Automated Malware Behaviour Analysis

Gérard Wagener[12] **Alexandre Dulaunoy**[2] **Radu State**[13]

[1]**MADYNES - LORIA**
Laboratoire Lorrain de Recherche en Informatique et ses Applications

[2]**CSRRT-LU**
Computer Security Research and Response Team - Luxembourg

[3]**INRIA**
Institut National de Recherche en Informatique et Automatique

October 19, 2007

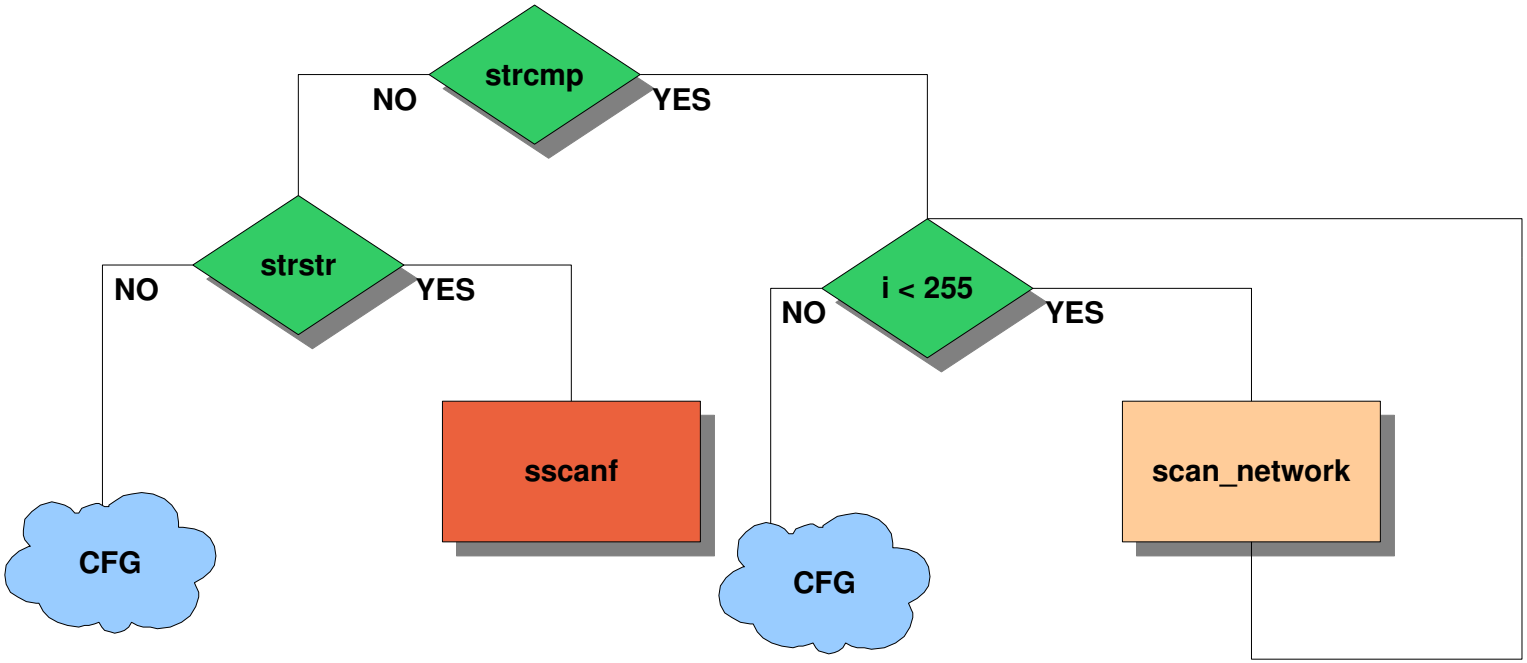
Outline

Introduction

- [Aycock, 2006] there are two malware analysis techniques:
  - Static Analysis: which is often hard to do due to packers and encryption techniques.
  - Dynamic Analysis: which is often incomplete due to conditions that are not fulfilled.
- [Wagener, 2007] et al. proposed the usage of phylogenetic trees in order to identify malware families based on sequences of called system functions.
- [Wagener, 2007] et al. proposed a design of a framework for analyzing malware. (Automated aNalysis and Network Emulation, ANNE)

Introduction

A piece of malware can have multiple behaviors.



Introduction
Contribution

- [Moser, 2007] et al. explored multiple execution paths of malware using snapshots from CPU emulators.
- We want to explore other strategies to unveil more information from malware.
- In order to discover such strategies we propose a tool fiw which:
  1. provides an interactive sandbox $\longrightarrow$ high-level debugger.
  2. tackles with current applied anti-debugging or anti - reverse engineering techniques.

Fiw - the debugger

Goal: Analyzing malware (unknown w32 binaries)

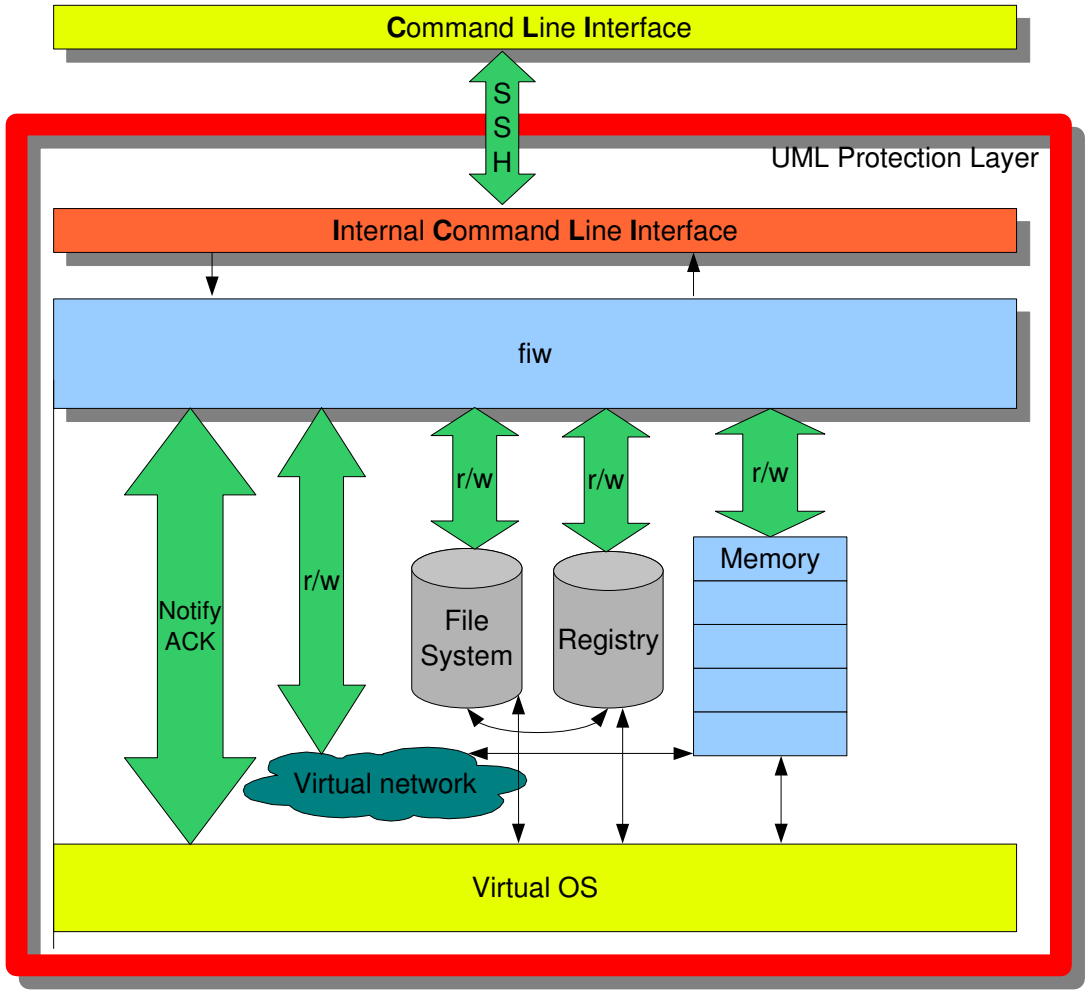
- High - level observations:
  - File system & registry modifications.
  - Malware networking.
    - Discover communication protocols of a malware sample.
- Low - level observations:
  - System function call observation.
  - Discovering packing / encryption techniques.
  - Discovering vulnerabilities $\longrightarrow$ dump & study machine code.
- Actions:
  - Apply fuzzing techniques via fiw - script interface or plugins.

Fiw - the debugger

Features

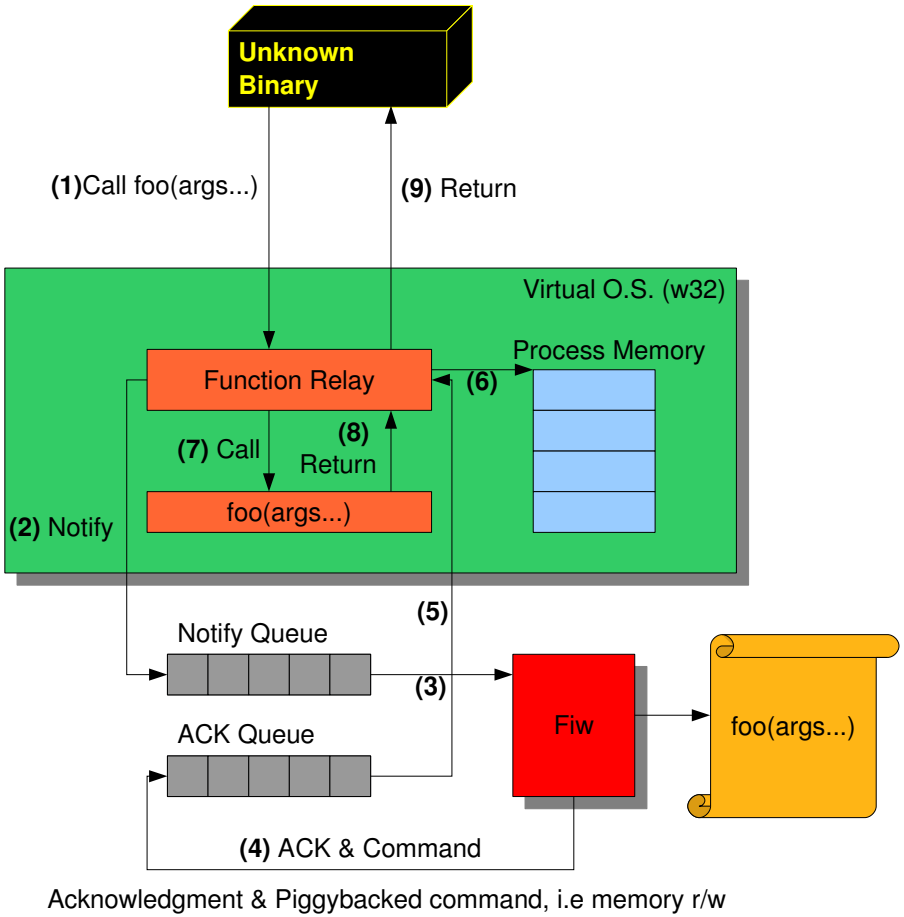
- Traditional debugging actions:
  - Step by by execution.
  - Memory layout inspection.
  - Replay of a debug session.
  - Memory access facility.
- Environmental changes:
  - File system changes.
  - Registry modifications.
  - Process memory alternation.
  - Crafting network messages.
  - Automated debug actions & plugins.
- The tool fiw was implemented in *C* and runs on linux operating system and has 6409 lines of code.

Fiw - the debugger
Architecture & design



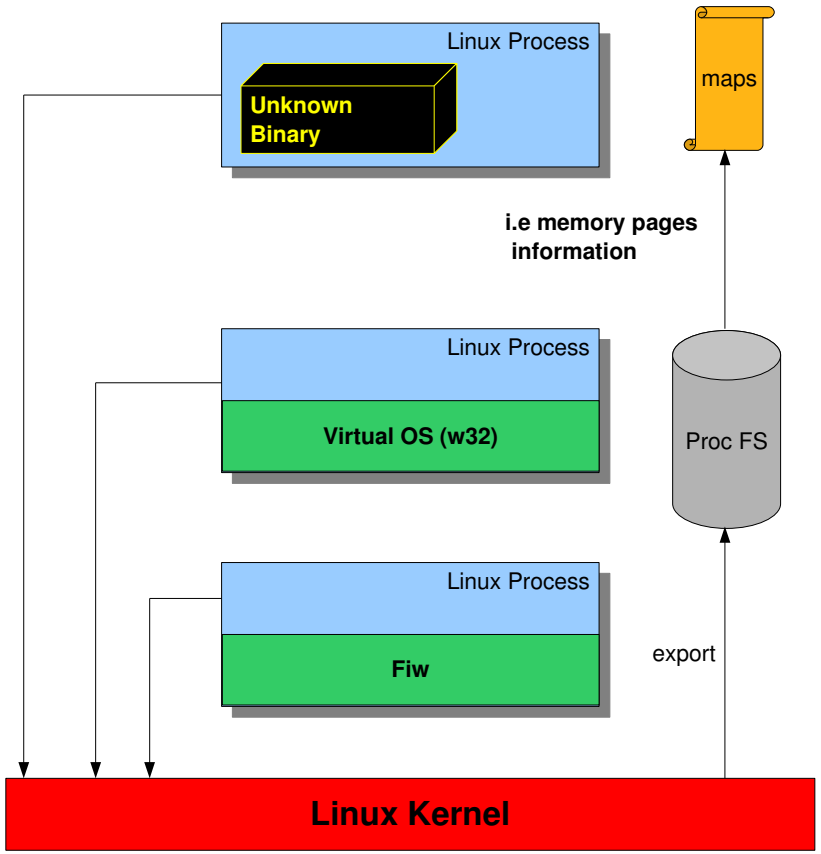
Fiw - the debugger
Architecture & design

Observing function calls & Memory Access.



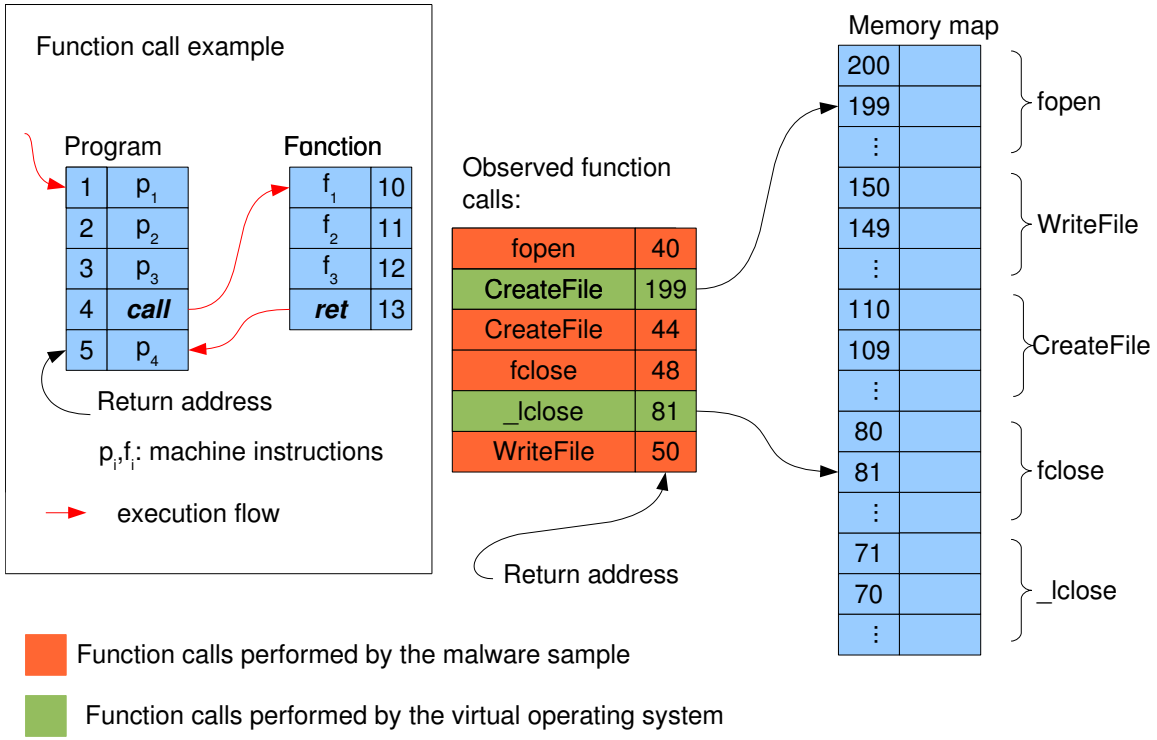
Fiw - the debugger
Architecture & design

Discovering memory layout.



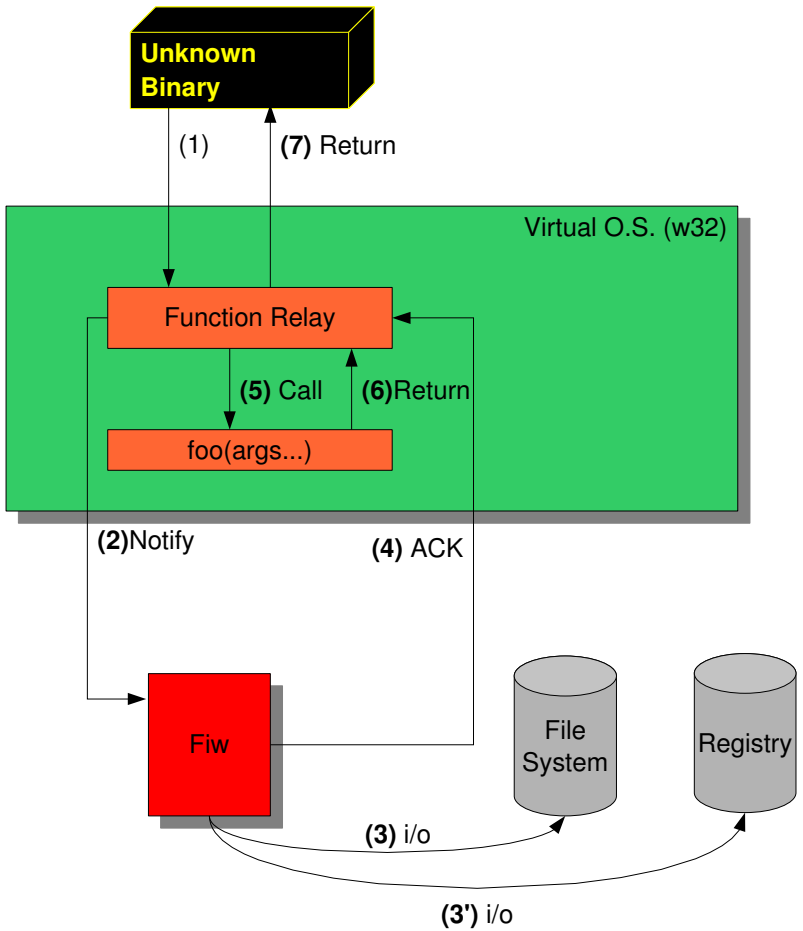
Fiw - the debugger
Architecture & design

Which function calls belong to the unknown binary?



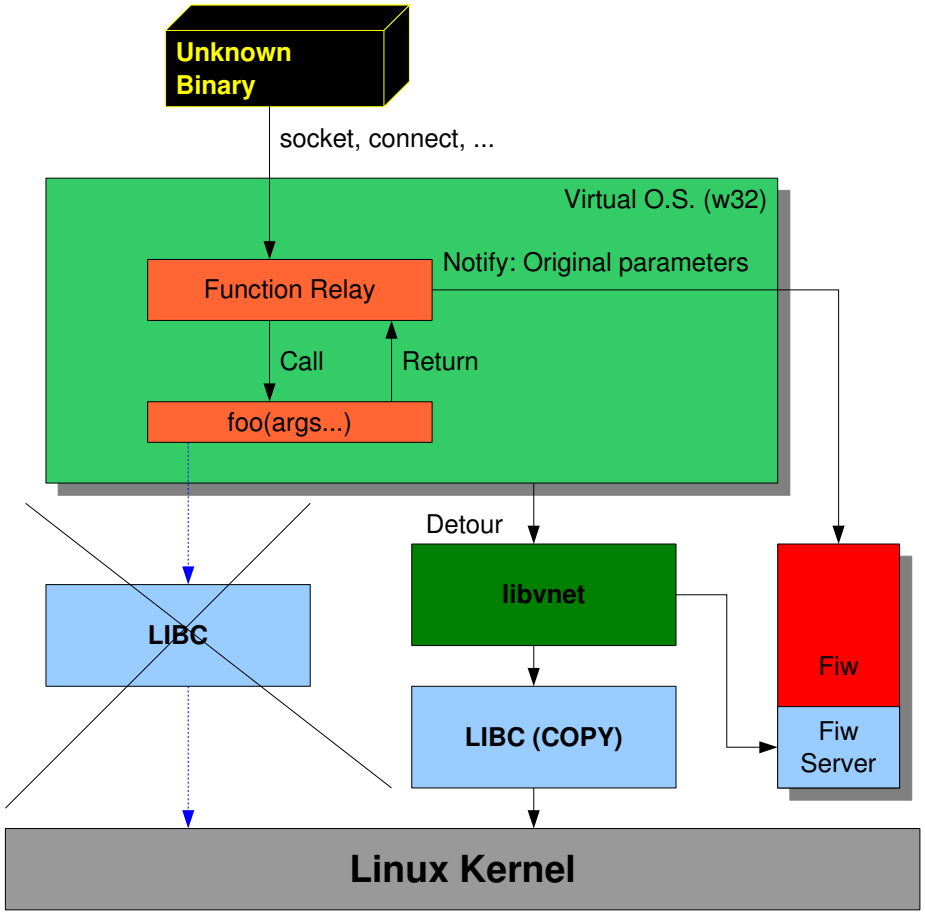
Fiw - the debugger
Architecture & design

Handling disk and registry access.



Fiw - the debugger
Architecture & design

Virtual networking.



Fiw - the debugger

- Replay of a debugging session:
  - Pseudo break points, function calls that do not match the break point are automatically acknowledged.
  - Stop auto acknowledgment if a given return address emerges.
  - Stop auto acknowledgment id a given function call emerges.
  - Replay with fiw script interface.

Fiw - the debugger
Architecture & design

- Automated debug actions:
  - Sometimes it is a burden to step through malware code.
  - i.e. Acknowledge long sequences of `GetProcAddress`, `AllocHeap` function calls.
- Plugins
  - The tool fiw puts debugging information in environment variables.
  - This information can be accessed via script / programs that can be launched from the tool fiw.
  - Results can be carried back via exit code of the script / programs.

Malware analysis scenarios

Discovering anti - debugging techniques

- Dealing with packing techniques
  - Let the piece of malware unpack itself and dump code from memory.
- Anti - debugging techniques
  - Detection of *soft-ice*, *regmon*, *filemon*, ... via `Createfile`, `EnumDeviceDrivers`, `IsDebuggerPresent`[1]
  - Code integrity checks.
  - Debugger traps (int 3 or 0xCC).
  - Registry key look-ups to related debuggers.
- $\longrightarrow$ these techniques do not work with the debugger fiw.

[1] Windows API functions

Malware analysis scenarios

Discovering anti - debugging techniques

Example (SdBot Analysis)

WORM/SdBot.506880.3, antivirname

First seen: 08.10.2007 02:58:46

Collected by nepenthes.csrrt.org

Note: Does not like debuggers :-)

Malware analysis scenarios

Discovering anti - debugging or anti - analysis techniques

Fiw - Command Line Interface

```
fiw>start sdbot_506880_3.exe
fiw>CreateFileA(00568cff, "\\\\.\\SICE", c0000000,3,0,3,80)
pid:  23325 tid:  0009 ret:  00568f1f
fiw>IsDebuggerPresent() pid:  23325 tid:  0009 ret:
0059b1d9
fiw>RegOpenKeyA(80000002, 0059f306, SOFTWARE\\NuMega\\
DriverStudio") pid:  23325 tid:  0009 ret:  0059f49e
FindWindowA(0059be9f, FilemonClass") pid:  23325 tid:
0009 ret:  0059bf6f
```

Malware analysis scenarios

Malware code obfuscation

- Goal: Make reverse engineering difficult as much as possible
- Techniques:
  - Use of dead code.
  - Use of not needed unconditional jumps.
  - Use of garabage instructions ...
  - Frameworks for these purpose exits!

Analysis of *Sdbot 506880 3*

Fiw - Command Line Interface

```
fiw>OutputDebugStringA(00577f31 "Themida Professional (c)
pid:  23721 tid:  0009 ret:  0057803b
fiw>
```

Malware analysis scenarios

Discovering termination cause

- Study cause of termination of execution.
- Let the tool fiw step trough ...

Analysis of *Sdbot 506880 3*

Fiw - Command Line Interface

```
fiw>start sdbot_506880_3.exe
fiw>auto ack
fiw>cont
[A] GetEnvironmentVariableA(00542fbd, LNumDLLsProt",
00542fcd) pid:  26887 tid:  0009 ret:  005437de
[A] TerminateProcess(ffffffff, 00000000) pid:  26887 tid:
0009 ret:  0054542f
```

2nd round adjust conditions!

Malware analysis scenarios

Discovering termination cause

- Set pseudo break point on an address or function name.
- Launch the analysis again.

Analysis of *Sdbot 506880 3*

Fiw - Command Line Interface

```
fiw>start sdbot_506880_3.exe
fiw>break ret 005437de
fiw>GetEnvironmentVariableA(00542fbd, LNumDLLsProt",
00542fcd) pid:  27206 tid:  0009 ret:  005437de
fiw>
```

Malware analysis scenarios

Discovering packing techniques

- Have a look at the memory layout.
- Observe return address of function calls.

Analysis of *Sdbot 506880 3*

Fiw - Command Line Interface

```
fiw>!./info_process_all
00400000-00401000 r-xp 00000000 62:00 456973
SdBot_506880_3.exe
00401000-0041e000 rwxp 00001000 62:00 456973
SdBot_506880_3.exe
fiw>cont
fiw>GlobalAddAtomW(6084c926 L"ux_theme") pid:  28009 tid:
0009 ret:  6083b9ad
```

0x6083b9ad $\notin$[0x00400000 - 0x0041e000].

Malware analysis scenarios

Communicating with the malware sample

- A piece of malware sometimes communicates with other entities.
- A piece malware that includes a backdoor accepts commands.
- It would be nice to discover the commands.
- The tool fiw has a command `vnet`
- In that case the messages send by a malware sample are sent to the tool fiw.

Communicating with the malware sample

Worm/Rbot.102912.13 analysis:

Fiw - Command Line Interface

```
fiw>vnet on
fiw>break name connect
fiw>start rbot_102912_13.exe
fiw>connect(58, 33f478, 10) pid:  28309 tid:  0009 ret:
0040e3a4
fiw>cont
fiw>send(00000058, 0033f220, 0000002e, 00000000) pid:
28309 tid:  0009 ret:  0040e4d1
fiw>cont
fiw>vnet tcp client list
*** Client socket list ***
Socket:  6
```

Malware analysis scenarios

Communicating with the malware sample

Fiw - Command Line Interface

```
fiw>vnet tcp recv 6 46
*** Reveived 46 bytes ***
0x4e 0x49 0x43 0x4b 0x20 0x55 0x53 0x41 0x7c 0x31 0x33
0x36 0x35 0x38 0x31 0xd 0xa 0x55 0x53 0x45 0x52 0x20 0x77
0x65 0x68 0x65 0x79 0x63 0x20 0x30 0x20 0x30 0x20 0x3a
0x55 0x53 0x41 0x7c 0x31 0x33 0x36 0x35 0x38 0x31 0xd 0xa
fiw>cont
```

- Malware often sends binary data
- This time: The string above is in ascii *NICK USA—136581 USER weheyc 0 0 :USA—136581*.

Discovering malware communication protocol

Disassembling the memory in order to discover conditions.

Fiw - Command Line Interface

```
fiw>recv(00000058, 0033d6a0, 00001000, 00000000) pid:
28309 tid:  0009 ret:  0040e519
fiw>dasm 0040e519 128
*** Disassemble address 40e519 size 128
 0040E519  85C0            test eax,eax
 0040E51B  7ECD            jng 0x40e4ea
 0040E51D  8D85E0F3FFFF    lea eax,[ebp-0xc20]
 0040E523  50              push eax
 0040E524  8D85E0E3FFFF    lea eax,[ebp-0x1c20]
 0040E52A  50              push eax
 0040E52B  E81CD7FFFF      call dword 0xffffd71c
fiw>
```

Experimental results

Table: General information about the malware set

Number of malware	104
Observation period	2005-2007
Average file size	135KB
Worms	34%
Mean detection rate	57.21%
Antivir detection rate	69.23%
Clamav detection rate	35.58%
Fprot detection rate	57.69%
Norman detection rate	66.35%

Experimental results

- Explore main execution path.
- Relaunch execution and check if another execution path is there.

Table: Control Flow division

Function calls that influence the CFG	#of malware
String compare functions	41%
Query of a registry value	60%
Query current date	41%
Check for internet connectivity	43%
Mutex	64%

Conclusion

- We presented a debugger for analyzing malware.
- The tool fiw is tightly bound to a virtual operating system w32.
- System function calls need to be acknowledged by the tool fiw before their execution.
- The tool fiw can inspect the memory of a malware sample
- The tool fiw can change the file-system and the registry in order to make a function call successful.
- The tool fiw can communicate with a malware sample via a virtual network.
- The tool fiw can do some automated debug actions via plugins.
- We showed various malware analysis scenarios.

Demo

- Demo
- Questions & Answers
- Acknowledgments: Hereby I want to thank the CSRRT-LU team for having given me access to their malware database for doing the experiments.

Future work

- Use a native windows OS as virtual operating system.
- Make the tool fiw more user-friendly.
- Add more debugging facilities.
- Integrate the tool in the malware analysis framework A.N.N.E. (Automated Analysis and Network Emulation, [ANNE,2006]).
  - Goal of the A.N.N.E. framework: Do malware analysis in automated way.
  - A server accepts malware samples and analyzes them with custom defined plugins.
- Improve replay debugging actions.
- Correlate observations performed by the tool fiw.
- Correlate observation performed by the tool fiw with network information.

Thank you for your attention!

Contact: haegardev@gmail.com

Bibliography


John Aycock.
Computer Viruses and Malware. Springer, 2006.


Gérard Wagener and Radu State and Alexandre Dulaunoy.
Malware Behaviour Analysis. In proceedings of the the 2nd International Workshop on the Theory of Computer Viruses 2007.


Andreas Moser and Christopher Krügel and Engin Kirda.
Exploring Multiple Execution Paths for Malware Analysis.. IEEE Symposium on Security and Privacy, 231-245, 2007.


Gérard Wagener and Alexandre Dulaunoy and Thomas Engel.
Automated aNalysis and Network Emulation.
http://www.csrrt.org.lu/wiki/index.php/Capturing_and_analyzing_Malware, 2006.