Section: **PUBP**
Project Title: **Enterprise-Grade Security on an SME Budget: Lightweight IDS with Suricata**
Student Name: **Grant Fitzgerald**
Progress Report #: **2**

Problem Statement:

Small and medium-sized enterprises (SMEs) face mounting cybersecurity threats – including ransomware, credential theft, and web application exploitation – but often lack the financial and technical resources to implement traditional enterprise-grade intrusion-detection systems (IDS). Per the 2025 Verizon Data Breach Investigations Report (DBIR), 88% of ransomware incidents targeting small businesses result in successful breaches, underscoring a critical gap in prevention capabilities. Industry literature, including the Ponemon SMB study, reinforces that SMEs struggle with vendor security oversight and timely breach response. These findings highlight an urgent need for affordable, accessible IDS solutions tailored to this demographic.

Solution Statement:

This research aims to design and prototype a lightweight, open-source intrusion detection system (IDS) using Suricata to equip SMEs with an affordable, user-friendly cybersecurity option. Key deliverables include a Suricata configuration optimized for low-resource environments, a natural language rule generator, a web-based GUI dashboard, and test scenarios that simulate real-world threat detection. Suricata was selected for its multithreaded architecture, native JSON logging, and prior exposure in my Spring Security Incident Response course – offering both technical advantages and educational continuity. This project addresses an imminent cybersecurity need while building meaningfully on a topic I found engaging in prior coursework but had limited opportunity to explore in depth.

Completed Tasks (Last 2 Weeks):

- Deployed Suricata in live mode on a bridged Linux system to monitor real-time network traffic.
- Verified alert generation and logging for ICMP pings, SSH attempts, and port scans, simulating realistic SME attack scenarios
- Developed the first version of the natural language rule translator in Python, with support for common rule types (e.g., ICMP, SSH, port scan).
- Implemented rule deduplication logic and SID management to ensure safe rule generation and persistence.
- Chose Flask as the GUI framework and designed a basic, functional web interface for rule input and system interaction.
- Defined a set of evaluation metrics for usability and detection accuracy and exported these in a formal write-up for reference.
- Integrated the rule translator with the Flask GUI, allowing users to submit natural language rules via the web interface.
- Added a log viewer to the GUI, parsing eve.json output and displaying the 20 most recent alerts with relevant metadata.
- Implemented auto-refresh for the log viewer (every 10 seconds), alert summaries (top IPs and signature counts), and theme toggle with local storage.
- Enhanced GUI styling with dark/light mode and persistent theme preferences
- Incorporated timestamp localization (UTC → local time) for log entries using Python’s pytz module.

- Created and tested a Suricata kill switch and validated rule changes with suricata -T.
- Added basic brute force rule generation support using threshold logic (e.g., count and seconds), enabling detection of repeated SSH login attempts. Flowbit-based state tracking was explored but not yet implemented, as additional tuning is needed due to inconsistent triggering during early testing.
- Prepared for stakeholder validation via LinkedIn survey: Drafted questions for a short Google Forms-based survey targeting IT and cybersecurity professionals to validate the project's real-world relevance and usability focus.

Tasks for the Next Project Report:

- Finalize advanced rule logic and flowbit support
  - Expand rule templates to include flowbit logic to cover scenarios like brute-force attacks.
  - Validate expected rule behavior through controlled test scenarios
- Expand natural language rule coverage
  - Add additional templates to support broader detection scenarios (e.g.m HTTP keyword matches)
- Integrate GUI-based rule preview and SID editing
  - Allow users to review Suricata rules prior to them being written to disk, optionally modify SIDs, and cancel/confirm actions through the GUI
- Implement logging diagnostics and error feedback
  - Add error handling to the GUI for failed rule translations, my.rules file write errors, and issues like duplicate SIDs (if user modifies during rule creation)
- Begin user testing and survey distribution
  - Begin collecting responses from LinkedIn survey and document insights for possible feature adjustments
- Develop draft sections of final paper
  - Begin outlining and writing the final report sections, focusing on background, literature review, and methodology
  - Capture lessons learned and screenshots of working components as evidence of progress
- Evaluate usability with informal feedback
  - Share the tool with a few peers/colleagues (potentially using ngrok) to observe usability, pain points, and interpretation of natural language input.
  - Gather feedback for inclusion in the evaluation section of the final report.
- Enable full-cycle alert testing and logging
  - Complete the detection loop by testing whether newly entered rules from the GUI fire correctly and are reflected in the log viewer.

Questions I have or Issues I'm running into:

- Rule Trigger Reliability: Some detection rules – particularly for brute force login attempts – do not consistently trigger during testing as they are superseded by other rules testing parts of them. I've implemented threshold logic (e.g., count: 5, seconds: 60), but further tuning may be required with flowbit logic. This is something that, in the real world, would be found during the testing phase and would not make it to production where it would be a risk to the customer. I will be looking into modifying these rules with flowbit logic that should fix issues.
- Rule Ordering and Priority: As more natural language templates are translated into Suricata rules, there is a little uncertainty about how rule ordering or flowbit dependencies may affect

performance. I'll be digging into Suricata's rule execution hierarchy and if the rule generator needs to account for different rules' dependencies.

- GUI Rule Verbiage for Non-Technical Users: As the rule translator becomes more advanced, I'm going to be looking at ways to make the alert messages more readable for non-technical users like automatically inserting port numbers or displaying whether intent is malicious or not.
- Distribution and Deployment: It would be ideal if the tool could be shareable for testing with classmates or colleagues, so I'm looking into ways to do that like creating a GitHub README to enable them to set it up locally.

Methodology Paragraph Summary:

This project will follow an iterative development methodology grounded in real-world cybersecurity use cases, particularly those relevant to small and midsize enterprises (SMEs). Suricata will be deployed in live mode within a bridged Linux environment to simulate SME network monitoring conditions. Test scenarios – including ICMP pings, SSH login attempts, and port scans – were selected based on common early-stage behaviors observed in actual breach investigations. According to the 2023 and 2025 Verizon Data Breach Investigation Reports, system intrusion and credential misuse are dominant patterns in SME incidents, with stolen credentials implicated in over 50% of breaches (DBIR 2023, p. 65; DBIR 2025, pp. 85-86). SMEs are particularly vulnerable due to limited IT resources, making low-effort tactics like brute force login attempts and basic network probing especially effective. These scenarios will be used to verify rule accuracy and system responsiveness under realistic attack conditions. A natural language rule translator will be developed in Python to simplify rule creation for non-technical users, and a lightweight Flask-based web interface will provide a user-friendly platform for rule input and alert monitoring. Evaluation will focus on usability (e.g., ease of rule creation and interpretation) and detection accuracy (i.e., true/false positive rates under simulated conditions). All components will be unit tested throughout development to ensure system stability and reliability.

To supplement the design and evaluation process, a short stakeholder survey will be distributed via LinkedIn targeting IT professionals and small business administrators. The survey – developed during week 3 in response to instructor feedback – focuses on challenges related to IDS adoption, technical barriers to rule creation, and perceived usefulness of simplified interfaces. Insights will be incorporated into the evaluation framework and discussed in the final report to strengthen the solution's justification.

Timeline:

Enter tasks for every week of this semester. One task per row and only include tasks related to your project (for example, do not include "Peer Feedback" or "Progress Report"). You'll most likely have multiple tasks per week. When you submit your first progress report, we're expecting to see a timeline of all tasks related to your project for the full length of the semester. It's expected that the tasks will develop and get more detailed over time but you must start with something. Any task you list should be actionable (for example, do not have a task "continue research" or "working on xyz"). For the Status column, let us know if you've completed the task, if it's still in-progress, maybe cancelled or whatever the status is.

Week #	**Description of Task**	**Status**
W1 (May 19-25)	Review DBIR and Ponemon literature for context	Complete
W1	Select and justify Suricata over other open-source IDS tools	Complete
W1	Finalize problem and solution statements	Complete

W1	Install and run Suricata in test mode on Kali Linux	Complete
W1	Investigate eve.json structure and Suricata logging behavior	Complete
W1	Create kill switch and validator script for Suricata process management	Complete
W1	Manually clear and verify Suricata logs and control instance spawning	Complete
W2 (May 26-June 1)	Begin drafting natural language rule mapping logic	Complete
W2	Deploy Suricata in live mode on bridged interface	Complete
W2	Develop first version of natural language rule parser (template-based)	Complete
W2	Choose GUI framework (Flask vs Node.js) and build basic layout skeleton	Complete
W2	Create test scenarios (e.g., port scan, SSH, ICMP) and verify alert logging	Complete
W2	Draft evaluation metrics for usability and detection accuracy	Complete
W3 (Jun 2-8)	Refine rule translation logic and expand supported templates	Complete
W3	Integrate rule engine with GUI rule input field	Complete
W3	Implement log viewer component in GUI	Complete
W3	Begin full-system test loop (traffic → alert → GUI)	Complete
W4 (June 9-15)	Begin writing final report (intro, methods, system design, testing setup)	In Progress
W4	Conduct additional rule testing (e.g., brute force + flowbit chaining)	In Progress
W4	Validate timestamp parsing, logging, consistency, and system stability	Complete
W4	Design survey, prepare for LinkedIn distribution	In Progress
W5 (June 16-22)	Conduct stakeholder feedback (2-3 users, non-technical if possible)	Planned
W5	Draft results and analysis section of final report (screenshots, metrics)	Planned
W5	Evaluate edge cases in rule translator (e.g., negations, ports)	Planned
W6 (June 23-29)	Polish GUI: styling, layout clarity, theme toggle UX	Planned
W6	Compare system output against expected alerts (false/true positive tracking)	Planned
W6	Start compiling appendices: rule samples, configuration files	Planned
W7 (June 30-July 6)	Draft executive summary, discussion, and lessons learned	Planned
W7	Finalize bibliography and citations (DBIR, Ponemon, SME sources)	Planned

W7	Implement any final user feedback from peer review	Planned
W8 (July 7-13)	Submit final draft for review	Planned
W8	Complete final usability and performance evaluation logs	Planned
W9 (July 14-19)	Submit final paper and code repository	Planned
W9	Record final presentation/demo	Planned
W9	Submit all project documentation and wrap-up tasks	Planned

Evaluation:

[Include any evaluation plans and/or results by Progress Report 4. This may expand as you finalize the report.]

Report Outline:

[Include an outline of your final report by Progress Report 4. This may expand as you finalize the report.]

References:

Verizon. (2025). *2025 Data Breach Investigations Report (DBIR)*. p. 43. Retrieved from https://verizon.com/dbir

Verizon. (2024). *2024 Data Breach Investigations Report (DBIR)*. Retrieved from https://verizon.com/dbir

Verizon. (2023). *2023 Data Breach Investigations Report (DBIR)*. pp. 65-68. Retrieved from https://verizon.com/dbir

Ponemon Institute. (2019). *2019 Global State of Cybersecurity in Small and Medium-Sized Businesses (SMBs)*. Sponsored by Keeper Security.

Emerging Threats. *Suricata Ruleset*. Retrieved from https://rules.emergingthreats.net

Suricata GitHub repository. https://github.com/OISF/suricata

Suricata IDS. (2024). *Official Documentation*. Retrieved from https://docs.suricata.io

Flask Documentation. Retrieved from https://flask.palletsprojects.com

pytz – World Timezone Definitions for Python, Modern and Historical. Retrieved from https://pypi.org/project/pytz

Appendix

If there are notes, sources, figures you want to reference please include that here.

A. **Key Suricata Commands and File Paths**
   - **a.** Suricata configuration file: /etc/suricata/suricata.yaml
   - **b.** Rules file path: /etc/suricata/rules/my.rules

c. Log output path: /var/log/suricata/rules/my.rule

d. Start Suricata in live mode (with interface monitoring):

```
sudo suricata -c /etc/suricata/suricata.yaml -i eth1 -v
```

e. Validate rules syntax:

```
sudo suricata -T -c /etc/suricata/suricata.yaml
```

f. Kill switch script snippet:

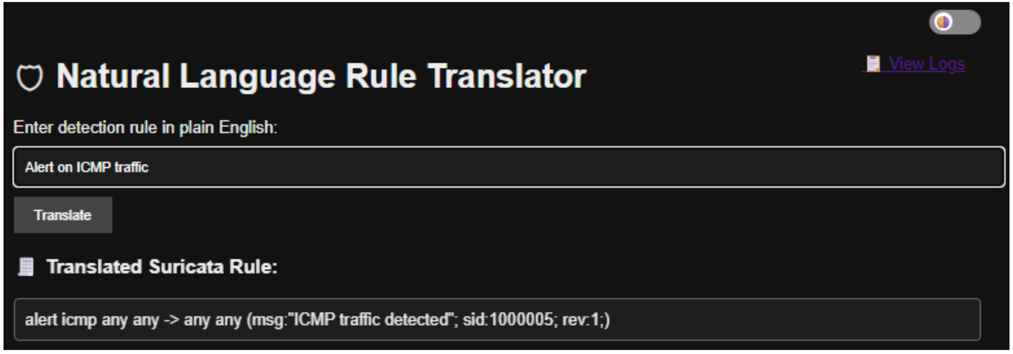
```
sudo pkill suricata
```

g. # or kill by PID:

```
sudo kill $(pidof suricata)
```

B. GUI Screenshots and Features

a. Main interface with natural language rule input



🛡 Natural Language Rule Translator

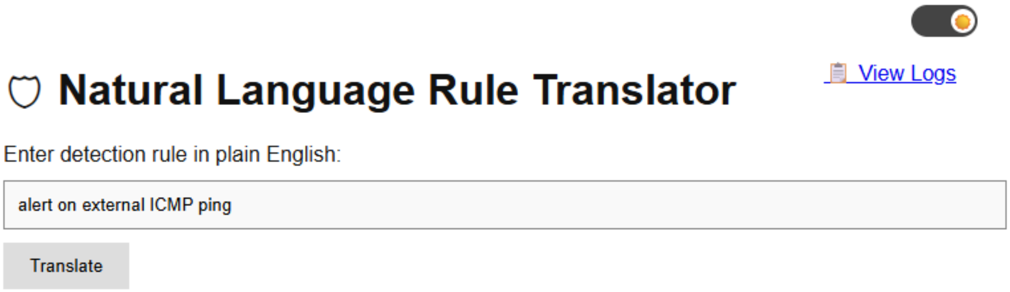
Enter detection rule in plain English:

Alert on ICMP traffic

Translated Suricata Rule:

```
alert icmp any any -> any any (msg:"ICMP traffic detected"; sid:1000005; rev:1;)
```

b. Light/Dark mode toggle with persistent preference via localStorage



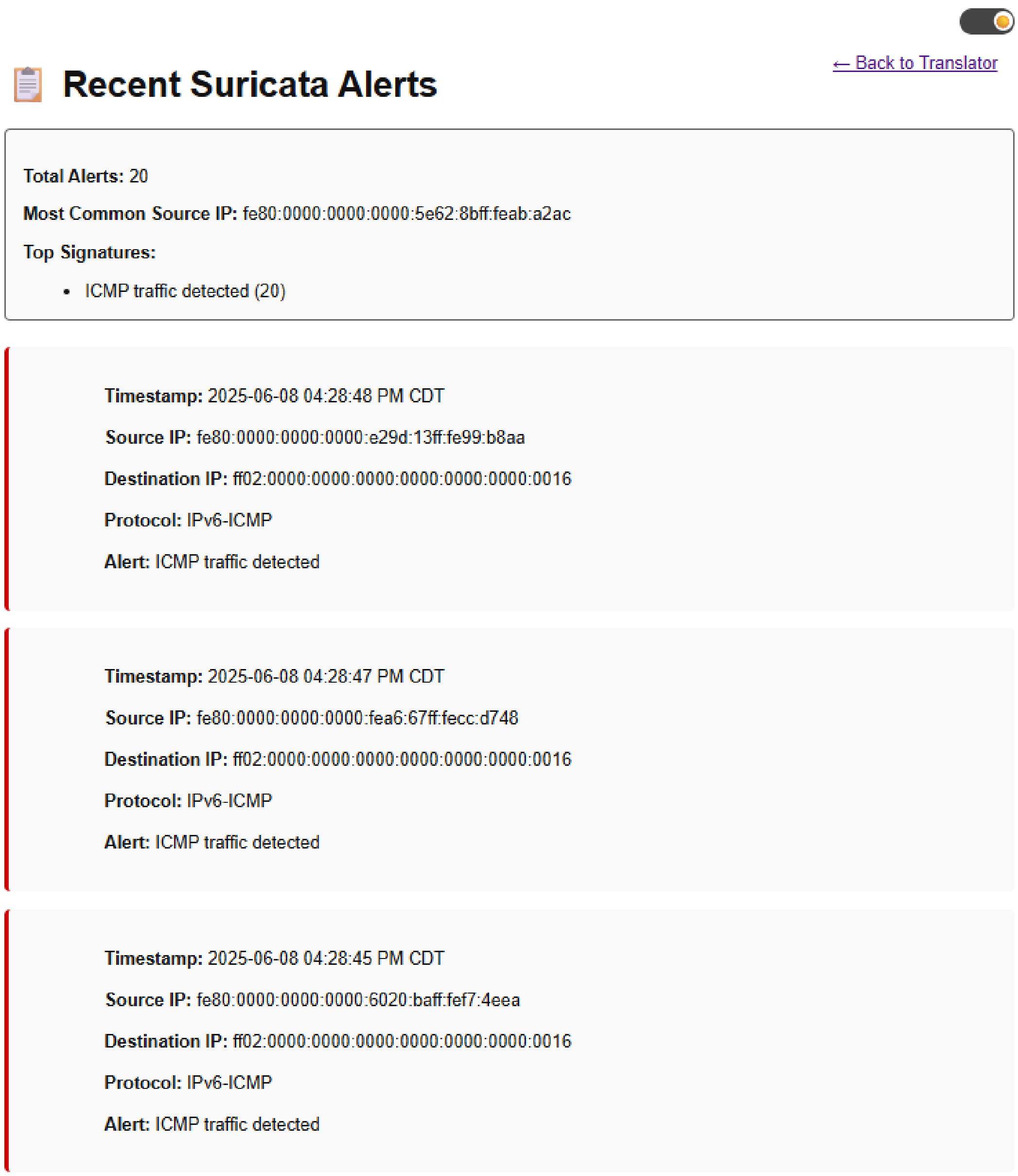
🛡 Natural Language Rule Translator

Enter detection rule in plain English:

alert on external ICMP ping

📋 Recent Suricata Alerts

c. Log viewer with auto-refresh (10 seconds) displaying latest 20 alerts



← Back to Translator

📋 **Recent Suricata Alerts**

Total Alerts: 20

Most Common Source IP: fe80:0000:0000:0000:5e62:8bff:feab:a2ac

Top Signatures:

- ICMP traffic detected (20)

Timestamp: 2025-06-08 04:28:48 PM CDT

Source IP: fe80:0000:0000:0000:e29d:13ff:fe99:b8aa

Destination IP: ff02:0000:0000:0000:0000:0000:0000:0016

Protocol: IPv6-ICMP

Alert: ICMP traffic detected

Timestamp: 2025-06-08 04:28:47 PM CDT

Source IP: fe80:0000:0000:0000:fea6:67ff:fecc:d748

Destination IP: ff02:0000:0000:0000:0000:0000:0000:0016

Protocol: IPv6-ICMP

Alert: ICMP traffic detected

Timestamp: 2025-06-08 04:28:45 PM CDT

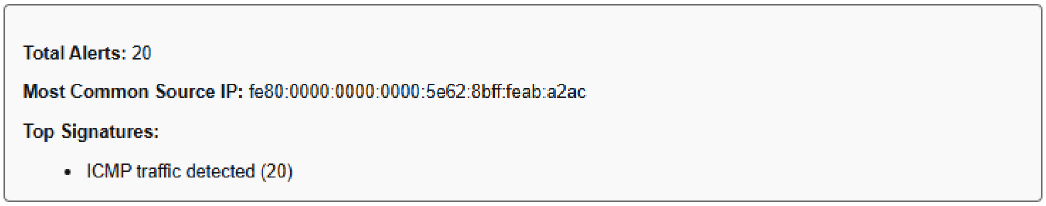
Source IP: fe80:0000:0000:0000:6020:baff:fef7:4eea

Destination IP: ff02:0000:0000:0000:0000:0000:0000:0016

Protocol: IPv6-ICMP

Alert: ICMP traffic detected

d. Summary stats including total alerts, top signatures, and top source IP



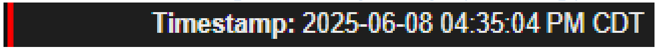
Total Alerts: 20

Most Common Source IP: fe80:0000:0000:0000:5e62:8bff:feab:a2ac

Top Signatures:

- ICMP traffic detected (20)

e. Localized timestamp rendering using Python’s pytz module (located in Green Bay, WI)



Timestamp: 2025-06-08 04:35:04 PM CDT

C. Survey Instrument for Stakeholder Feedback

Google Forms Survey for LinkedIn Distribution

D. Sample Translations and Rule Logic

Generated Suricata Rule:

```
alert icmp any any -> any any (msg:"ICMP ping detected";
sid:1000001; rev:1;)
```

Resulting eve.json Log Entry:

```
{
  "timestamp": "2025-05-23T23:32:28.510559-0400",
  "event_type": "alert",
  "src_ip": "108.157.142.105",
  "dest_ip": "192.168.6.134",
  "proto": "ICMP",
  "alert": {
    "signature_id": 1000001,
    "signature": "ICMP ping detected",
    "severity": 3
  }
}
```

This confirms the rule was correctly loaded, triggered, and logged during live packet inspection.

Additional examples of translations and rule logic to follow in further progress reports

[The inclusion of draft work product should begin by Progress Report 3.]