Networks and Systems Security
Week 04
Malicious Software

Aims of the Seminar

Welcome to the Malicious Software workshop. Malicious software—commonly known as malware—remains one of the most pervasive and evolving threats to computer systems. From self-replicating worms to stealthy rootkits and deceptive trojans, malware exploits weaknesses in software, networks, and human behaviour alike. Understanding how these threats operate is essential not only for defending against them, but also for anticipating how they might adapt in the future.

This workshop begins with a conceptual overview of the main categories of malware, their propagation mechanisms, and payload behaviours. You will then move into a series of hands-on Python exercises designed to illustrate how malware can be detected, analysed, and contained using simple yet powerful tools.

Rather than handling real malicious code, the activities simulate key behaviours—file integrity checking, signature scanning, and worm propagation—so participants can safely explore how malware interacts with systems. By the end of the workshop, you will have a clearer view of how malicious software functions, how it can be identified, and how thoughtful countermeasures—both technical and human—can reduce its impact.

Workshop Outline:

1. Explain the main categories of malware (virus, worm, trojan, spyware, rootkit, etc.).
2. Understand how malware propagates and hides itself.
3. Demonstrate basic malware detection and analysis principles using Python.
4. Apply file integrity checking and simple anomaly detection as defensive countermeasures.

Feel free to discuss your work with peers, or with any member of the teaching staff.

Reminder

We encourage you to discuss the content of the workshop with the delivery team and any findings you gather from the session.

Workshops are not isolated, if you have questions from previous weeks, or lecture content, please come and talk to us.

Exercises herein represent an example of what to do; feel free to expand upon this.

Helpful Resources

hashlib Library

https://www.geeksforgeeks.org/python/hashlib-module-in-python/

Section 1: File Integrity Checker

Objective

To understand how **host-based antivirus systems** detect unauthorized file modifications by using file hashing as a baseline for system integrity.

Key Concepts

- **Integrity Checking:** A preventive technique that detects unauthorized changes to files or system components.
- **Hashing (SHA-256):** Generates a unique fingerprint for each file; even a one-bit change produces a completely different hash.
- **Baseline Comparison:** Security systems record "clean" file hashes and later compare them to detect tampering or infection.
- **Malware Connection:** Many viruses modify executable files — this method detects such unauthorized changes.

💻 **Exercise: Initially create a folder and save several files into it. Then write the script to loop through the files and generate a SHA-256 hash for each. The results should be saved in a csv file with the file name and the hash and time stamp.**

🗣️ Discussion Points

- Why do security systems rely on *file hashes* instead of timestamps or file sizes?
- How might malware attempt to evade integrity checking?
- What happens if a legitimate system update changes many hashes — how should you manage that?

Section 2: Detecting Suspicious File Changes

Objective
To simulate how malware detection software identifies file tampering by comparing current file hashes to a trusted baseline.

Key Concepts

- **Change Detection:** Compares new hashes against the baseline to flag altered or deleted files.
- **Heuristic Insight:** Sudden modification of system or executable files often indicates infection.
- **Forensics Use:** Detecting *when* and *which* files changed helps trace intrusion paths.

Write Python Code to Compare and Detect Changes

Discussion Points

- How could this approach complement antivirus scanning?
- What are the limitations if a rootkit hides file access at the kernel level?
- How could you extend this system to automatically restore modified files from backup?

✍️🔐 Section 3: Signature-Based Malware Detection

Objective

To understand how early antivirus programs detected malware using pattern matching, and why modern malware easily bypasses this method.

Key Concepts

- **Signatures:** Unique byte or code patterns associated with known malware.
- **Pattern Matching:** Scans files for predefined suspicious expressions or function calls.
- **Weakness:** Fails against obfuscated, encrypted, or polymorphic malware.
- **Modern Shift:** Behavioural analysis and machine learning now complement signature detection.

💻 Write Python Code to Scan for Suspicious Patterns such as

```
SIGNATURES = [r"eval\(", r"base64\.b64decode", r"socket\.connect", r"exec\(", r"import os"]
```

🗣️ Discussion Points

- Why is signature-based detection ineffective against polymorphic or metamorphic viruses?
- Could attackers intentionally include harmless “signature-looking” strings to trigger false alarms?
- How do heuristic and behavioural scanners improve on this approach?

🪱🪱 Section 4: Worm Propagation Simulation

Objective
To visualize how a worm spreads rapidly across a network using random and opportunistic scanning strategies.

Key Concepts

- **Worms:** Self-contained programs that propagate through networks without user action.
- **Scanning:** Worms randomly or strategically locate vulnerable systems.
- **Propagation Dynamics:** The infection rate depends on network topology and available hosts.
- **Containment:** Rate limiting, ingress filtering, and anomaly detection reduce spread.

💻 Write Python Code to Simulate Network Propagation

🗣️ Discussion Points

- How does the infection curve change if you double the number of attempts per host?
- Why might a worm choose a *local subnet* propagation strategy?
- Which containment strategy (rate halting, scan detection, thresholding) would be most effective here?

⚖️ Section 5: Countermeasure Design Challenge

Objective
To consolidate learning by designing a layered defence plan combining prevention, detection, and mitigation techniques.

Key Concepts

- **Defence in Depth:** Layering multiple, complementary protection methods.
- **Host-based vs. Network-based:** Combining internal integrity monitoring with perimeter defences.
- **User Awareness:** Social engineering remains one of the weakest links.
- **Resilience:** Recovery and containment strategies matter as much as prevention.

💻 Task

Students form small groups and design a Python-based monitoring prototype that:

- Detects unusual network activity (e.g., excessive outbound connections).
- Logs or alerts administrators when anomalies are found.
- Uses existing code from previous exercises as building blocks.

🗣️ Discussion Points

- Which layer of defence failed first in most real-world malware outbreaks?
- How can AI-based systems be trained to detect abnormal patterns safely?
- How should security teams balance detection sensitivity and false positives?

☑ Key Takeaways & Best Practices

This workshop explored how malicious software operates, spreads, and can be detected or mitigated using simple yet effective defensive strategies. Always remember:

- Maintain System Integrity: Establish cryptographic file baselines and regularly verify them to detect unauthorized modifications.
- Keep Software Patched: Most malware exploits known vulnerabilities. Regular updates drastically reduce exposure.
- Use Behavioural Detection, Not Just Signatures: Relying solely on static signatures leaves you blind to polymorphic and metamorphic malware. Combine heuristic and behavioural methods.
- Restrict Privileges: Malware often relies on elevated access. Enforce least-privilege policies to limit the scope of compromise.
- Monitor Network Anomalies: Unusual outbound connections or scanning behaviour often indicate worm or botnet activity.
- Educate Users: Many infections still begin with social engineering. Awareness is one of the cheapest and most effective countermeasures.
- Layer Your Defences: Combine host-based integrity checking, network-level monitoring, and user awareness to build resilience against modern threats.
- Plan for Recovery: Detection without response is half a defence. Maintain clean backups and an incident response plan for containment and restoration.

The end 😊