form works on your system throughout this project.

Step 3: Run Suricata with Default Rules

From the project root directory, run:

```
docker compose up suricata
```

This will:

- pull the Suricata Docker image (first run only),
- download the ET Open ruleset via `suricata-update`,
- run Suricata against `capture.pcap`, and
- write output to `output/suricata/`.

When Suricata finishes (it exits after processing the PCAP), you will find:

- `output/suricata/eve.json` — the main output file, containing alerts and protocol metadata in JSON format (one JSON object per line).
- `output/suricata/fast.log` — a human-readable summary of alerts.

To view alerts quickly:

```
# Count total alerts
grep -c "\"event_type\":\"alert\"" output/suricata/eve.json

# View alert summaries
cat output/suricata/fast.log

# Pretty-print a specific alert (requires jq)
grep "\"event_type\":\"alert\"" output/suricata/eve.json | head -1 | jq .
```

`jq` is a command-line JSON processor. If you do not have it installed, you can install it via your package manager (e.g., `brew install jq` on macOS, `apt install jq` on Ubuntu) or read the raw JSON directly.

Step 4: Run Zeek

From the project root directory, run:

```
docker compose up zeek
```

This will run Zeek against the same PCAP and produce log files in `output/zeek/`. The key log files are:

Log File	Contents
`conn.log`	Every network connection (IPs, ports, bytes, duration)
`dns.log`	DNS queries and responses
`http.log`	HTTP requests and responses
`ssl.log`	TLS handshake details (SNI, certificates)
`files.log`	Files transferred (MIME type, size, hashes)
`weird.log`	Unusual or malformed protocol events

Zeek's log files are tab-separated. The first line beginning with `#fields` lists the column names. To extract specific fields, you can use `zeek-cut` (available inside the Zeek container) or standard Unix tools:

```
# Extract source IP, dest IP, dest port, and protocol from conn.log
docker compose run --rm zeek zeek-cut id.orig_h id.resp_h \
    id.resp_p proto < output/zeek/conn.log

# Or use awk directly (field positions depend on the log):
awk "/^[^#]/" output/zeek/conn.log | head -20
```

Step 5: Run Suricata with Custom Rules

After writing your custom rules (see Part 2 below), place them in a file called `custom-rules/local.rules`. Then re-run Suricata with custom rules included:

```
docker compose up suricata-custom
```

This runs Suricata with both the default ET Open rules and your custom rules. Check `output/suricata-custom/eve.json` and `fast.log` for your custom alerts. Your custom rule SIDs should be in the range 9000001–9099999 to avoid collisions with ET rules.

Debugging Common Issues

- **Docker not running:** Start Docker Desktop or the Docker daemon, then re-run the commands.
- **"No such service" error:** Make sure you are in the project root directory (where `docker-compose.yml` is located).
- **Empty output files:** Check that `capture.pcap` is in the project root. Re-run `docker compose up suricata`.
- **Permission errors on output files:** Run `chmod -R u+rw output/`.
- **Disk space errors:** Run `docker system prune -a -f` and retry.

Part 1: Analyze the PCAP

Task 1: Analyze Default Suricata Alerts

Run Suricata with the default ET Open ruleset (Step 3 above). Examine the alerts in `fast.log` and `eve.json`.

For each alert, record:

- the alert message and SID (signature ID),
- the source and destination IPs and ports,
- the timestamp, and
- which attack type from the Background section the alert corresponds to (if any).

Group the alerts by attack type. Note which attack types generated alerts and which did not.

Task 2: Investigate with Zeek

Run Zeek (Step 4 above) and use its logs to build a more complete picture of the network activity. Your goal is to identify malicious traffic that Suricata's default rules did *not* alert on.

Start with the Suricata alerts as leads — for each alerted IP address or connection, look at what *else* that host did in Zeek's `conn.log`, `dns.log`, `http.log`, and `files.log`. Then look for the attack patterns described in Background section that did not produce any Suricata alerts.

Suggested approach:

1. Identify the internal hosts involved in Suricata alerts. For each, list all connections that host made (from `conn.log`).
2. Look for beaconing patterns: repeated connections to the same destination at regular intervals.
3. Look for DNS anomalies: unusually long queries, high query volume from a single host, many unique subdomains under one parent domain.
4. Look for internal-to-internal connections on administrative ports (445, 22, 3389) that might indicate lateral movement.
5. Look for suspicious file downloads in `files.log` and `http.log`: executable MIME types, unusual file sizes, downloads from hosts involved in other alerts.

Document your findings. For each suspicious activity you identify, note:

- what you observed (specific log entries or patterns),
- which attack type it corresponds to,
- which hosts were involved, and
- whether Suricata's default rules detected it.

Hints for Your Investigation

The following hints will help orient your analysis. They do not tell you what the attacks are or what rules to write — that is your job — but they narrow the search space so you can focus on analysis rather than guessing.

- The internal network is `10.0.0.0/24`. The gateway and DNS server is `10.0.0.1`. There is at least one other internal host in the capture.
- Default ET Open rules provide incomplete coverage. Some malicious activity is surfaced clearly, some is only weakly signaled, and some requires Zeek-based investigation to recognize confidently.
- Not all default alerts are equally informative. Some alerts are labeled `ET INFO` (informational) rather than `ET MALWARE` or `ET HUNTING`. Pay attention to the alert *category*: an `INFO` alert tells you something happened but does not necessarily flag it as malicious.
- Zeek produces several log files beyond the core ones. Check which log files were generated in `output/zeek/` — the *existence* of certain log files tells you what protocols Zeek observed. For example, if you see `smb_files.log` or `dce_rpc.log`, Zeek parsed SMB and DCE/RPC traffic.
- For DNS analysis: try sorting queries by length. Normal DNS queries (e.g., `www.google.com`) are short. If you see queries that are dramatically longer, investigate what domain they belong to and how many unique queries share the same parent domain.
- For finding beaconing: in `conn.log`, group connections by source–destination pair and count how many times each pair appears. A legitimate server might be contacted a few times; a C2 server will be contacted many times in a pattern.
- For lateral movement: filter `conn.log` for connections where *both* the source and destination are internal (`10.0.0.x`). What ports were used? What protocols does Zeek identify? Cross-reference with any specialized log files (like `smb_mapping.log`) to understand what happened.
- The timeline matters. Use timestamps from Suricata alerts and Zeek logs to reconstruct the order of events and test hypotheses about how different pieces of activity may be related.

Part 2: Write Custom Suricata Rules

Write **at least three** custom Suricata rules that detect attack activity present in the PCAP that the default ET Open rules did *not* detect or detected poorly. Each rule should target a **different** attack behavior or indicator — do not write three variations of the same rule. Ideally, your rules should cover at least two distinct attack types from Background section. Your rules should target specific malicious behaviors you identified in Part 1 using Zeek.

Suricata Rule Syntax Reference

A Suricata rule has a **header** and **options**:

```
action protocol source_ip source_port -> dest_ip dest_port (options)
```

Here is an annotated example:

```
alert http $HOME_NET any -> $EXTERNAL_NET any (
    msg:"CUSTOM Suspicious EXE download from unknown host";
    flow:established,to_client;
    content:"application/x-dosexec"; http_content_type;
    sid:9000001; rev:1;
)
```

Key rule keywords you are likely to need:

Keyword	Purpose
`msg`	Human-readable alert message (required)
`flow`	Match connection state and direction (e.g., `established,to_server`)
`content`	Match a byte string in the payload (text or hex with `|xx xx|`)
`nocase`	Make the preceding `content` match case-insensitive
`http_method`	Apply preceding `content` to the HTTP method (GET, POST, ...)
`http_uri`	Apply preceding `content` to the HTTP URI
`http_host`	Apply preceding `content` to the HTTP Host header
`http_user_agent`	Apply preceding `content` to the User-Agent header
`http_content_type`	Apply preceding `content` to the Content-Type header
`dns_query`	Apply preceding `content` to the DNS query name
`pcre`	Perl-compatible regular expression match
`threshold`	Alert only after N matches in T seconds
`sid`	Unique signature ID (use 9000001–9099999)
`rev`	Rule revision number

Example rules for common patterns:

```
# Detect a specific suspicious User-Agent string
alert http $HOME_NET any -> $EXTERNAL_NET any (
    msg:"CUSTOM Suspicious User-Agent";
    flow:established,to_server;
    content:"SuspiciousBot/1.0"; http_user_agent;
    sid:9000002; rev:1;
)

# Detect high-frequency DNS queries to a specific domain
alert dns $HOME_NET any -> any any (
    msg:"CUSTOM High-frequency DNS to suspicious domain";
    dns_query; content:"evil.example.com"; nocase;
    threshold:type threshold, track by_src,
              count 10, seconds 60;
    sid:9000003; rev:1;
)

# Detect DNS queries with unusually long names (possible tunneling)
alert dns $HOME_NET any -> any any (
    msg:"CUSTOM Possible DNS tunneling - long query name";
```

```
    dns_query;
    pcre:"/^[a-zA-Z0-9\-]{40,}\./";
    sid:9000004; rev:1;
)

# Detect internal SMB connections from a host that is not
# normally a file server (lateral movement indicator)
alert tcp $HOME_NET any -> $HOME_NET 445 (
    msg:"CUSTOM Internal SMB connection - possible lateral movement";
    flow:to_server;
    sid:9000005; rev:1;
)
```

These examples are **starting points only** and are intentionally generic. Your rules should be tailored to the specific malicious traffic you found in the provided PCAP. Do not copy these examples verbatim as your submission.

Rule Writing Guidelines

- Each rule must have a unique `sid` in the range 9000001–9099999.
- Each rule must have a descriptive `msg` that starts with `"CUSTOM"`.
- In `custom-rules/local.rules`, put each rule on a single line.
- Your rules should be targeted — a rule that alerts on all HTTP traffic or all DNS traffic is too broad to be useful.
- Prefer protocol-aware keywords (`http_uri`, `dns_query`, etc.) over raw `content` matching against the full packet payload. This reduces false positives and is faster.
- Test your rules by running `docker compose up suricata-custom` and checking that they produce alerts on the expected traffic in the PCAP.
- When using `flow` keywords, note that `flow:established,to_server` requires Suricata to have observed the TCP three-way handshake. Since the provided PCAP was assembled from multiple capture sources, some TCP connections may have been captured mid-stream (without the initial SYN/SYN-ACK/ACK). If a rule with `flow:established` does not fire when you expect it to, try `flow:to_server` instead.

Validation

After writing your rules, re-run Suricata with custom rules (Step 5) and verify:

1. Your custom rules fire on the malicious traffic you targeted.
2. Your custom rules do not produce an excessive number of false alerts on benign traffic in the PCAP.

Include evidence of validation in your writeup: show the alerts your custom rules generated and confirm they match the expected malicious activity.

Part 3: Detection Evasion Analysis

For each of your custom Suricata rules, write a short analysis (one paragraph per rule) answering:

1. **What specific feature does your rule match on?** For example: a domain name, a User-Agent string, a DNS query length threshold, a traffic volume threshold, a specific URI pattern.

2. **How could an attacker modify their behavior to evade this rule while still accomplishing their goal?** Be specific. For example: if your rule matches a particular C2 domain, the attacker could rotate domains; if your rule detects DNS queries longer than 40 characters, the attacker could use shorter encoding with more frequent queries.

3. **What would a more robust detection look like?** This could be a better Suricata rule, or it could be something that signature-based detection cannot express at all (e.g., statistical analysis of timing patterns, machine learning on traffic metadata, or correlation across multiple log sources). You do not need to implement this — just describe it.

The purpose of this section is to think critically about the limits of signature-based detection. As we discussed in lecture, the adversarial nature of security means that any fixed detection rule can potentially be evaded by an attacker who understands it. The Sommer & Paxson (2010) and Alahmadi et al. (2022) papers both highlight this fundamental challenge.

A strong evasion analysis will:

- identify the *specific* feature or assumption each rule depends on,
- propose a *concrete* evasion technique (not just "the attacker could change things"), and
- reflect on what class of detection (behavioral, statistical, multi-source correlation) would be harder to evade and why.

Part 4: Network Incident Report

Write a network incident report that tells the story of what happened in the PCAP. Your report should be structured as follows:

Executive Summary (1 paragraph)

A brief, non-technical summary: what happened, how severe is it, and what should be done immediately.

Timeline of Events

Reconstruct the sequence of attack activity. Use timestamps from Suricata alerts and Zeek logs. For each event, identify:

- what happened (e.g., "Host 10.0.0.5 began beaconing to 203.0.113.10 over HTTP"),
- the evidence (specific log entries, alert SIDs, Zeek log fields), and
- which attack stage this represents (initial compromise, C2 establishment, lateral movement, data exfiltration, etc.).

Affected Hosts

List each host involved in the attack activity. For each host, describe:

- its IP address and apparent role (client, server, domain controller, etc.),
- what malicious activity it was involved in, and
- whether it was a victim, an attacker-controlled server, or both.

Data at Risk

Based on the traffic you observed, what data may have been compromised or exfiltrated? This may require inference — for example, if you observe DNS tunneling, data was likely being exfiltrated even if you cannot decode the content.

Recommended Response Actions

What should the organization do in response? Structure your recommendations following the NIST incident lifecycle phases discussed in lecture:

- **Containment**: immediate actions to stop the attack (e.g., isolate hosts, block IPs).
- **Eradication**: actions to remove the attacker's presence (e.g., re-image hosts, revoke credentials).
- **Recovery**: actions to restore normal operations.
- **Lessons learned**: what detection or prevention improvements should be made.

Deliverables

Submit the following on Canvas as one zipfile:

- `custom-rules/local.rules` — your custom Suricata rules file (minimum 3 rules).
- `zeek-analysis.pdf` — your Zeek analysis notes from Part 1, Task 2. Document what you found, how you found it, and what commands or queries you used. Include relevant log excerpts.
- `rule-validation.pdf` — evidence that your custom rules fired correctly (Part 2 validation). Include the relevant alerts from `fast.log` or `eve.json`.
- `evasion-analysis.pdf` — your detection evasion analysis (Part 3).
- `incident-report.pdf` — your network incident report (Part 4).

You may combine the Zeek analysis notes, rule validation, and evasion analysis into a single PDF if you prefer, as long as each section is clearly labeled.

Do not submit the PCAP file, Docker images, or raw Zeek/Suricata output directories.

Grading

Component	Weight
Part 1: Suricata alert analysis (completeness, accuracy)	10%
Part 1: Zeek investigation (depth, methodology)	15%
Part 2: Custom rules (correctness, specificity, coverage)	20%
Part 2: Rule validation (evidence that rules work)	10%
Part 3: Evasion analysis (specificity, depth of reasoning)	20%
Part 4: Incident report (completeness, clarity, evidence)	25%

Correct rules are necessary but not sufficient. The quality of your analysis, your reasoning about detection evasion, and the clarity of your incident report are weighted heavily.

Academic Integrity

You may discuss high-level ideas with classmates (e.g., "I used Zeek's dns.log to find the tunneling"), but all analysis, custom rules, evasion analysis, and written reports must be your own work. Do not share custom rules, specific findings from the PCAP, evasion analysis text, or incident reports with other students.

Use of Generative AI Tools

Students are permitted to use generative AI tools (e.g., large language models) to assist with aspects of this project, including:

- understanding Suricata rule syntax or Zeek log formats,
- writing or debugging Suricata rules,
- learning about attack techniques described in the Background section,
- crafting `jq`, `awk`, or `grep` commands to query log files, and
- improving the clarity of written explanations.

However, all submitted work must reflect your own understanding and reasoning. In particular:

- You may not submit rules, analysis, or reports that you do not understand.
- You should be able to explain what each of your custom rules detects, why you wrote it, and what it would miss.
- AI-generated rules and shell commands may be syntactically plausible but still fail on this PCAP. Validate them against the actual Suricata and Zeek output before relying on them.
- The evasion analysis must reflect genuine reasoning about the adversarial limits of your specific rules, not generic statements about IDS limitations.

You are responsible for the correctness of all submitted rules, analysis, and reports regardless of whether an AI tool was used to help produce them. We may ask questions about your work to assess understanding.

Appendix: Quick Reference for Log Analysis Commands

The following commands may be useful for analyzing Zeek and Suricata output. These are suggestions, not requirements — use whatever tools you are comfortable with. (Beware cutting & pasting the ones with a single quote – the doc prep process misformats it for use in the shell.)

```
# --- Suricata EVE JSON ---

# List unique alert messages and their counts
grep '"event_type":"alert"' output/suricata/eve.json \
  | jq -r '.alert.signature' | sort | uniq -c | sort -rn

# Show all alerts for a specific source IP
grep '"event_type":"alert"' output/suricata/eve.json \
  | jq 'select(.src_ip == "10.0.0.5")'

# List unique destination IPs contacted by a host
grep '"event_type":"alert"' output/suricata/eve.json \
  | jq -r 'select(.src_ip == "10.0.0.5") | .dest_ip' \
  | sort -u

# --- Zeek logs (tab-separated) ---

# Show the column headers for any Zeek log
head -10 output/zeek/conn.log | grep '#fields'

# Extract specific fields using zeek-cut (inside container)
docker compose run --rm zeek zeek-cut id.orig_h id.resp_h \
    id.resp_p proto duration < output/zeek/conn.log

# Find DNS queries longer than 50 characters
awk -F'\t' '/^[^#]/ && length($10) > 50' output/zeek/dns.log

# Count connections per source-destination pair
awk -F'\t' '/^[^#]/{print $3, $5, $6}' output/zeek/conn.log \
  | sort | uniq -c | sort -rn | head -20

# List HTTP requests with their user agents
docker compose run --rm zeek zeek-cut host uri \
    user_agent < output/zeek/http.log

# Find files with executable MIME types
grep -i 'application/x-dosexec\|application/octet-stream' \
    output/zeek/files.log
```

Note: The exact field positions in `awk` commands depend on the Zeek log format. Always check the `#fields` header line to confirm column positions. The `zeek-cut` approach (using the Docker container) is more reliable than positional `awk` because it uses field names.