# Basic Static analysis

Arvind S Raj
(arvindsraj@am.amrita.edu)

16SN708 Malware Analysis

M.Tech CSN Jul-Nov 2017

# Lecture overview

Techniques for basic static analysis

- Using anti-malware scanners and hashes to confirm malware.
- Learning more about malware by inspecting it's contents at superficial level.
- Mostly focused on using tools.

# Basic static analysis: objectives

- Quick analysis of sample to determine if already known malware.
- Extract information by inspecting malware using tools.
- Not comprehensive but quick and provides useful and easy to find information about malware.
- Can reveal lot of information and guide next steps in analysis.

# Broad steps in sequence

- Scan using anti-malware engines to find if already known.
- Share malware sample with other analysts to see if they have encountered this.
- View strings in the sample.
- Inspect executable: libraries, functions, resources etc and file structure.
- Apply some simple tests for well-known protections.

# Outline

# Anti-malware scans

- Check malware or it's hash with anti-malware software.
- Most popular choice: VirusTotal.
- Share hash/sample with malware analysts **you know and trust**.
- Helps avoid unnecessary and speed up analysis.

# Malware hashes

- Unique ID for correctly identifying malware samples.
- Use cryptographic hashing algorithms. Can you name some?

# Malware hashes

- Unique ID for correctly identifying malware samples.
- Use cryptographic hashing algorithms. Can you name some?
- MD5, SHA1, SHA256 etc.
- Preferred choice today SHA256 but others still used.

# Exercise!

Check the hashes on VirusTotal to see which are malicious and what information it provides.

# Outline

# Strings primer

- **String**: A contiguous sequence of characters from a specific language.
- Can reveal useful information about the malware such as IP addresses, functions used, DLLs loaded and more.
- Discoverable using special tools that scan for strings.
- Two types of string: ASCII and UNICODE.

# ASCII string encoding

- Original and most popular character encoding scheme.
- Character is 7-bits in length.
- Superseded by other encoding schemes; backward-compatible.
- Eg: “CSN” = hex string 43 53 4e 00. Why the ”00” at end?

# UNICODE string encoding

- Standard encoding scheme used today.
- Multiple variants exist: UTF-8 and UTF-16 most popular.
- UTF-8 extremely popular: uses 1 to 4 bytes per character.
- Windows uses UTF-16 like variant. 2 bytes per character.
- Eg: "CSN" = hex string 4300 5300 4e00 0000.

# ASCII and UNICODE exercise

- What is the ASCII and UNICODE representation of "malware"?

# ASCII and UNICODE exercise

- What is the ASCII and UNICODE representation of "malware"?
- ASCII: 6d 61 6c 77 61 72 65.
- UNICODE: 6d00 6100 6c00 7700 6100 7200 6500.

# Viewing strings in an executable

- Use “strings” tool. Prints all printable strings it finds.
- Usage: strings.exe </path/to/file>. Try it on file1.xex.
- Also available on Linux.
- Be careful - malware may exploit vulnerability in strings! Run command in a VM/safe environment.

# Viewing strings in an executable(cont.)

- strings detects too many strings - many of which not useful.
- Enter IDA: the de-facto disassembler used in the industry.
- Extremely powerful features, many plugins and extremely expensive.
- Does a better job at detecting strings. Load file1.xex and see strings it finds!

# Viewing strings in an executable(cont.)

- Do you find any interesting strings in file1.xex?

# Viewing strings in an executable(cont.)

- Do you find any interesting strings in file1.xex?
- “deflate 1.2.3 Copyright 1995-2005...”.
- Lots of HTTP request strings.
- What information do these give you?

# Viewing strings in an executable(cont.)

- Do you find any interesting strings in file1.xex?
- "deflate 1.2.3 Copyright 1995-2005...".
- Lots of HTTP request strings.
- What information do these give you?
- Probably uses zlib compression and sends HTTP requests to some server. Very useful information!

# Viewing strings in an executable(cont.)

- Strings give away quite a bit of information useful for analysis.
- Malware authors "hide" strings for this reason. Makes analysis difficult.
- String decrypted during runtime or some similar technique.
- See strings in file2.xex. Check with strings and then IDA.

# Outline

# Introduction

- PE = Portable Executable. Nearly all Windows executables of this type.
- PE format describes structure of the entire executable and it's contents.
- Proper understanding helps infer useful information during static analysis.
- Also useful to identify if malware author has adopted any tricks.

# PE file format

- Consists of a header, code, data, resources, external libraries/functions used etc.
- Header useful for finding out what information is stored where in the file.
- Two parts: a DOS header(for compatibility) and a PE header.
- **Sections**: Remainder of file divided into many parts consisting of various information.

# PE sections

- **.text**: Consists of program code.
- **.data**: Consists of global variables.
- **.rsrc**: Consists of resources used by executable - icons, strings, images etc.
- **.rdata**: Consists of imports, exports and maybe read only global data. More on imports and exports in a while.

# PE sections(cont.)

- More sections can exist. These are most popular.
- No restriction on section names: can be named anything.
- Regular software stick to common names.
- Malware changes names and even adds new sections!

# Analysing PE file format

Let's investigate the PE file format by analysing calc.exe using PEview and PEstudio.

# Analysing PE file format(cont.)

Let's now take a look at file structure of a malicious executable(file01.xex).

# file01.xex PE structure

- Additional section: saber. Non-standard name.
- Saber section is executable and writeable.
- Saber section is entry point section and not .text.
- Version information: possible fakes standard Microsoft values.

# Analysing PE file format(cont.)

Let's now take a look at file structure of a malicious executable(file02.xex).

# file02.xex PE structure

- Non-standard section names: UPX0 and UPX1.
- UPX0 and UPX1 are executable and writeable.
- UPX0 has 0 size in file but 405KB size in memory.

# Linking to Libraries and Functions

- Many useful functionality implemented in libraries.
- *Export*ed by libraries and *import*ed by binaries.
- Import-export connection established via linking.
- Three types: static, runtime and dynamic linking.

# Linking to Libraries and Functions(cont.)

- **Dynamic**: Most common. Specify which libraries are required and OS loads them when needed. Easily identifiable.
- **Static**: Directly copy library functions' code into executable. No more loading during runtime but no longer possible to easily identify them and bigger binaries.
- **Runtime**: Load libraries and find functions manually during program execution. Masks which libraries and functions are used while keeping binary size small.

# Understanding dynamic linking

Let's look at imported DLLs and functions in calc.exe using Dependency Walker, IDA pro and PE studio.

# Understanding dynamic linking(cont.)

Let's look at imported DLLs and functions in file01.xex using Dependency Walker, IDA pro and PE studio.

# Understanding dynamic linking(cont.)

- Linking against DLLs and using specific functions gives clues about program behaviour.
- Many functions in Windows standard libraries. Impossible to remember all.
- Look up detailed descriptions of functions in Windows documentation when encountering a new function.
- With time and practice, you will learn to identify interesting from uninteresting function calls.

# PE format summary

- Helps understand structure of the executable.
- Careful analysis reveals lot of information about executable.
- Anomalies in section sizes, names etc can indicate possibly malicious nature.
- Imported DLLs and function names provide more clues on linking and other behaviour of executable.

# Static analysis exercise

- Given these imports and exports, can you identify what the malware probably does?
- List of functions imported and exported uploaded to AUMS.
- A string from binary: "Software\Microsoft\Windows \CurrentVersion \Run"
- Use MSDN documentation to figure out what the functions perform and try to infer behaviour.

# Static analysis exercise solution

- OpenProcess, GetCurrentProcess and GetProcessHeap $\Longrightarrow$ process manipulation features.
- ReadFile, WriteFile, FindFirstFileW, FindNextFileW $\Longrightarrow$ file manipulation features.
- LowLevelMouseProc and LowLevelKeyboardProc: call back functions for SetWindowsHookExW.
- SetWindowsHookExW used to setup callbacks when specific events occur. Here - keyboard and mouse events.

# Static analysis exercise solution(cont.)

- RegisterHotKey: Registers a hotkey combination to invoke an application.
- RegOpenKeyExW, RegCloseKey et al $\Longrightarrow$ Windows registry is queries and modified. Registry controls among many things startup application.
- String "Software\Microsoft\Windows \CurrentVersion\Run"
- With this information can you guess what this malware does?

# Static analysis exercise solution(cont.)

- Malware is a keylogger.
- Records keypresses and mouse movements using SetWindowsHookExW.
- Saves them to a file using file operations.
- Has GUI components and hotkey installed: possibly to view recorded keylog.
- Adds itself to system startup using registry.

# Outline

# Motivation

- Static analysis is quite powerful: reveals lot of useful information.
- Sometimes easily determine what a program does without even executing it!
- Malware authors want to make it more difficult to detect and analyse malware.
- Slow down malware analysts $\Longrightarrow$ slow down detection and containment of malware in wild.

# Packing and obfuscation

- Objective: Make it harder to analyse by hiding many easily findable and useful information.
- Can you name some information that could be hidden?

# Packing and obfuscation

- Objective: Make it harder to analyse by hiding many easily findable and useful information.
- Can you name some information that could be hidden?
- Strings, libraries and functions used, program code, section names.
- More details in later lectures. Let's look at effects.

# Packing and obfuscation(cont.)

Let's analyse file02.xex using PEstudio to see what are effects of packing.

# Observations

- Valid PE but section names are strange.
- Common sections(.text, .data etc) are missing.
- Sections with 0 raw size present.
- Libraries and functions used seem pretty less.
- Verdict: file02.xex is probably packed/obfuscated.

# Defeating packing/obfuscation

- Can we somehow disable the packing/obfuscation?
- In nearly all cases, yes!
- Difficulty depends on packer or obfuscation technique used.
- Simple and well known techniques easily bypassed. Others not as simple as this.

# Detecting packing/obfuscation

- Some common techniques are predictable in effects.
- Scripts and tools can look for these and detect the technique used.
- Similar to how anti-malware work: look for patterns in behaviour, content etc to detect malware.
- Let's test out DiE, exeinfo and PE detective on file02.xex.

# Detecting packing/obfuscation(cont.)

- 2 out of 3 tools detected the well known packer UPX.
- UPX easily defeated: scripts are available to undo it's effects.
- After undoing, analysis is much more simpler!
- Let's analyse uncompressed version of file02.xex using PEstudio.

# Detecting packing/obfuscation(cont.)

- All techniques may not be as easily detectable as UPX.
- Let's analyse file03.xex using the same 3 tools.

# Detecting packing/obfuscation(cont.)

- All techniques may not be as easily detectable as UPX.
- Let's analyse file03.xex using the same 3 tools.
- Only 1 out of 3 detects the technique used.
- All tools look only for well-known techniques. Can't detect new techniques adopted.

# Conclusion

- Static analysis quite powerful in information provided.
- Can unravel nearly everything about the malware.
- Malware authors adopt techniques to make this quick analysis hard.
- Some techniques can be easily broken. Others will take time but not impossible.