Intro to Malware Analysis

ft. Anti-analysis and anti-anti-analysis techniques

UMASS CYBERSEC CLUB

What is Malware? What do you know about it?

Turn and talk for 5 minutes!

UMASS CYBERSEC CLUB

What is malware analysis?

Given a malware sample, how can we:

- Figure out its behavior?
- Classify it based on existing malware samples?
- Triage its threat level?

UMASS CYBERSEC CLUB

Approaches to malware analysis

There are 2 main branches of reverse engineering in general:

- **Static analysis:** We look at the binary and try to determine the behavior from there. Manually, you can use Ghidra to decompile the binary into a more readable form.
- **Dynamic analysis:** We run the binary within a controlled environment and analyze its behavior directly. Manually, you can use a debugger (e.g. gdb) to set breakpoints at control instructions, manipulating variables as needed.

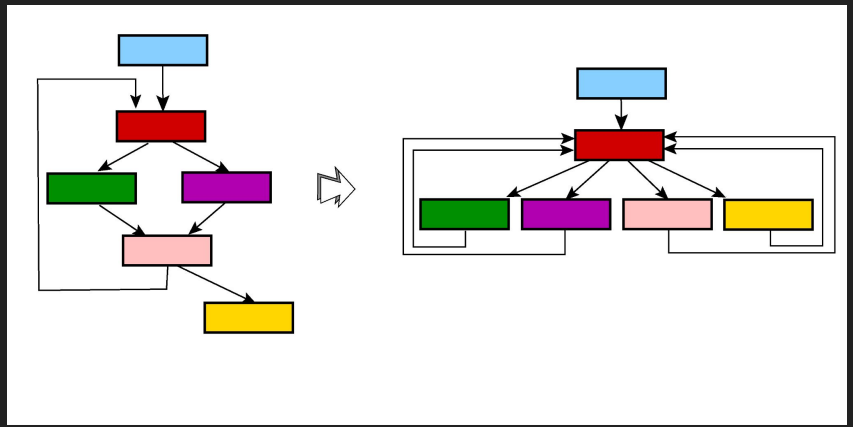
This talk will primarily focus on Dynamic analysis.

Why is static analysis hard?

- In static analysis, **extra obfuscation** can be applied to the control flow of the program in order to hide what code would actually be executed.
- In fact, automating analysis would run straight into **the halting problem**: no program should be able to always correctly determine the behavior of every program. The programs we care about aren't exactly arbitrary, but there is unlikely an **efficient solution**.
- Malware can also get information from **external sources**, such as a control-and-command server.

Example: control flow flattening

- For example, one can turn an entire program into a loop with a switch statement for what code block to execute next.
- If you've taken 250, this would be like turning an normal program into a state machine, removing meaningful labels and adding random junk states.



UMASS CYBERSEC CLUB

Automating dynamic analysis

- The plan is simple: we run the program somewhere safe and observe.
  - What system calls does it actually use? For example, which files are being read and written?
  - What external servers does it connect to? (command-and-control, data exfiltration)
  - Does it obtain external payloads? How do the external payloads behave?
  - Does it make use of well-known libraries for cryptography? (ransomware)
- Of course, this is not foolproof either, but we can more effectively make use of the assumption that the malware will try to execute malicious code whenever possible.
- Manually attaching a debugger and continuing step-by-step would be more comprehensive, but we can capture a lot of information about a “typical run” just by automating the data collection process above.

Example: CAPE sandbox

- CAPE sandbox is an open-source project for automating analysis of malware.
- Lots of customizability, kind of a pain to set up.
- Also automates extraction of malware configuration if the sample belongs to well-known family.
- Used by well-known sites like VirusTotal.

Official demo site: https://capesandbox.com/analysis

Activity: reviewing an analysis result

We'll take a look at an example analysis result:

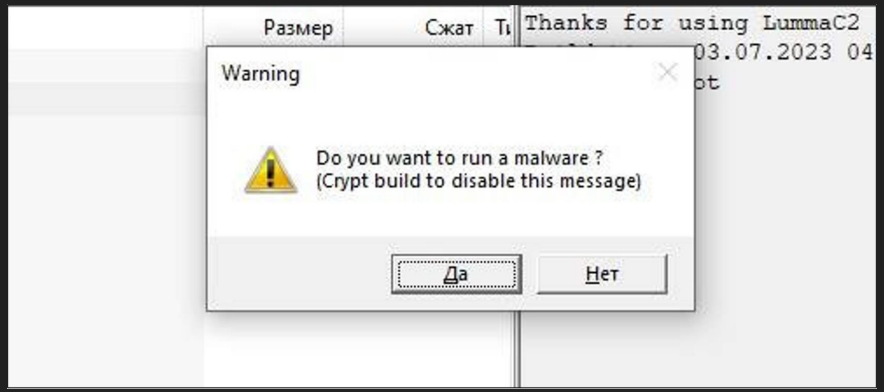
https://capesandbox.com/analysis/27762/

Think about how the information present on the page would help:

- A malware analyst figure out the behavior of the malware?
- A incident response team member figure out what went down?
- How can you prevent the same sample from infecting user machines again?

Why is dynamic analysis hard?

- External command-and-control server dies => payload might not be available.
- Modern malware developers are aware of these analysis techniques, and are starting to employ active countermeasures against them.
- Around 30% of recent samples use some form of anti-analysis.

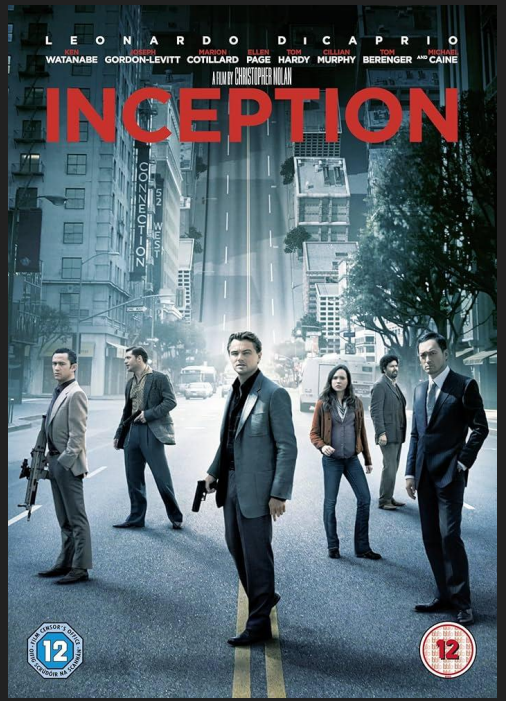


UMASS CYBERSEC CLUB

Anti-analysis techniques

We'll look at the problem from the perspective of a malware developer.

- As it turns out, we should not let the malware sample run on every machine.
- If we detect an environment that seems "fake", we should not "detonate" the sample.
- What are signs of an environment made specifically for analysis?



UMASS CYBERSEC CLUB

Anti-anti-analysis techniques

Hypervisor: Software that manages the VM & the communication between them and the host kernel

Now let's look at this from the perspective of a malware analyst:

- You have control over the environment. This means that you can modify **literally almost anything** to allow you to counteract these measures:
  - Hypervisor-level modification: Make changes to match bare metal behavior.
  - VM/OS-level modification: Modify VM and OS configurations, e.g. add external hardware tables, hook into system calls.
  - Program-level modification: Modify the binary directly, specifically capture behavior from the program and spoof return values.
- However, it is significantly harder to automate against these measures, since:
  - Many of them exhibit the same behavior as a non-malicious program.
  - Similar techniques need not behave the same way.

Virtual environment checks

Most analysis environments involve a hypervisor running virtual machines. Malware can simply choose not to run on VMs. How can VMs be detected?

- For efficiency reasons, modern emulators often chooses to not emulate hardware if they don’t have to.
- For similar reasons (and bugs), modern emulators do not handle every single instruction the same way a physical processor would.

Technique: Sleep modes

- Modern Windows laptops have S4 sleep along with either S0 or S3 sleep.
- Almost all popular VM vendors do not support S0 or S3 by default:
  - KVM/QEMU supports neither by default, but could be configured (with a lot of troubleshooting) to support S3.
  - VMWare supports S1 by default and has features that acts like S3 sleep but isn't considered so by the firmware.
  - VirtualBox also does not support the feature by default.
- To circumvent this, there are 2 approaches:
  - Approach 1: dynamically spoof the return value from the checks.
  - Example: https://github.com/kevoreilly/capemon/pull/102
  - Approach 2: implement the sleep mode in the virtual firmware.

Technique: ACPI tables

- ACPI tables give you all the information about the supposed hardware the system is running on. Many hypervisors self-report here.
- Basic checks include brand name checks, CPU core/thread mismatch, low-memory configuration or having too few tables.
- More advanced checks include looking at the metadata inside the tables, hardcoded hardware IDs from the hypervisor, etc.
- To circumvent this, we would need to patch the hypervisor itself.

Technique: Checking parent process

- The parent process for a malware should be predictable:
  - User runs executable: should be explorer.exe.
  - Specific malicious program or automated tool: should be whatever that program is.
- Typically, debuggers and automated tools will spawn the process directly, which would result in the malware not being run by explorer.exe.
- Malware can easily check for this, and it is not abnormal for a process to check its parent's process.
- This is not impossible to circumvent, but can be annoying to automate against.

UMASS CYBERSEC CLUB

Technique: Check for tampering

- Debugging tools will load a program's code into memory, and then potentially modify the code to add breakpoints, or directly try to inject itself.
- Programs must have read permission to read their own process memory, which includes code to be executed.
- To check for integrity, a program can check the loaded code against its own binary to prevent tampering.
- Furthermore, it can check against a fixed hash provided externally to avoid modifications made directly to the binary.
- Not easy to circumvent, depends a lot on how the check is actually performed.

Technique: Basic waiting check

- Delay execution of payload for a long time to circumvent automated tools - they are often configured with a timeout.
- To circumvent this, many automated tools will skip sleeps.
- To circumvent that, malwares will check if the timing between the sleep commands actually matched the expected timing.
- To circumvent all of that, we can spoof the system calls used for timings.

UMASS CYBERSEC CLUB

Technique: Timing-based detection

- “VM exits” happen when the guest needs the host to perform privileged action, such as querying for hardware info.
- These instructions takes significantly longer on most hypervisors, since control has to be passed around more than running on bare metal.
- Timing-based detection measures how long these instructions take compared to instructions that don’t cause VM exits and should have similar duration.
- More advanced measures would use an external time server to counteract local timing methods, or directly measuring against the reported clock speeds.
- It is hard to properly circumvent without killing performance.

Resources

- Wiki for popular techniques: https://evasions.checkpoint.com/
- Open-source implementation of techniques: https://github.com/kernelwernel/VMAware
- Some mitigations against anti-VM techniques: https://github.com/ChrisEric1/Hypervisor-Phantom

  (the OG repo got nuked, idk why)

- User space debugger that allows automated hooking of Windows API calls: https://github.com/kevoreilly/capemon

NECCDC Applications open soon!

If you’re interested in blue teaming, this is a competition where you defend a simulated corporate network against industry professional hackers!

- Fill out our interest form!
- Application materials will be released next week.

Interest form

UMASS CYBERSEC CLUB

Join us on Friday for a CTF!

Play an international CTF alongside the team, or get guided practice on our training platform!

Friday, 9/26 | 4-7 PM | LGRC A104

(Rootkit Part 2 Delayed)

UMASS CYBERSEC CLUB