Tampereen yliopisto

Joni Lehtonen 50401468

SECURE PROGRAMMING PROJECT – NETWORK INTRUSION DETECTION SYSTEM

General description

The idea at the start was to create a Network Intrusion Detection System, essentially a lightweight Windows program that detects malicious packets/payloads sent to the network. With it, students can observe and learn live network traffic in a safe offline environment, while learning about secure programming principles.

There was an idea to make a simple Intrusion Prevention System (IPS), where the program would automatically block IP’s depending on the payloads they send. For security, testing and functional reasons the program is running a feedback loop on 127.0.0.1, so it is not actually logging any outside IPs, so creating an IPS would require some modifications, which would make it more vulnerable and complicate testing.

It is completely done with Python with libraries, with a simple GUI for login and network traffic detection. It only runs on Windows, as scapy, the library used for network traffic sniffing, is a Windows only library. Uses editable rules.txt for attack signatures. The rules.txt right now contain OWASP Top 10 attack signatures, meaning it will detect those attacks. More can be added.

Workflow

1. Start main.py → a **login screen** appears. Admin//Admin123 for login. No new users can be created.
2. After successful authentication the **dashboard window** opens.
3. All packets travelling over the **loopback interface (127.0.0.1)** are analysed in real time.
4. Any traffic matching a detection rule or anomaly triggers an alert that is shown immediately in the GUI and written to malicious.log.
5. For to see malicious packet logging, packet_test.py should be run. It will send test packets inside your network to yourself.

Structure

Network Intrusion Detection System

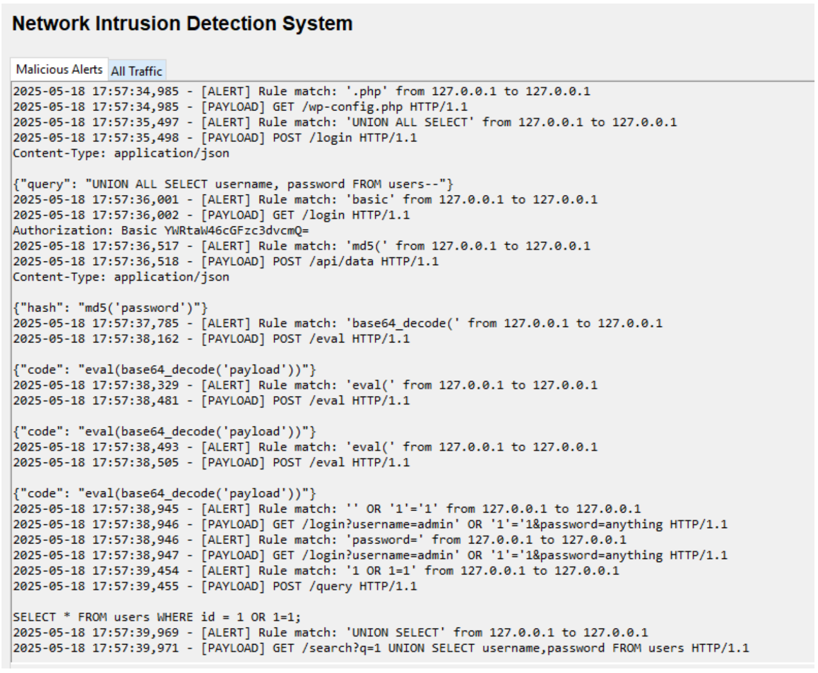
```
├── main.py        # startup + thread orchestration
├── login.py       # bcrypt authentication, users.json storage
├── capture.py     # Scapy sniffer bound to 127.0.0.1
├── detection.py   # rate-limit logic + rule matching
├── packet_logging.py# dual logging (packet.log & malicious.log created) + SHA-256
hashing
├── gui.py         # Tkinter dashboard with two tabs
├── rules.txt      # editable signature list
└── users.json     # bcrypt-hashed credentials – Created after first boot
```

OWASP Top 10

OWASP item	Where addressed	Brief explanation
A01 Broken Authentication	login.py	Passwords hashed with **bcrypt**; max 5 login attempts / minute.
A02 Cryptographic Failures	packet_logging.py	Every write to malicious.log and packet.log is followed by a SHA-256 digest calculation; displayed in GUI for integrity checks.
A03 Injection	Detection rules	Signatures for SQLi/XSS/command-injection are matched in payloads even though the tool itself does not run SQL.
A04 Insecure Design	Modular layout	Packet capture, detection, logging, GUI, and auth are isolated; each can be unit-tested separately.
A08 Software & Data Integrity Failures	Log hashing + pip requirement pinning	Hashes detect tampering; requirements.txt freezes package versions.
A10 Logging & Monitoring	Dual-logger approach	All packets - packet.log; alerts malicious.log; live hash + GUI alert counter.

Testing

The testing was done manually and with some python scripts. One testing file packet_test.py is included in the codebase. This file tests the malicious packet detection and logging feature, which is the main function of the software. It sends packets to yourself, which the program should catch according to the signatures set in rules.txt. Here is a screenshot of the malicious packets caught by the software and we can see it is working properly. Using packet_test.py on Windows might require the user to run it on administrator and/or create rules for the firewall.



```
2025-05-18 17:57:34,985 - [ALERT] Rule match: '.php' from 127.0.0.1 to 127.0.0.1
2025-05-18 17:57:34,985 - [PAYLOAD] GET /wp-config.php HTTP/1.1
2025-05-18 17:57:35,497 - [ALERT] Rule match: 'UNION ALL SELECT' from 127.0.0.1 to 127.0.0.1
2025-05-18 17:57:35,498 - [PAYLOAD] POST /login HTTP/1.1
Content-Type: application/json

{"query": "UNION ALL SELECT username, password FROM users--"}
2025-05-18 17:57:36,001 - [ALERT] Rule match: 'basic' from 127.0.0.1 to 127.0.0.1
2025-05-18 17:57:36,002 - [PAYLOAD] GET /login HTTP/1.1
Authorization: Basic YWRtaW46cGFzc3dvcmQ=
2025-05-18 17:57:36,517 - [ALERT] Rule match: 'md5(' from 127.0.0.1 to 127.0.0.1
2025-05-18 17:57:36,518 - [PAYLOAD] POST /api/data HTTP/1.1
Content-Type: application/json

{"hash": "md5('password')"}
2025-05-18 17:57:37,785 - [ALERT] Rule match: 'base64_decode(' from 127.0.0.1 to 127.0.0.1
2025-05-18 17:57:38,162 - [PAYLOAD] POST /eval HTTP/1.1

{"code": "eval(base64_decode('payload'))"}
2025-05-18 17:57:38,329 - [ALERT] Rule match: 'eval(' from 127.0.0.1 to 127.0.0.1
2025-05-18 17:57:38,481 - [PAYLOAD] POST /eval HTTP/1.1

{"code": "eval(base64_decode('payload'))"}
2025-05-18 17:57:38,493 - [ALERT] Rule match: 'eval(' from 127.0.0.1 to 127.0.0.1
2025-05-18 17:57:38,505 - [PAYLOAD] POST /eval HTTP/1.1

{"code": "eval(base64_decode('payload'))"}
2025-05-18 17:57:38,945 - [ALERT] Rule match: '' OR '1'='1' from 127.0.0.1 to 127.0.0.1
2025-05-18 17:57:38,946 - [PAYLOAD] GET /login?username=admin' OR '1'='1&password=anything HTTP/1.1
2025-05-18 17:57:38,946 - [ALERT] Rule match: 'password=' from 127.0.0.1 to 127.0.0.1
2025-05-18 17:57:38,947 - [PAYLOAD] GET /login?username=admin' OR '1'='1&password=anything HTTP/1.1
2025-05-18 17:57:39,454 - [ALERT] Rule match: '1 OR 1=1' from 127.0.0.1 to 127.0.0.1
2025-05-18 17:57:39,455 - [PAYLOAD] POST /query HTTP/1.1

SELECT * FROM users WHERE id = 1 OR 1=1;
2025-05-18 17:57:39,969 - [ALERT] Rule match: 'UNION SELECT' from 127.0.0.1 to 127.0.0.1
2025-05-18 17:57:39,971 - [PAYLOAD] GET /search?q=1 UNION SELECT username,password FROM users HTTP/1.1
```

Issues & Unimplemented features

There are few features that did not make the final version. Log files were supposed to be encrypted, but there were some problems with real-time encryption and it was deemed unnecessary. Hashing of the logs is enough to prevent tampering. This does create a small security issue.

The password is hardcoded right now, software was supposed to ask for the user to change the password on first login, but this was dropped. Creates a security issue.

Rules.txt is hardcoded, no way to edit except manually. Software could have a tab/option for it. Intrusion Prevention System (IPS) was dropped, as the software is only running locally. Running locally is also a limitation. There was an idea to create this as a web application so more vulnerabilities could be created/patched, but this was dropped.

AI Usage

AI was used extensively while creating this software. Especially while troubleshooting malicious packet detection. The first version captured all the packets coming outside the network, working as a simple packet sniffer, but sending malicious packets to myself did not work at all, because I have limited knowledge about network issues. I did understand where the problem was, and guided AI to find the solution. AI helped to turn the software into feedback loop to 127.0.0.1, which did remove some planned features. Otherwise, things like rules.txt was created by AI completely.

Conclusion

In the end the program came out fine, learned quite a bit about the security vulnerabilities this kind of thing could have. Some planned features did not make the final version, but otherwise I am content. It has the start of a larger program. Looking back, choosing another type of software with more technologies used would have been better, as I could have used pipelines and automated tests.