CS7038 - Malware Analysis - Wk08.2

Dynamic Analysis and Run-Time Debugging

Coleman Kane
kaneca@mail.uc.edu

March 1, 2018

UNIVERSITY OF Cincinnati

Static Analysis Limitations

Often, it is difficult to determine what's going on with static analysis alone. The techniques discussed so far are very helpful for safely extracting identifiable information from malware so that you may identify it when you receive it.

Another aspect to analysis is endpoint, network, & similar behavioral detection techniques. When the attack has already breached into a network, learning the side effects or behaviors of the malware in a target environment is also necessary, and isn't frequently quick to get via static analysis.

Enter Run-Time Analysis

Run-Time analysis involves opening or kicking off the attack inside of a controlled environment. There are two common approaches to this:

- VM Hypervisor or Emulation
- "Bare Metal" hardware malware lab

In the observance of limited financial resources, we will be focusing on using VMs. Specifically using VirtualBox as the container (`https://virtualbox.org/`).

Run-Time Attack Surface

I like to think of run-time analysis as trying to analyze the *run-time attack surface* the adversary may have to work within.

Generally speaking, this could be carved up into multiple bins:

- Filesystem changes (paths / directories)
- Execution environment (CPU, RAM)
- Operating System and User configurations
- Persistence mechanisms
- System and user object resources
- Network

Filesystem Modifications

In many cases, malware as well as attackers actions when utilizing remote access will involve making filesystem changes. On most systems, there's a volume directory (or Master File Table, in Windows) and sometimes even a filesystem journal, that keeps an index linking names of filesystem objects to their content.

Filesystem Impact Example

Frequently, the attack chain involves multiple actions that expose an intrusion on the filesystem. Here's an example:

1. User downloads BAD.PDF into Downloads folder (creation)
2. User opens BAD.PDF (access)
3. BAD.PDF exploits Acrobat Reader
4. Attacker shellcode writes GOOD.PDF and BAD.EXE into %TEMP% folder (creation)
5. Attacker shellcode opens both GOOD.PDF and BAD.EXE using WinAPI (access)
6. BAD.EXE deletes BAD.PDF (expunge/delete)
7. BAD.EXE writes a symlink in Startup folder in Start Menu (creation)
8. BAD.EXE establishes communication link with external *Command and Control* server

Execution Environment

The term "execution environment" is the broad term I use to describe the part of the system where execution occurs. In our model environment we largely consider this to be the CPU (where execution instructions are streamed through), system memory, and swap memory.

We discussed some obfuscation techniques earlier. In many cases these can be applied to persistent code as well, concealing the existence of malware on-disk, but it still needs to be decrypted in memory in order to execute.

Analysis of this component of the attack surface typically involves a combination of memory image collection (there are numerous tools for this), as well as controlling execution through use of debuggers and other process-management utilities.

Execution Environment Impact Example

Using the same example:

1. User downloads BAD.PDF into Downloads folder
2. User opens BAD.PDF
3. BAD.PDF exploits Acrobat Reader (arbitrary code execution / memory writes)
4. Attacker shellcode writes GOOD.PDF and BAD.EXE into %TEMP% folder
5. Attacker shellcode opens both GOOD.PDF and BAD.EXE using WinAPI (program creation)
6. BAD.EXE deletes BAD.PDF
7. BAD.EXE writes a symlink in Startup folder in Start Menu
8. BAD.EXE establishes communication link with external *Command and Control* server

Operating System and User Configurations

Most normal working systems are configured to operate under a decently high level of security. Once an adversary achieves access, it is not uncommon for them to attempt to reconfigure a system such that it will operate under a lower security level.

Some examples of configuration changes that might be employed:

- Default proxy settings
- User-authentication check
- Event logging
- Service changes

In particular, Windows contains a persistent configuration database known as the *System Registry* where much of the system configuration and state information is managed.

Operating System/User Configuration Impact Example

Using the same example:

1. User downloads BAD.PDF into Downloads folder
2. User opens BAD.PDF
3. BAD.PDF exploits Acrobat Reader
4. Attacker shellcode writes GOOD.PDF and BAD.EXE into %TEMP% folder (user-profile folder lookup)
5. Attacker shellcode opens both GOOD.PDF and BAD.EXE using WinAPI (Operating system call, maybe uncommon for Acrobat Reader to call this)
6. BAD.EXE deletes BAD.PDF
7. BAD.EXE writes a symlink in Startup folder in Start Menu
8. BAD.EXE establishes communication link with external *Command and Control* server (network resources accessed)

Persistence Mechanisms

A subset of configuration is called *Persistence Mechanisms*. These are used for maintaining presence on a machine that *persists* across reboots, user logins, network changes, etc.

Some common methods for persistence include:

- Linking or copying malware in the "Startup" folder in the start menu
- Registry entries instructing program execution on startup
- Hijacking or creating a service encapsulating the malware
- Replacing or hijacking common runtime libraries to load malware
- Scheduled tasks

Replication methods can also, in part, be incorporated into this set of activities.

Persistence Impact Example

Using the same example:

1. User downloads BAD.PDF into Downloads folder
2. User opens BAD.PDF
3. BAD.PDF exploits Acrobat Reader
4. Attacker shellcode writes GOOD.PDF and BAD.EXE into %TEMP% folder
5. Attacker shellcode opens both GOOD.PDF and BAD.EXE using WinAPI
6. BAD.EXE deletes BAD.PDF
7. BAD.EXE writes a symlink in Startup folder in Start Menu (BAD.EXE will start next time user logs in)
8. BAD.EXE establishes communication link with external *Command and Control* server

Network Traffic

To maintain remote access or at least transmit data back to the adversary, malware needs to make use of the existing built network at the home or business the target is working from.

Much of the malware ecosystem employs functionality that isn't traditionally implemented as part of standard network traffic. In some cases, it also employs proprietary means to conceal itself. For these reasons, it is common for malware providing access or data theft to employ unique remote-access protocols. Sometimes, these are implemented over an existing standard, like HTTP, and other times there are binary traffic protocols entirely unique to the malware (such as Gh0st).

There are a few common ways these are leveraged in analysis:

- Monitoring, traffic capture, compare to known threats
- Simulated network services to explore malware functionality

Persistence Impact Example

Using the same example:

1. User downloads BAD.PDF into Downloads folder (data is transferred TO the system)
2. User opens BAD.PDF
3. BAD.PDF exploits Acrobat Reader
4. Attacker shellcode writes GOOD.PDF and BAD.EXE into %TEMP% folder
5. Attacker shellcode opens both GOOD.PDF and BAD.EXE using WinAPI
6. BAD.EXE deletes BAD.PDF
7. BAD.EXE writes a symlink in Startup folder in Start Menu
8. BAD.EXE establishes communication link with external *Command and Control* server (bidirectional, adversary-controlled data movement) (malware protocol visible)

System-wide Resources and Handles

The operating system acts as a resource manager for your computer. One of the resources it manages are data structure instances for user and system level objects. In a traditional PC, there are a lot of mechanisms enabling applications to intercommunicate and also for programs to manage access to limited resources. The employment of these mechanisms may also suggest an intrusion. Examples may include:

- System pipes
- Mutexes
- Semaphores
- Listening network sockets
- Lock files