

Malware Detection & Classification

From static signatures to AI-powered analysis — building the defender's toolkit for identifying, classifying, and writing detection rules for malicious software.

⚠ Educational Use Only

Malware analysis techniques are for defensive security research and CTF competitions only. Handle malware samples only in isolated environments. Never create or distribute malware for malicious purposes.

Blue Team

Malware Analysis

YARA

AI-Assisted Detection

Threat Intel

Why Malware Analysis Matters

- **560,000+ new malware samples** detected daily by AV-TEST Institute
- **Incident response** — understanding what happened and how far the compromise spread
- **Detection engineering** — writing signatures and rules to catch variants
- **Threat intelligence** — attributing campaigns and tracking adversary TTPs
- **CTF competitions** — reverse engineering challenges require rapid triage and classification

The Defender's Dilemma

Attackers need to succeed once. Defenders need to detect every variant. AI helps us scale detection to match the volume.

Malware Taxonomy

Type	Propagation	Primary Objective
Virus	Infects files; requires host execution	Code injection, corruption
Worm	Self-replicating over networks	Lateral movement, spreading
Trojan	Disguised as legitimate software	Backdoor access, data theft
Ransomware	Phishing, exploits, RDP brute force	Encrypt files, extort payment
Rootkit	Exploit or bundled install	Persist at kernel/boot level
RAT	Phishing, trojanized apps	Remote control, surveillance
Cryptominer	Exploit, supply chain, trojan	Steal CPU/GPU for mining

Malware Taxonomy — Deeper Look

Ransomware Families

- **LockBit 3.0** — RaaS model, double extortion
- **BlackCat/ALPHV** — Rust-based, cross-platform
- **Cl0p** — MOVEit zero-day exploitation
- **Royal/BlackSuit** — targets enterprise, partial encryption

RAT Examples

- **Cobalt Strike Beacon** — widely used in red/blue team
- **AsyncRAT** — open-source, heavily modified
- **Remcos** — commercial "remote admin" tool
- **ShadowPad** — APT-linked modular backdoor

Blended Threats

Modern malware rarely fits one category. A typical attack chain: **phishing dropper (trojan)** → **Cobalt Strike beacon (RAT)** → **lateral movement (worm-like)** → **ransomware deployment**.

Analysis Approaches Overview

Static Analysis

- Examine without executing
- File hashes (MD5, SHA256)
- Strings extraction
- PE/ELF header parsing
- Import/export table review
- Disassembly & decompilation

Dynamic Analysis

- Execute in sandboxed environment
- API call monitoring
- Network traffic capture
- File system changes
- Registry modifications
- Process behavior tracking

Notes: Always start with static analysis. Only move to dynamic analysis in an isolated VM or sandbox. Never run unknown samples on production systems.

Static Analysis — File Hashes

The first step: compute cryptographic hashes to identify known malware via threat intelligence databases.

```
# Compute hashes of a suspicious file
$ md5sum suspicious.exe
a1b2c3d4e5f6789012345678abcdef01  suspicious.exe

$ sha256sum suspicious.exe
e3b0c44298fc1c149afbf4c8996fb92427ae41e4649b934ca495991b7852b855  suspicious.exe

# Query VirusTotal via API
$ curl -s "https://www.virustotal.com/api/v3/files/e3b0c442..." \
  -H "x-apikey: $VT_API_KEY" | jq '.data.attributes.last_analysis_stats'
{
  "malicious": 48,
  "suspicious": 3,
  "undetected": 12,
  "harmless": 0
}
```

MD5 — fast lookup, collision-vulnerable. **SHA-256** — standard for IoCs. **ssdeep/TLSH** — fuzzy hashes to find similar variants.

Static Analysis — Strings Extraction

Extracting strings reveals embedded URLs, file paths, registry keys, encryption keys, and error messages left by malware authors.

```
# Extract strings from a suspicious binary
$ strings -n 8 suspicious.exe | head -40
!This program cannot be run in DOS mode.
C:\Users\admin\AppData\Local\Temp\payload.dll
http://185.243.115[.]42/gate.php
Mozilla/5.0 (Windows NT 10.0; Win64; x64)
HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\Run
cmd.exe /c schtasks /create /tn "WindowsUpdate"
RegSetValueExA
VirtualAllocEx
WriteProcessMemory
CreateRemoteThread
NtUnmapViewOfSection
powershell -enc JABjAGwAaQBlAG4AdAAgAD0A...
-----BEGIN RSA PUBLIC KEY-----
```

Red Flags in Strings Output

Watch for: hardcoded IPs/domains, base64-encoded PowerShell, process injection APIs, AV killer commands, cryptocurrency wallet addresses, and C2 URI patterns.

Static Analysis — PE Header Inspection

The Portable Executable (PE) header reveals compilation details, section layout, and imported functionality.

```
# Using pefile (Python) to inspect PE headers
$ python3 -c "
import pefile
pe = pefile.PE('suspicious.exe')
print(f'Compile Time: {pe.FILE_HEADER.TimeDateStamp}')
print(f'Sections: {len(pe.sections)}')
for s in pe.sections:
    name = s.Name.decode().rstrip('\x00')
    entropy = s.get_entropy()
    print(f' {name:8s} size={s.SizeOfRawData:8d} entropy={entropy:.2f}')
"
Compile Time: 1709251200 (2024-03-01 00:00:00)
Sections: 5
.text      size= 204800 entropy=6.42
.rdata     size=  51200 entropy=5.18
```

Entropy as a Packing Indicator

Entropy > 7.0 in a section strongly suggests packing or encryption. The `.packed` section above at 7.98 is almost certainly compressed/encrypted payload data. Normal code sections typically range 5.0–6.5.

Static Analysis — Import Table Analysis

Imported Windows API functions reveal the malware's intended capabilities.

API Function	Suspicious Capability
<code>VirtualAllocEx</code>	Allocate memory in another process
<code>WriteProcessMemory</code>	Write to another process's memory
<code>CreateRemoteThread</code>	Execute code in another process (injection)
<code>NtUnmapViewOfSection</code>	Process hollowing technique
<code>CryptEncrypt</code>	File encryption (ransomware indicator)
<code>InternetOpenUrlA</code>	Download from remote URL
<code>RegSetValueExA</code>	Modify registry (persistence)
<code>AdjustTokenPrivileges</code>	Privilege escalation

Notes: Many legitimate programs use these APIs. The combination matters: VirtualAllocEx + WriteProcessMemory + CreateRemoteThread together is a classic injection pattern.

Dynamic Analysis — Sandboxing

Execute the sample in an isolated environment and observe its behavior without risk to production systems.

Popular Sandboxes

- **Cuckoo/CAPEv2** — open-source, on-prem
- **ANY.RUN** — interactive cloud sandbox
- **Joe Sandbox** — deep behavior analysis
- **Hybrid Analysis** — free community tier
- **Windows Sandbox** — quick disposable env

What Sandboxes Capture

- Process creation trees
- File system modifications
- Registry changes
- Network connections & DNS
- API call sequences
- Screenshots of execution

Sandbox Evasion

Advanced malware detects sandboxes via: VM artifacts, timing checks, mouse movement detection, environment checks (username, hostname, MAC address). Defenders must harden sandbox environments.

Dynamic Analysis — Behavioral Indicators

Sandbox reports reveal malicious behaviors. Here is a realistic behavioral summary:

```
sample: suspicious.exe
sha256: e3b0c44298fc1c149afbf4c8996fb924...
verdict: MALICIOUS (score: 92/100)

behaviors:
  persistence:
    - Registry Run key: HKCU\Software\Microsoft\Windows\CurrentVersion\Run\svchost
    - Scheduled task: "WindowsUpdate" every 5 minutes
    - Dropped to: C:\Users\admin\AppData\Roaming\svchost.exe

  process_injection:
    - Technique: Process Hollowing (T1055.012)
    - Target: explorer.exe (PID 4892)
    - APIs: NtUnmapViewOfSection → VirtualAllocEx → WriteProcessMemory
```

Dynamic Analysis — API Call Monitoring

API traces reveal the exact sequence of system interactions. Here is a partial trace from a process injection attack:

```
PID: 3412 (suspicious.exe)
[00:00.012] CreateFileA("C:\Windows\System32\svchost.exe", GENERIC_READ)
[00:00.015] ReadFile(handle=0x1A4, buffer=0x00400000, size=245760)
[00:00.031] CreateProcessA("svchost.exe", CREATE_SUSPENDED) → PID 5288
[00:00.045] NtUnmapViewOfSection(PID 5288, 0x00400000)
[00:00.048] VirtualAllocEx(PID 5288, 0x00400000, 0x50000, MEM_COMMIT)
[00:00.052] WriteProcessMemory(PID 5288, 0x00400000, payload, 327680)
[00:00.061] SetThreadContext(PID 5288, EntryPoint=0x00401230)
[00:00.063] ResumeThread(PID 5288)
--- Process 5288 (hollowed svchost.exe) now executing malicious code ---
[00:00.120] InternetOpenA("Mozilla/5.0 ...")
[00:00.250] InternetConnectA("185.243.115[.]42", 80)
[00:00.310] HttpSendRequestA("POST", "/gate.php", encrypted_blob)
```

Notes: This trace shows classic process hollowing: create suspended process, unmap legitimate code, write malicious payload, resume execution. The hollowed process then initiates C2 communication.

Registry Modifications — Key Indicators

Malware frequently modifies the Windows registry for persistence, defense evasion, and configuration storage.

```
# Persistence
[SET] HKCU\Software\Microsoft\Windows\CurrentVersion\Run
      "WindowsDefenderUpdate" = "C:\Users\admin\AppData\Roaming\svchost.exe"

# Disable Security
[SET] HKLM\SOFTWARE\Policies\Microsoft\Windows Defender
      "DisableAntiSpyware" = DWORD:1
[SET] HKLM\SOFTWARE\Policies\Microsoft\Windows Defender\Real-Time Protection
      "DisableRealtimeMonitoring" = DWORD:1

# Disable UAC
[SET] HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\Policies\System
      "EnableLUA" = DWORD:0

# Store encrypted C2 config
```

Detection Opportunity

Monitor registry changes with **Sysmon Event ID 13** (Registry value set). Alert on modifications to Run keys, security policy keys, and unusual CLSID entries.

Using AI for Malware Analysis

Large language models excel at pattern recognition across malware artifacts. They can accelerate triage that would take analysts hours.

What to Feed the LLM

- Strings output from binaries
- API call traces from sandboxes
- Decompiled pseudocode (Ghidra/IDA)
- Network packet captures
- PE header metadata
- YARA match results

What the LLM Can Do

- Classify malware family
- Explain obfuscated code logic
- Identify C2 protocol patterns
- Map behaviors to MITRE ATT&CK
- Generate detection signatures
- Summarize for incident reports

AI Analysis — Strings Triage Prompt

Feed extracted strings to an LLM for rapid classification and IoC identification.

PROMPT:

You are a malware analyst. Analyze these strings extracted from a suspicious PE binary and provide:

1. Likely malware classification (type and family if possible)
2. Key indicators of compromise (IPs, domains, file paths)
3. MITRE ATT&CK techniques observed
4. Confidence level and reasoning

STRINGS OUTPUT:

```
http://185.243.115[.]42/gate.php
Mozilla/5.0 (Windows NT 10.0; Win64; x64)
cmd.exe /c schtasks /create /tn "WindowsUpdate" /sc minute /mo 5
RegSetValueExA
VirtualAllocEx / WriteProcessMemory / CreateRemoteThread
powershell -enc JABiAGwAaOBLAG4AdAAaAD0A...
```

Expected AI Response

Classification: Ransomware (likely custom variant). **Key IoCs:** C2 at 185.243.115[.]42, bitcoin wallet bc1qxy2k..., file extension .locked. **ATT&CK:** T1055 (Process Injection), T1053.005 (Scheduled Task), T1486 (Data Encrypted for Impact), T1490 (Inhibit System Recovery).

AI Analysis — Decompiled Code Review

LLMs can explain decompiled pseudocode from Ghidra or IDA Pro, identifying malicious functionality.

```
// Ghidra decompiled output — function at 0x00401230
void FUN_00401230(void) {
    char *key = "S3cr3tK3y!@#";
    HANDLE hSnap = CreateToolhelp32Snapshot(TH32CS_SNAPPROCESS, 0);
    PROCESSENTRY32 pe;
    pe.dwSize = sizeof(PROCESSENTRY32);

    if (Process32First(hSnap, &pe)) {
        do {
            if (strcmp(pe.szExeFile, "MsMpEng.exe") == 0 ||
                strcmp(pe.szExeFile, "avgnt.exe") == 0 ||
                strcmp(pe.szExeFile, "mbam.exe") == 0) {
                HANDLE hProc = OpenProcess(PROCESS_TERMINATE, 0, pe.th32ProcessID);
                TerminateProcess(hProc, 0);
            }
        } while (Process32Next(hSnap, &pe));
    }
}
```

AI identifies: AV process termination (T1562.001) followed by full-drive encryption (T1486). The hardcoded XOR key `S3cr3tK3y!@#` may enable decryption.

YARA Rules — Introduction

YARA is the pattern-matching language for malware detection. Rules combine string patterns, byte sequences, and conditions to identify malware families.

```
rule ExampleRule {
  meta:
    author = "Analyst"
    description = "Detects Example Malware"
    date = "2024-03-01"

  strings:
    $s1 = "malicious string" ascii
    $s2 = { 4D 5A 90 00 03 00 00 00 } // MZ header bytes
    $s3 = /https?:\/\/[a-z0-9\-\.\.]+\gate\.php/ nocase

  condition:
    uint16(0) == 0x5A4D and // File starts with MZ
    filesize < 5MB and
    2 of ($s1, $s2, $s3)
```

YARA Rule Components

meta — descriptive metadata. **strings** — text, hex, or regex patterns. **condition** — boolean logic combining string matches, file properties, and offsets.

YARA Rule — Ransomware Detection

A realistic YARA rule targeting the ransomware sample we analyzed earlier.

```
rule Ransomware_CustomLocker_2024 {  
  meta:  
    author = "BlueTeam CTF"  
    description = "Detects CustomLocker ransomware variant"  
    date = "2024-03-15"  
    severity = "critical"  
    mitre_attack = "T1486, T1490, T1055"  
  
  strings:  
    // C2 communication  
    $c2_url = "gate.php" ascii wide  
    $c2_ip = "185.243.115" ascii  
  
    // Ransom note indicators  
    $ransom1 = "YOUR FILES HAVE BEEN ENCRYPTED" ascii wide nocase
```

YARA Rule — Process Injection Detection

Detecting common process injection techniques with byte-level patterns.

```
rule ProcessInjection_Hollowing {
  meta:
    author = "BlueTeam CTF"
    description = "Detects process hollowing technique"
    mitre_attack = "T1055.012"

  strings:
    // API call sequence for process hollowing
    $api1 = "NtUnmapViewOfSection" ascii
    $api2 = "VirtualAllocEx" ascii
    $api3 = "WriteProcessMemory" ascii
    $api4 = "SetThreadContext" ascii
    $api5 = "ResumeThread" ascii
    $api6 = "CreateProcessA" ascii
```

Notes: This rule may generate false positives with legitimate software that uses these APIs. Tune the condition based on your environment and combine with behavioral detection.

AI-Assisted YARA Rule Writing

Use LLMs to generate, improve, and validate YARA rules from analysis findings.

PROMPT:

I found these indicators from a malware sample. Write a YARA rule:

- SHA256: alb2c3d4...
- Strings: "cmd.exe /c net user", "mimikatz", "sekurlsa::logonpasswords"
- Drops DLL to %TEMP%\msupdate.dll
- Connects to evil-domain[.]xyz on port 8443
- Uses XOR key 0x3F for string obfuscation
- Compiled with MinGW, UPX packed
- Contains PDB path: C:\Users\dev\source\stealer\Release\stealer.pdb

```
// AI-generated YARA rule
rule Stealer_Mimikatz_Dropper {
    meta:
        author = "AI-Assisted Analysis"
        description = "Credential stealer with embedded Mimikatz"
        hash = "alb2c3d4..."

    strings:
        $pdb = "\\stealer\\Release\\stealer.pdb" ascii
        $mimil = "sekurlsa::logonpasswords" ascii wide
```

```
$mimi2 = "mimikatz" ascii wide nocase  
$drop = "msupdate.dll" ascii wide  
$c2 = "evil-domain" ascii  
$cmd = "net user" ascii
```

IoC Extraction & Enrichment with AI

Indicators of Compromise (IoCs) are the atomic evidence of a breach. AI excels at extracting and contextualizing them from unstructured data.

IoC Type	Example	Where Found
IP Address	185.243.115[.]42	Strings, PCAP, sandbox
Domain	update.microsoft-cdn[.]xyz	DNS queries, strings
File Hash	sha256:e3b0c442...	Dropped files
File Path	%APPDATA%\svchost.exe	Sandbox, registry
Registry Key	HKCU\...\Run\svchost	Registry monitor
Mutex	Global\MYHACKTOOL2024	API trace
User-Agent	Mozilla/5.0 (custom)	Network capture
Email	ransom@protonmail[.]com	Ransom note

AI Enrichment Workflow

Extract IoCs from report → Query VirusTotal, AbuseIPDB, Shodan → Feed results to LLM → Get threat actor attribution, campaign links, and recommended blocks.

VirusTotal-Style Report Analysis

LLMs can parse and summarize multi-vendor detection results to accelerate triage.

```
{
  "sha256": "e3b0c44298fc1c149afbf4c8996fb924...",
  "file_name": "invoice_march2024.exe",
  "file_size": 1245184,
  "file_type": "PE32 executable (GUI) Intel 80386",
  "detection_stats": {
    "malicious": 48, "suspicious": 3, "undetected": 12
  },
  "vendor_detections": {
    "CrowdStrike": "Win/Ransomware.CustomLocker",
    "Microsoft": "Ransom:Win32/CustomLock.A!cl",
    "Kaspersky": "Trojan-Ransom.Win32.CustomLocker.gen",
    "Sophos": "Troj/Ransom-QKL",
    "ESET": "Win32/Filecoder.CustomLocker.A"
  }
}
```

AI Summary

High-confidence ransomware detection (48/63 vendors). Family: **CustomLocker**. First seen March 2024. UPX-packed PE32. Contacts known malicious infrastructure. Recommend immediate blocking of all network IoCs.

Network IoCs — Deep Dive

Network indicators from PCAP analysis and sandbox output reveal command-and-control patterns.

```
# DNS Queries (from sandbox)
2024-03-15 10:22:01  A      update.microsoft-cdn[.]xyz → 185.243.115.42
2024-03-15 10:22:15  A      api.legit-service[.]top   → 91.215.85.17
2024-03-15 10:23:00  TXT    check.miner-pool[.]cc     → "stratum+tcp://..."

# HTTP Traffic
POST /gate.php HTTP/1.1
Host: 185.243.115[.]42
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64)
Content-Type: application/octet-stream
Content-Length: 256
[encrypted_beacon_data]

# Exfiltration (later in timeline)
POST /upload.php HTTP/1.1
```

Detection Tip

Look for DNS queries to recently registered domains, periodic beaconing patterns (fixed intervals), and large outbound POSTs. The `X-Campaign-ID` header is a tracking artifact left by the malware operator.

Machine Learning for Malware Classification

Beyond signatures and rules, ML models can classify malware at scale using features extracted from samples.

Feature Engineering Approaches

Static Features

- PE section entropy distribution
- Import/export function counts
- String statistics (avg length, charset)
- File size and structure ratios
- Byte n-gram frequency vectors
- Opcode sequence patterns

Dynamic Features

- API call sequences and frequency
 - Network behavior patterns
 - File system operation counts
 - Registry modification patterns
 - Memory allocation behavior
 - Process creation trees
-

ML Classification Pipeline

```
import numpy as np
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report

# Feature vector: [entropy, import_count, string_count,
#                  file_size, has_upx, suspicious_api_count,
#                  section_count, avg_section_entropy]
X = np.array([
    [7.98, 42, 380, 1245184, 1, 8, 5, 6.54], # ransomware
    [6.12, 28, 150, 524288, 0, 3, 4, 5.21],  # benign
    [7.85, 55, 290, 819200, 1, 12, 6, 7.01], # trojan-rat
    [5.90, 15, 95, 102400, 0, 1, 3, 4.88],  # benign
    # ... thousands more samples
])
```

Notes: Real-world models use thousands of features and need regular retraining as malware evolves. EMBER and SOREL-20M are public datasets for malware ML research.

ML Approaches Comparison

Approach	Strengths	Weaknesses
Random Forest	Fast, interpretable, good baseline	Struggles with novel families
Gradient Boosting (XGBoost)	High accuracy, handles imbalanced data	Slower training, risk of overfitting
CNN on raw bytes	No feature engineering needed	Computationally expensive, black box
RNN/LSTM on API calls	Captures temporal behavior	Needs dynamic analysis data
Transformer models	Best on decompiled code analysis	Largest compute requirements
LLM-based classification	Flexible, uses natural language context	Expensive per-query, non-deterministic

Practical Recommendation

Use **gradient boosting** for bulk classification, **YARA rules** for known families, and **LLMs** for novel sample triage and report generation. Layer your defenses.

MITRE ATT&CK Mapping with AI

The MITRE ATT&CK framework provides a structured taxonomy of adversary techniques. AI can automatically map observed behaviors to ATT&CK.

PROMPT:

Map these observed malware behaviors to MITRE ATT&CK techniques:

1. Creates scheduled task "WindowsUpdate" for persistence

2. Injects code into explorer.exe via process hollowing

3. Disables Windows Defender via registry modification

4. Queries http://ip-api.com/json for geolocation

5. Encrypts files on all drives with .locked extension

6. Deletes volume shadow copies

7. Drops ransom note to every directory

8. Exfiltrates data to C2 before encryption

Behavior	Technique ID	Tactic
Scheduled task	T1053.005	Persistence
Process hollowing	T1055.012	Defense Evasion

Behavior	Technique ID	Tactic
Disable Defender	T1562.001	Defense Evasion
Geolocation check	T1016	Discovery
File encryption	T1486	Impact
Delete shadow copies	T1490	Impact
Ransom note	T1491.001	Impact
Data exfiltration	T1041	Exfiltration

Threat Intelligence Integration

Connecting malware analysis findings to broader threat intelligence accelerates response and attribution.

Intelligence Sources

- **VirusTotal** — multi-AV scan, behavior
- **AbuseIPDB** — IP reputation
- **Shodan/Censys** — infrastructure recon
- **MISP** — threat sharing platform
- **AlienVault OTX** — open threat intel
- **URLhaus** — malicious URL tracking

AI-Powered Enrichment

- Cross-reference IoCs across all sources
- Identify related campaigns and actors
- Predict likely next attack stages
- Generate actionable blocking rules
- Auto-create STIX/TAXII objects
- Draft stakeholder notifications

```
# Quick enrichment script
import requests

def enrich_ip(ip, vt_key, abuseipdb_key):
```

```
vt = requests.get(f"https://www.virustotal.com/api/v3/ip_addresses/{ip}",
                  headers={"x-apikey": vt_key}).json()
abuse = requests.get("https://api.abuseipdb.com/api/v2/check",
                    params={"ipAddress": ip},
                    headers={"Key": abuseipdb_key}).json()

return {
    "ip": ip,
    "vt_malicious": vt["data"]["attributes"]["last_analysis_stats"]["malicious"],
    "abuse_score": abuse["data"]["abuseConfidenceScore"],
    "country": abuse["data"]["countryCode"],
```

YARA Rule — Cryptominer Detection

Detecting cryptomining malware that hijacks system resources.

```
rule Cryptominer_Generic {  
  meta:  
    author = "BlueTeam CTF"  
    description = "Detects cryptocurrency mining malware"  
    mitre_attack = "T1496"  
  
  strings:  
    // Mining pool protocols  
    $stratum1 = "stratum+tcp://" ascii wide  
    $stratum2 = "stratum+ssl://" ascii wide  
  
    // Common mining pool domains  
    $pool1 = "pool.minexmr.com" ascii  
    $pool2 = "xmrvpool.eu" ascii  
    $pool3 = "monerohash.com" ascii
```

YARA Rule — Cobalt Strike Beacon

Detecting one of the most common post-exploitation tools found in enterprise compromises.

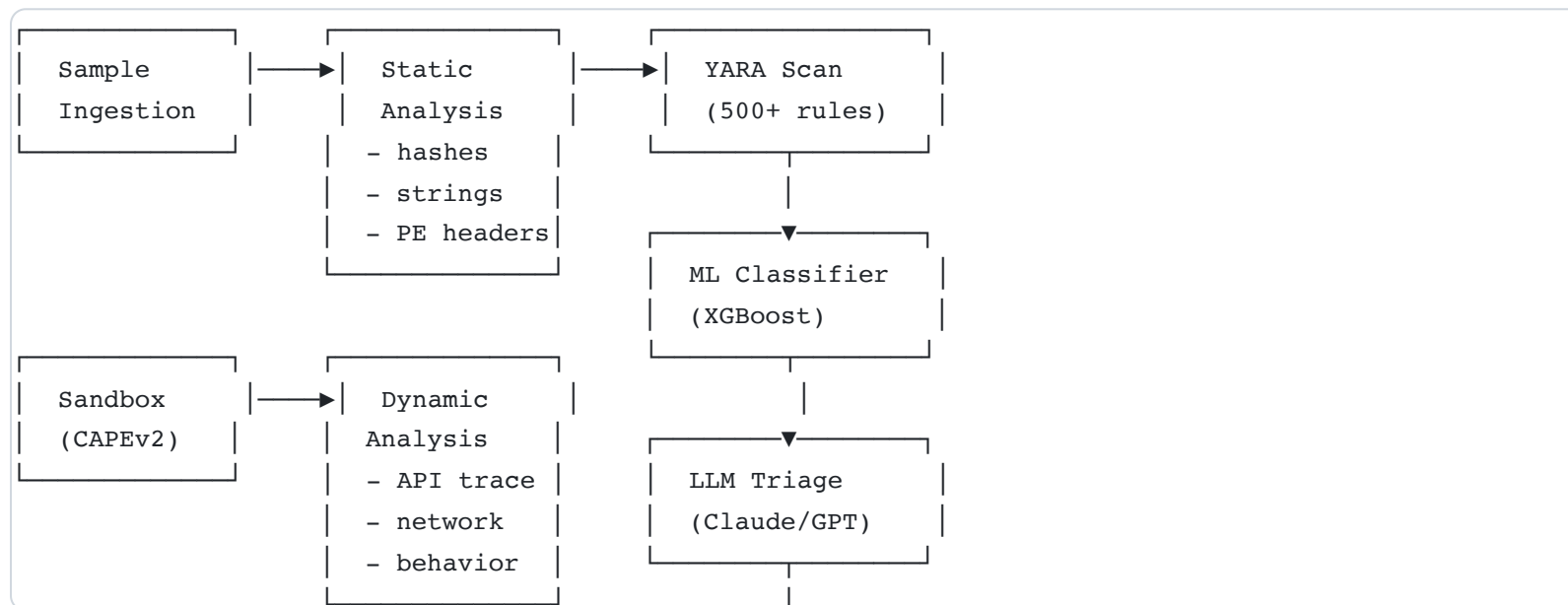
```
rule CobaltStrike_Beacon_Generic {
  meta:
    author = "BlueTeam CTF"
    description = "Detects Cobalt Strike Beacon payloads"
    mitre_attack = "T1071.001, T1573"
    severity = "critical"

  strings:
    // Default named pipes
    $pipe1 = "\\.\pipe\msagent_" ascii
    $pipe2 = "\\.\pipe\MSSE-" ascii
    $pipe3 = "\\.\pipe\%" ascii

    // Beacon config markers
    $cfg1 = { 00 01 00 01 00 02 ?? ?? 00 02 00 01 00 02 }
```

Building a Detection Pipeline

Combining all techniques into an automated malware analysis and detection workflow.



ELF Malware — Linux Considerations

Linux malware is increasingly common, targeting cloud infrastructure, containers, and IoT devices.

```
# ELF header analysis
$ readelf -h suspicious_elf
ELF Header:
  Class:                               ELF64
  Type:                               EXEC (Executable file)
  Machine:                             Advanced Micro Devices X86-64

# Check for static linking (common in Go/Rust malware)
$ file suspicious_elf
suspicious_elf: ELF 64-bit LSB executable, x86-64, statically linked,
Go BuildID=abc123..., stripped

# Extract Go build metadata
$ go version suspicious_elf
suspicious_elf: go1.21.5
```

Go and Rust binaries are increasingly popular for cross-platform malware. Statically linked, large file sizes, harder to reverse engineer. Watch for XMRig miners, botnet clients, and ransomware targeting NAS/ESXi.

Anti-Analysis Techniques

Understanding how malware evades analysis helps you bypass these protections.

Technique	Detection	Bypass
Packing (UPX, Themida)	High entropy sections	Unpack or dump from memory
String encryption	Low readable string count	Dynamic analysis, emulation
VM detection	Checks for VMware/VBox artifacts	Harden sandbox environment
Time-based evasion	Sleep calls, GetTickCount	Patch sleep, accelerate time
Anti-debug	IsDebuggerPresent, INT 3	Patch checks, ScyllaHide
Code obfuscation	Control flow flattening	Symbolic execution, LLM analysis

AI Advantage

LLMs can analyze obfuscated decompiled code and identify the underlying logic even when traditional pattern matching fails. Feed Ghidra output directly to Claude for explanation.

CTF Challenge: Malware Classification & Detection

🚩 Challenge Brief

Classify these malware samples and write detection signatures for each.

You are given five unknown samples. For each sample, you must:

1. **Classify** — determine the malware type and family
2. **Extract IoCs** — identify all network, file, and registry indicators
3. **Map to ATT&CK** — identify at least 3 MITRE techniques per sample
4. **Write YARA rules** — create a detection rule for each sample
5. **Generate report** — produce a concise analyst report using AI assistance

Samples

Sample	Hint
<code>sample_A.exe</code>	High entropy, contacts mining pools
<code>sample_B.dll</code>	Loaded via rundll32, named pipe comms
<code>sample_C.elf</code>	Statically linked Go binary, targets SSH
<code>sample_D.exe</code>	Drops ransom note, modifies boot record
<code>sample_E.doc</code>	Macro-enabled, downloads second stage

CTF Challenge — Scoring Rubric

Per Sample (20 pts each)

- **4 pts** — Correct classification (type + family)
- **4 pts** — IoC extraction (min 5 per sample)
- **4 pts** — MITRE ATT&CK mapping (min 3 techniques)
- **5 pts** — Working YARA rule (must not false-positive on clean set)
- **3 pts** — Analyst report quality

Bonus Points

- **+5 pts** — Cross-sample correlation (shared infrastructure)
- **+5 pts** — Threat actor attribution with evidence
- **+5 pts** — Automated analysis pipeline script
- **+3 pts** — Fuzzy hash matching between variants

Tools Allowed

`strings`, `file`, `readelf`, `pefile`, `yara`, `ssdeep`, any LLM (Claude, ChatGPT), VirusTotal, CyberChef. Sandbox VM provided.

Key Takeaways

- **Malware is diverse** — modern threats blend multiple categories (trojan + RAT + ransomware)
- **Static before dynamic** — always triage with hashes, strings, and headers before executing in a sandbox
- **YARA is essential** — the lingua franca of malware detection; combine string, byte, and condition logic
- **AI accelerates analysis** — feed strings, API traces, and decompiled code to LLMs for rapid triage and classification
- **ML scales detection** — gradient boosting on static/dynamic features handles volume that humans cannot
- **ATT&CK provides structure** — map every behavior to a technique for consistent reporting and threat intel sharing
- **Layer your defenses** — YARA rules + ML classifiers + LLM triage + threat intel = comprehensive detection

Next Lesson

Lesson 04: Security Hardening — proactive defense through system configuration, attack surface reduction, and compliance baselines.
