Malware Analysis
Tools and Techniques

Jan Arends

Hochschule Bonn-Rhein-Sieg

11. Jan 2018

Overview

1. Introduction
   - Why
   - Goals
   - Save environment

Overview

1. Introduction
   - Why
   - Goals
   - Save environment
2. Static Analysis
   - File Identification
   - Embedded Artifact Extraction
   - Code Reversing

Overview

1. Introduction
    - Why
    - Goals
    - Save environment
2. Static Analysis
    - File Identification
    - Embedded Artifact Extraction
    - Code Reversing
3. Dynamic Analysis
    - Monitoring
    - Memory forensics
    - Debugging

Overview

1. Introduction
   - Why
   - Goals
   - Save environment
2. Static Analysis
   - File Identification
   - Embedded Artifact Extraction
   - Code Reversing
3. Dynamic Analysis
   - Monitoring
   - Memory forensics
   - Debugging
4. Conclusion

Why

Motivation

- Number of malware is still growing
- Malware is getting more complex
- Analysing is important because for
  - Recovery
  - Defeating
  - Eliminating
  - Detection

Goals

To answer questions like . . .

- Nature and purpose of the program
- What type of malware is it?
- What is the intended purpose and how does it accomplish it?
- What is the functionality and capability?
- What affect does it have on the system?
- How does is spread?

Overview

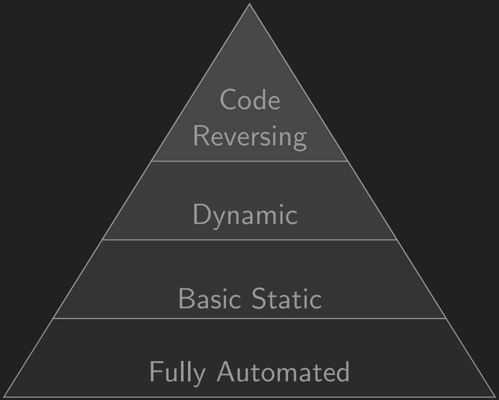


Figure: Stages of Malware Analysis

Overview

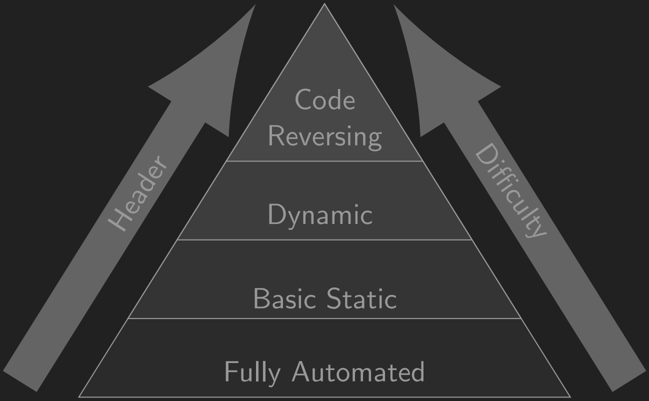


Figure: Stages of Malware Analysis

Lab

Safty first

- Potentially damaging code
- Safe and secure lab environment
- Place the file on an isolated or sandboxed system
- Ensure that the code is contained
- Unable to connect to or otherwise affect any production system

1. Introduction
   - Why
   - Goals
   - Save environment
2. Static Analysis
   - File Identification
     - File Formats
     - Fingerprint
     - Obfuscation
   - Embedded Artifact Extraction
   - Code Reversing
3. Dynamic Analysis
   - Monitoring
   - Memory forensics
   - Debugging
4. Conclusion

Formats

- Have to be identified for proper anaylsis
- File signatures are used
- File identification tool or binary artifacts

Examples

- Portable Executable (PE)
- Executable and Linkable Format (ELF)
- Mach-O

Fingerprint

- Hash value for identification
- MD5 or SHA-1
- Malware might already be explored
- Antivirus scanning: Useful first Step

Tool

VirusTotal

Obfuscation

- Code is converted to a different version
- Semantic equivalence
- Different fingerprint
- Example techniques:
    - Junk code insertion
    - Pattern-based obfuscation
    - Stack-based obfuscation
    - . . .

Wrapper

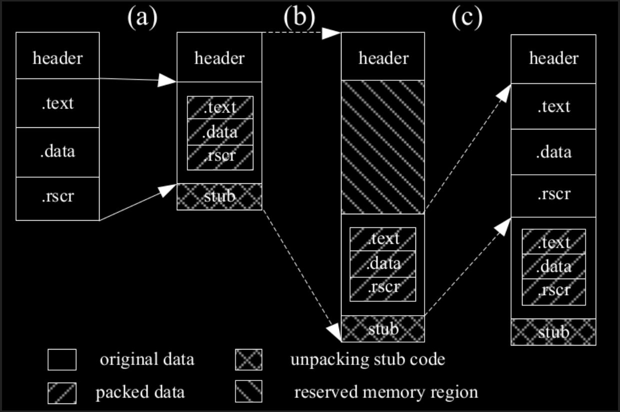


Figure: Typical packing and unpacking process [1]

1. Introduction
    - Why
    - Goals
    - Save environment
2. Static Analysis
    - File Identification
    - Embedded Artifact Extraction
        - Strings
        - Dependencies
        - Symbols
        - Metadata
    - Code Reversing
3. Dynamic Analysis
    - Monitoring
    - Memory forensics
    - Debugging
4. Conclusion

Vorbereitung I

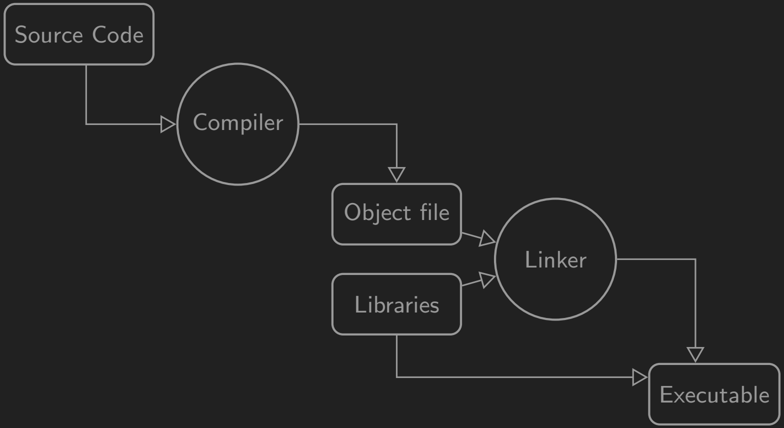


Figure: Creation of an executable

Vorbereitung II

Linux	- GNU C Library manual - The Open Group index of functions - Linux man-pages
Windows	- Windows API reference - Microsoft DLL Help Database - TechNet Library

Table: Web references

Strings

Infos embedded in binary as ASCII or Unicode format

- Calls to functions, shared libraries and APIs
- Error messages and Comments
- Network information
- IRC Channels and C&C Server
- Directory and file names
- Compiler and versions

Tool

`Strings`

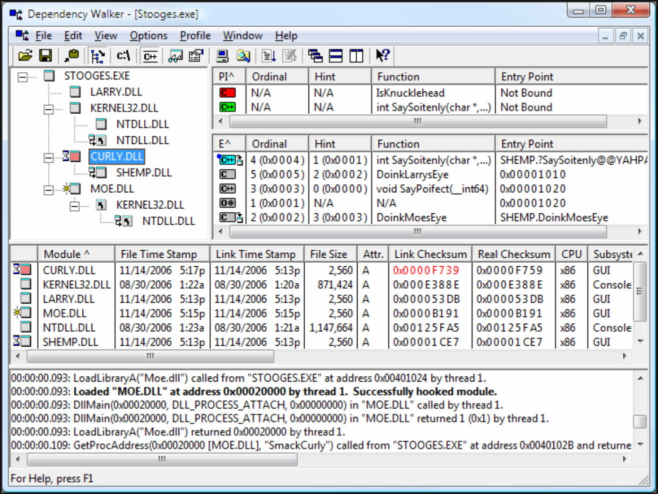
Dependencies I

- Identifying within the string search is a good starting point
- Additional research
- Good guess about behavior
- Actually called functions can be explored

Tool

- Windows: Dependency Walker
- Unix/Linux: `ldd`

Dependencies Tool Beispiel



Symbols

- Stored in a symbol table
- Identifiers for program variables and functions
- Variable names, addresses, data types and scopes
- Structure and class definitions

Requieres

Author didn't discarded the symbol information

Tool

- Windows: DUMPBIN
- Unix/Linux: `eu-nm`

Metadata

- Plant during the creation of an executable
- Can be generated from various parts of the file structure
- Information about the origin, ownership, and history
- Timestamps, location and previous file names

1 Introduction
- Why
- Goals
- Save environment

2 Static Analysis
- File Identification
- Embedded Artifact Extraction
- Code Reversing
  - Disassambly
  - Decompiled code

3 Dynamic Analysis
- Monitoring
- Memory forensics
- Debugging

4 Conclusion

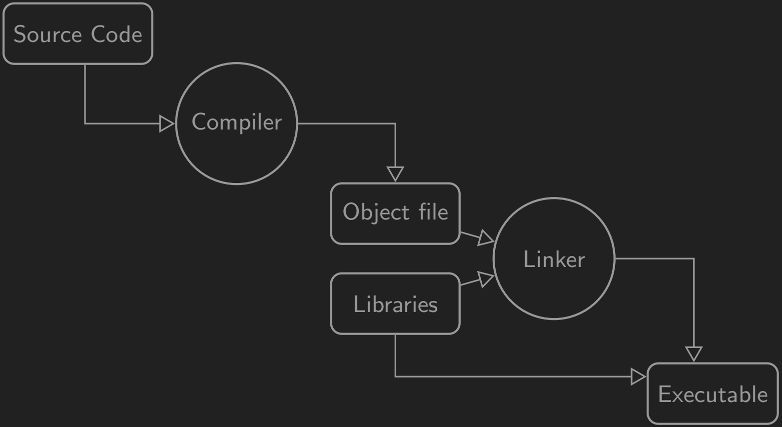


Figure: Creation of an executable

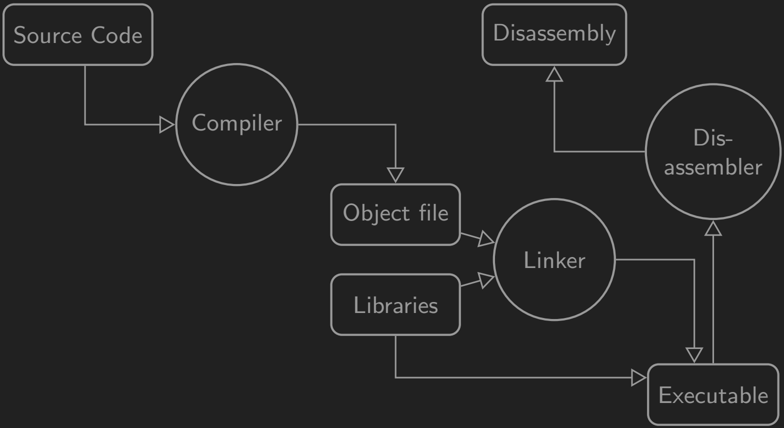


Figure: Creation of an executable

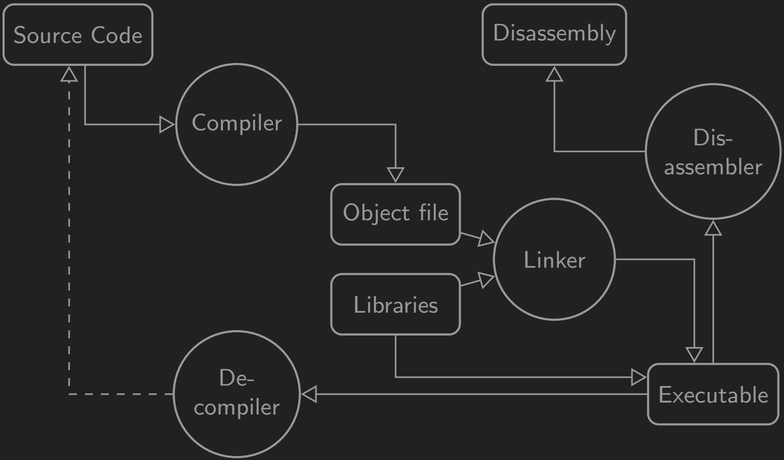


Figure: Creation of an executable

Disassambly

- Disassambler: Recovers a low-level language code
- Human-readable version of a computer architecture's instruction set
- Reliably and consistently
- Instructions: One-to-many relationship
- Analyzing instructions as groups

Tool

IDA (Interactive Disassembler) von Hey-Rays

Example

```
int i;
for(i=0; i<100; i++) {
        printf("i equals %d\n", i);
}
```

Listing 1: For-loop written in C

```
00401004               mov     [ebp+var_4], 0
0040100B               jmp     short loc_401016
0040100D loc_40100D:
0040100D               mov     eax, [ebp+var_4]
00401010               add     eax, 1
00401013               mov     [ebp+var_4], eax
00401016 loc_401016:
00401016               cmp     [ebp+var_4], 64h
0040101A               jge     short loc_40102F
0040101C               mov     ecx, [ebp+var_4]
0040101F               push    ecx
00401020               push    offset aID ; "i equals %d\n"
00401025               call    printf
0040102A               add     esp, 8
0040102D               jmp     short loc_40100D
```

Listing 2: Assembly code for the for loop example above

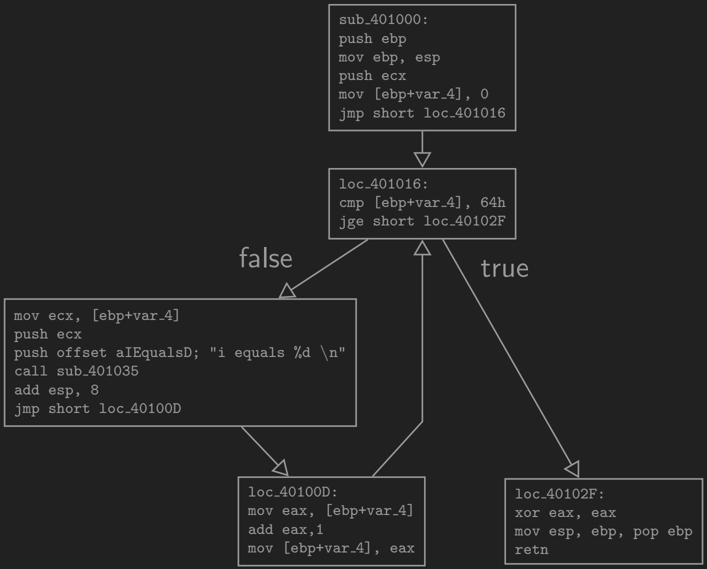


Figure: Graph view for the for loop example above

Decompiled code

- Decompiler: Recovers high-level form of code
- Abstractions and structures inclusive
- Research topic

Major improvements in 2016 and 2017

- Goto-free DREAM and extended version DREAM^{++}
- Retargetable Decompiler RetDec by Avast

Beispiel

```
void *__cdecl sub_10006390(){
        __int32 v13; // eax@14
        int v14; // esi@15
        unsigned int v15; // ecx@15
        int v16; // edx@16
        char *v17; // edi@18
        bool v18; // zf@18
        unsigned int v19; // edx@18
        char v20; // dl@21
        char v23; // [sp+0h] [bp-338h]@1
        int v30; // [sp+30Ch] [bp-2Ch]@1
        __int32 v36; // [sp+324h] [bp-14h]@14
        int v37; // [sp+328h] [bp-10h]@1
        int i; // [sp+330h] [bp-8h]@1
        // [...]
        v30 = *"qwrtpsdfghjklzxcvbnm";
        v37 = *"eyuioa";
        // [...]
        v14 = 0;
        v15 = 3;
        if ( v13 > 0 )
        {
                v16 = 1 - &v23;
                for ( i = 1 - &v23; ; v16 = i )
                {
                        v17 = &v23 + v14;
                        v19 = (&v23 + v14 + v16) & 0x80000001;
                        v18 = v19 == 0;
                        if ( (v19 & 0x80000000) != 0 )
                                v18 = ((v19 - 1) | 0xFFFFFFFE) == -1;
                        v20 = v18 ? *(&v37 + dwSeed / v15 % 6)
                                  : *(&v30 + dwSeed / v15 % 0x14);
                        ++v14;
                        v15 += 2;
                        *v17 = v20;
                        if ( v14 >= v36 )
                                break;
                }
        }
        // [...]
}
```

Listing 3: Hey-Rays

Beispiel Fortsetzung

```
LPVOID sub_10006390(){
        int v1 = *"qwrtpsdfghjklzxcvbnm";
        int v2 = *"eyuioa";
        // [...]
        int v18 = 0;
        int v19 = 3;
        if(num > 0){
                do{
                        char * v20 = v18 + (&v3);
                        int v21 = v18 + 1;
                        int v22 = v21;
                        int v23 = v21 & 0x80000001L;
                        bool v24 = !v23;
                        if(v23 < 0)
                        v24 = !(((v23 - 1) | 0xfffffffeL) + 1);
                        char v25;
                        if(!v24)
                        v25 = *(((dwSeed / v19) % 20) + (&v1));
                        else
                        v25 = *(((dwSeed / v19) % 6) + (&v2));
                        v18++;
                        v19 += 2;
                        *v20 = v25;
                        }while(v18 < num);
        }
        // [...]
}
```

Listing 4: Dream

```
LPVOID sub_10006390(){
        char * v1 = "qwrtpsdfghjklzxcvbnm";
        char * v2 = "eyuioa";
        // [...]
        int v13 = 3;
        for(int i = 0; i < num; i++){
                char v14 = i % 2 == 0 ? v1[(dwSeed / v13) % 20]
                                : v2[(dwSeed / v13) % 6];
                v13 += 2;
                v3[i] = v14;
        }
        // [...]
}
```

Listing 5: Dream++

Dynamic Analysis

Intro

- Running the malware
- Observing its behavior
- Overcomes anti static techniques
- Anti techniques exists:
  - Monitoring Detection
  - VM detection
  - Memory capturing detection

1. Introduction
   - Why
   - Goals
   - Save environment
2. Static Analysis
   - File Identification
   - Embedded Artifact Extraction
   - Code Reversing
3. Dynamic Analysis
   - Monitoring
     - Processes
     - System & API calls
     - File system & Registry
     - Network
   - Memory forensics
   - Debugging
4. Conclusion

Monitoring

Basic actions

- File Actions $= \{Create, Delete, Read, Write, Rename\}$
- Process Actions $= \{Create, Terninate\}$
- Network Actions $= \{TCP, UDP, IP\} \oplus \{Listen, Connect, Send, Recv\}$
- Registry Actions $= \{OpenKey, CloseKey, CreateKey\} \wedge \{SetValue, DeleteValue, QueryValue\}$

Processes

- Program in execution
- Explore all the processes related to the malware
- Which are running, created or terminated
- Loaded shared libraries

Tools

Windows: Process Monitor
Linux: htop

System & API calls

- Provided by the operating system
- Low level communications only via system calls, e.g.
  - File system interactions
  - Network functionality
- Tracking of parameters and return values
- great chance to get further knowledge
- Conformation of founded calls in string search

File system & Registry

- Used for persistence or configuration data
- Malware often uses the Windows registry
- Snapshot comparison

Tools

Regshot

Network

- Malware often requires internet access
- Additional components must be downloaded
- Capture packages in simulated network
- Analyzing packages, filter suspicious connections
- Reverse-engineering of the protocol
- Exploration of downloaded files

Tools

- Wireshark
- tcpdump
- Bro

1. Introduction
   - Why
   - Goals
   - Save environment
2. Static Analysis
   - File Identification
   - Embedded Artifact Extraction
   - Code Reversing
3. Dynamic Analysis
   - Monitoring
   - Memory forensics
     - Acquisition
     - Analysis
   - Debugging
4. Conclusion

Memory forensics

- Visibility into the runtime state of the system
- Full reconstruction of events
- Every action end up in specific modifications in RAM
- Actions performed can persist a long time after it was taken

Acquisition

- Process of copying the contents of volatile memory to non-volatile storage
- Happens in controlled environment (Lab)
- How: capturing, dumping or sampling
- When:
  - Terminating-Based
  - Interval-Based
  - Trigger-Based

Analysis

- Happens on another environment "offline"
- No API calls $\rightarrow$ unallocated or hidden data can bee seen
- Visibility into the runtime state
  - Processes
  - Network connections
  - Executed commands
  - ...

Tool

Volatility Framework

Assembly-level debugging

- Information that would be difficult to get from a disassembler
- Ability to measure and control a program's execution
- Every memory location, register, and argument to every function
- Change and test variables
- Values of memory addresses as they change throughout the execution
- Conditional Breakpoints

Tools

OllyDbg, IDA

Conclusion

Summary

- Multiple techniques to use to full capacity
- Basic analysis: simple tools, easy to use and understand
- Advanced analysis: working knowledge and practice necessary
- Still a lot to research for reverse engineering

Future

- Containing race between malware authors and analysts
- Anti techniques will get even smarter
- Detection methods of those as well
- Fully automated approach might be satisfying in near future
- Artificial intelligence, Deep Learning, Neural Networks helpful

Literatur I

- G. Yuxin, L. Zexin, and L. Yuqing, "Survey on malware anti analysis," in *Fifth International Conference on Intelligent Control and Information Processing*, 2014.
- M. Sikorski and A. Honig, *Practical Malware Analysis: A Hands-On Guide to Dissecting Malicious Software*. No Starch Press, 2012.
- E. K. M. Egele, T. Scholte and C. Kruegel, *A Survey on Automated Dynamic Malware-Analysis Techniques and Tools*. ACM Computing Surveys, Vol. 44, ss, 2012.
- C. Malin, E. Casey, and J. Aquilina, *Malware Forensics: Investigating and Analyzing Malicious Code*. Elsevier Science, 2008.

Literatur II

M. Kerrisk, *Linux Manual Page: ldd(1)*.
https://www.kernel.org/doc/man-pages/, September 2017.

Dependency Walker: Using Dependency Walker for General Information about Modules.
www.dependencywalker.com/help/html/overview_3.htm.

REMux.
www.remnux.org/.

I. You and K. Yim, "Malware obfuscation techniques: A brief survey," in *International Conference on Broadband, Wireless Computing, Communication and Applications*, 2010.

Literatur III

K. Yakdan, S. Eschweiler, E. Gerhards-Padilla, and M. Smith, "No more gotos: Decompilation using pattern-independent control-flow structuring and semantics-preserving transformations," in *??*, 2015.

K. Yakdan, S. Dechand, E. Gerhards-Padilla, and M. Smith, "Helping johnny to analyze malware: A usability-optimized decompiler and malware analysis user study," in *2016 IEEE Symposium on Security and Privacy*, 2016.

J. T. Streib, *Guide to Assembly Language - A Concise Introduction*.
Springer, 2011.

Literatur IV

- M. Ligh, S. Adair, B. Hartstein, and M. Richard, *Malware Analyst's Cookbook and DVD: Tools and Techniques for Fighting Malicious Code*.
Books 24x7 IT PRO, Wiley, 2010.
- D. Distler, *Malware Analysis: An Introduction*.
SANS Institute, 2007.
- P. Ren, L. W., S. D., W. J., and L. K., "Analysis and forensics for behavior characteristics of malware in internet," in *14th Annual Conference on Privacy, Security and Trust (PST)*, 2016.
- A. Software, *Retargetable Decompiler RetDec*.
`https://retdec.com/`, December 2017.

Literatur V

M. Hale Ligh, A. Case, J. Levy, and A. Walters, *The Art of Memory Forensics: Detecting Malware and Threats in Windows, Linux, and Mac Memory*.
John Wiley & Sons, Inc., 2014.

T. Haruyama and H. Suzuki, *One byte Modification for Breaking Memory Forensic Analysis*.
https://media.blackhat.com/bh-eu-12/Haruyama/bh-eu-12-Haruyama-Memory_Forensic-Slides.pdf, 2012.

T. Teller and A. Hayon, "Enhancing automated malware analysis machines with memory analysis," BlackHat, 2014.

Literatur VI

- T. Shibahara, T. Yagi, M. Akiyama, D. Chiba, and T. Yada, "Efficient dynamic malware analysis based on network behavior using deep learning," in *2016 IEEE Global Communications Conference (GLOBECOM)*, 2016.
- J. Aycock, *Computer Viruses and Malware*. Advances in Information Security, Springer US, 2006.

The End

Fragen?