CHAPTER 10

Malware Analysis

Chapter Overview

01

Online Analysis Services

VirusTotal, sandbox services, and threat intelligence feeds

02

Static Analysis

File properties, strings, PE headers, and YARA rules

03

Dynamic Analysis

Behavior monitoring in isolated environments

04

Reverse Engineering

Disassembly and decompilation for deep understanding

Online Analysis Services

Fast initial triage with online tools

Online Malware Analysis Tools

VirusTotal

Submit file hash or file for multi-engine AV scanning

Behavioral sandbox reports (via integrated partners)

IMPORTANT: submitting files may expose sensitive data

Prefer hash lookups over file uploads in sensitive cases

Community comments often include IOC context

Sandbox Services

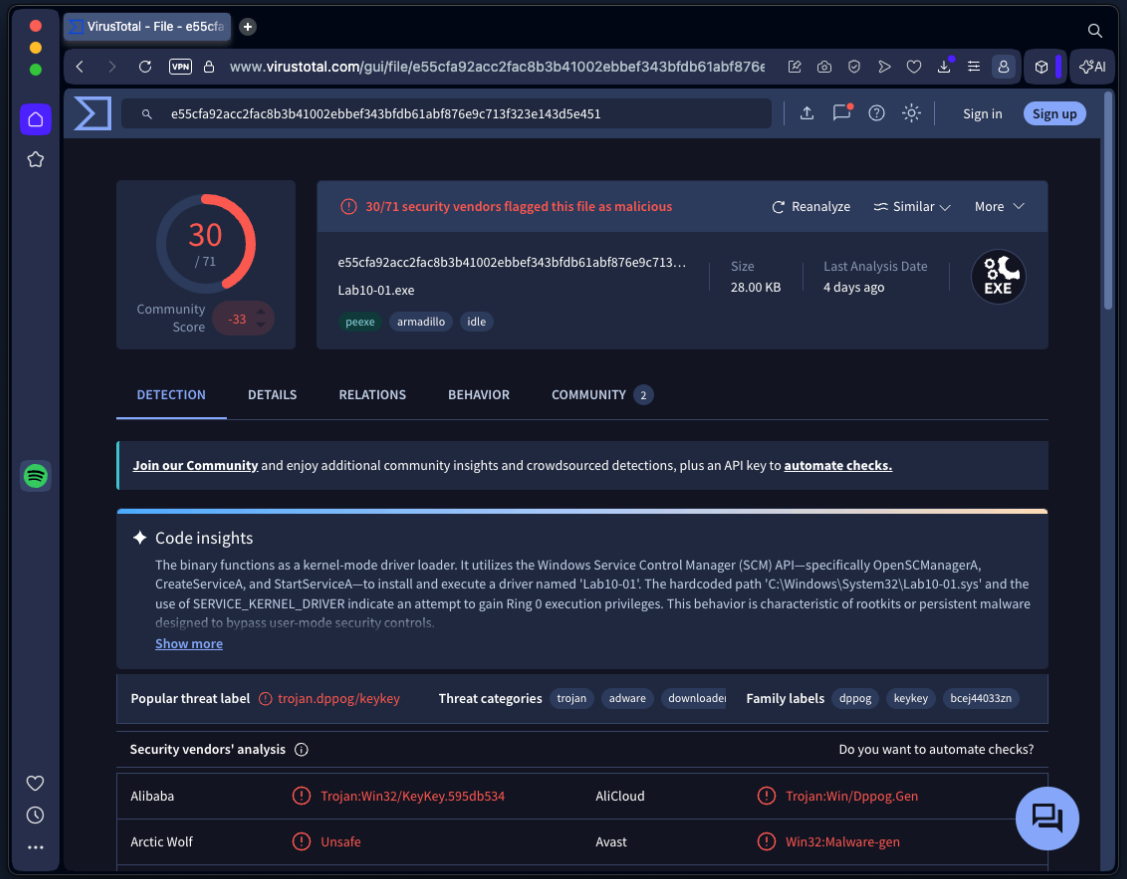
Any.run: interactive sandbox — watch malware execute in real time

Hybrid Analysis: automated sandbox with YARA and network IOCs

Joe Sandbox: deep behavioral analysis, Windows + Android

MalwareBazaar: sample repository for threat researchers

Triage (Hatching): fast, automated, community-shared reports



https://www.virustotal.com/gui/file/e55cfa92acc2fac8b3b41002ebbef343bfdb61abf876e9c713f323e143d5e451/detection

Static Analysis

Examine the file without running it

Static Analysis Techniques

File hashing: MD5/SHA-256/imphash — lookup against threat intel databases

String extraction: strings -n 8 malware.exe → reveals URLs, registry keys, commands

FLOSS (FireEye): extract obfuscated strings that strings misses

PE header analysis: imports, exports, sections, timestamps, compiler artifacts

Suspicious imports: VirtualAlloc + WriteProcessMemory + CreateRemoteThread = injection

Section entropy: high entropy (.text section) = packed/encrypted code

Tools: PEStudio, PE-bear, CFF Explorer, DIE (Detect It Easy)

PE File Format Anatomy

PE (Portable Executable): file format for Windows executables (.exe, .dll, .sys)

DOS header: 'MZ' magic bytes at offset 0 — identifies file as a PE

PE header: machine type (x86/x64), timestamp, characteristics flags

Section table: **.text** (code), **.rdata** (read-only data), **.data** (writeable), **.rsrc** (resources)

Import Directory Table (IDT): DLLs and functions imported — key for static analysis

Export table: functions exposed by DLLs — look for unusual exports in renamed .exe

Resources (.rsrc): embedded files, icons, strings — malware often hides payloads here

Entropy: randomness per section — packed/encrypted = high entropy (>7.0)

YARA Rules — Writing and Using

YARA: pattern matching language for malware identification — works on files and memory

Rule structure: rule name { meta: ... strings: ... condition: ... }

String matching: $str = 'CreateRemoteThread' — match literal strings

Hex pattern: $hex = { 4D 5A 90 00 03 00 } — match byte patterns (useful for shellcode)

Regex: $re = /powershell.*-enc[a-z]*/i — match encoded PS execution patterns

Condition: all of them — all strings must match; any of them — at least one match

Running: yara rule.yar /path/to/scan/ — scan files

Community rules: valhalla.nextron-systems.com, signature-base (GitHub/Neo23x0)

Identifying Packing and Encryption

Packed malware: compressed or encrypted payload that decompresses at runtime

Indicators: very few imports (only LoadLibrary + GetProcAddress), high section entropy

DIE (Detect It Easy): identifies packers, compilers, and obfuscation methods

Common packers: UPX (most common), Themida, VMProtect (very complex)

UPX unpacking: upx -d malware.exe — unpacks standard UPX; patched UPX needs manual work

Manual unpacking: run in VM + x64dbg → find OEP (Original Entry Point) → dump

Encrypted config: look for byte arrays > 256 bytes with high entropy near string decryption loop

Strings before/after unpacking: very few strings packed; hundreds after unpacking

Dynamic Analysis

Watch malware execute safely

Manual & Automated Dynamic Analysis

Manual Dynamic Analysis

Isolated VM + snapshot (ALWAYS revert after each run)

FakeNet-NG: simulates network services to capture C2 traffic

Process Monitor (ProcMon): file, registry, network activity

Process Hacker: process and memory viewer

Regshot: compare registry before/after malware execution

Wireshark: capture all network activity

Sandbox Detection Evasion

Sleep(): malware sleeps longer than sandbox timeout

VM artifact checks: registry keys, process names, CPUID

User interaction checks: mouse movement, active window

Domain join check: sandbox is often standalone

Counter: patch sleep calls, add VM artifacts, automate interaction

Sandbox Evasion Techniques & Countermeasures

Sleep-based evasion: Sleep(300000) — waits longer than sandbox timeout

Counter: patch sleep calls to NOP or short value before running in sandbox

VM detection: check registry keys (HKLM\SOFTWARE\VMware), process names, CPUID hypervisor bit

Counter: add realistic VM artifacts, use hardware passthrough (nested virtualization)

User interaction: no mouse movement after 5 minutes = sandbox; counter: simulate movement

Screen resolution: 800x600 = sandbox; counter: set to 1920x1080

Domain join check: sandbox is standalone; counter: join test VM to a domain

DLL injection evasion: some samples check if security product DLLs are loaded first

Automated Dynamic Analysis Workflow

Step 1 — Prepare VM: snapshot known-clean state; set up FakeNet-NG, ProcMon, Regshot, Wireshark

Step 2 — Baseline: Regshot 1st snapshot before execution; record running process/registry state

Step 3 — Execute: run sample with admin rights (many samples check for admin before activating)

Step 4 — Monitor: watch ProcMon for file/registry writes; Wireshark for network connections

Step 5 — FakeNet: note DNS queries, HTTP requests, C2 callback attempts

Step 6 — Regshot compare: before/after diff shows ALL registry changes by malware

Step 7 — Collect IOCs: file paths, registry keys, network destinations, mutex names

Step 8 — ALWAYS revert snapshot before next sample — never analyze on dirty VM

Reverse Engineering

Understanding malware at the code level

Reverse Engineering Fundamentals

Disassembly: convert binary to assembly (IDA Pro, Ghidra, Binary Ninja, Radare2)

Decompilation: convert binary to pseudo-C code (IDA Pro + Hex-Rays, Ghidra, RetDec)

Ghidra: NSA-developed, free, excellent decompiler — start here

Key functions to find: main(), WinMain(), DllMain(), exported functions

Networking: look for WSAStartup, connect, HttpSendRequest, InternetOpen

Persistence: RegSetValueEx, CreateService, WriteFile to startup location

Anti-analysis: IsDebuggerPresent, RDTSC timing, GetTickCount checks

Decompiler Comparison — Ghidra, IDA, Binary Ninja

Ghidra: NSA-developed, free, Java-based — excellent decompiler, good for most malware

IDA Pro + Hex-Rays: gold standard for commercial RE — most accurate decompiler, expensive

Binary Ninja: modern UI, Python API for automation, excellent for shellcode and loaders

Radare2 / Cutter: open-source, command-line and GUI — highly scriptable, steep learning curve

RetDec: online decompiler at retdec.retdec.com — quick check without local tooling

Workflow: Ghidra for first pass → IDA for complex obfuscation → Binary Ninja for scripted work

Symbol resolution: Ghidra auto-imports type libraries; IDA has extensive FLIRT signature DB

Collaboration: Ghidra server allows team sharing of analysis across analysts

Identifying Key Malware Behaviors in Code

Network: WSAStartup → socket → connect → send / recv — basic C2 socket communication

HTTP: InternetOpen → InternetConnect → HttpOpenRequest → HttpSendRequest → InternetReadFile

Persistence: RegSetValueEx (registry) → CreateService → WriteFile to startup paths

Process injection: VirtualAllocEx → WriteProcessMemory → CreateRemoteThread

Hollowing: CreateProcess SUSPENDED → ZwUnmapViewOfSection → VirtualAllocEx → WriteProcessMemory → ResumeThread

Anti-debug: IsDebuggerPresent → exit or behave differently if debugger detected

Encoding: XOR loops, Base64 decode, custom ROT/shift ciphers for C2 traffic obfuscation

Lateral movement: NetShareEnum + CopyFile + service install via SCM (OpenSCManager)

Conclusion

Online services provide rapid triage — but avoid uploading sensitive files

Static analysis extracts IOCs (hashes, strings, imports) without execution risk

Dynamic analysis reveals behavior: C2, persistence, lateral movement actions

Sandbox evasion is common — manual analysis may be required for sophisticated malware

YARA rules turn findings into repeatable detection across your entire environment

Reverse engineering provides the deepest insight but requires the most skill and time

Knowledge Check