Advanced Topics in Malware Analysis

Dynamic Malware Analysis Tools and Techniques

Brendan Saltaformaggio, PhD
Assistant Professor
School of Electrical and Computer Engineering

Advantages and Disadvantages of Dynamic Malware Analysis

1

Learning Objectives

- Integrate various types of dynamic malware analysis tools for analysis
- Identify data transmitted/ received by applications
- Employ basic techniques of malware sandboxing
- Use debuggers to analyze malware executable at runtime

2

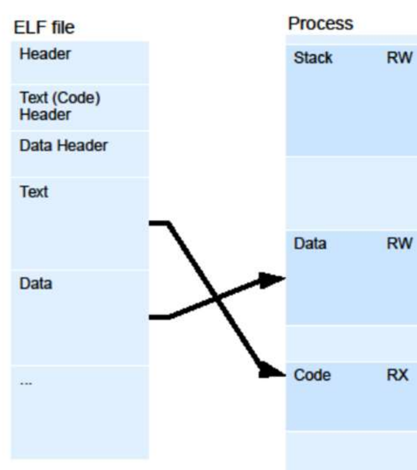
Welcome to Dynamic Analysis!

- Dynamic program analysis solves problems by inspecting software execution
- Inspecting executions vs. static binaries
  - Not all statements are executed, but one statement may be executed many times
  - Analysis is always local to a single path --- the executed path
  - All variables are instantiated
    - No aliasing! Yay!
    - Memory addresses are concrete
- This results in dynamic analysis having ….
  - A relatively lower learning curve
  - Better precision
  - More widely applicable analyses
  - Better scalability (i.e., no more path explosion!)

3

A Look Inside of an Executing Process

- Loader maps runtime sections of all shared objects into virtual address space
- The loader calls global initialization functions of all libraries
  - Why is the order important?
  - E.g., Libc initializes heap data structures and uses the sbrk and brk system calls to set up space for the memory allocator
- In Summary: Executing binaries look nothing like binary files!
  - But it is possible to go back from memory to a file! We will see it…



4

Dynamic Analysis of Malware

- Execute and monitor malware sample
- Be careful…
- **Advantages**
  - Insight into behavior w/o deep understanding of code
  - Can overcome encryption/packing schemes
- **Disadvantages**
  - Can & will accidentally execute dangerous payloads!
  - You may have to overcome anti-analysis behaviors
  - Some tools may help you, but…
  - Malware detect tools, virtual machines, debuggers, anything…

Georgia Tech

5

Advanced Topics in Malware Analysis

Dynamic Malware Analysis Tools and Techniques

Brendan Saltaformaggio, PhD

Assistant Professor

School of Electrical and Computer Engineering

Basic Tools

Georgia Tech

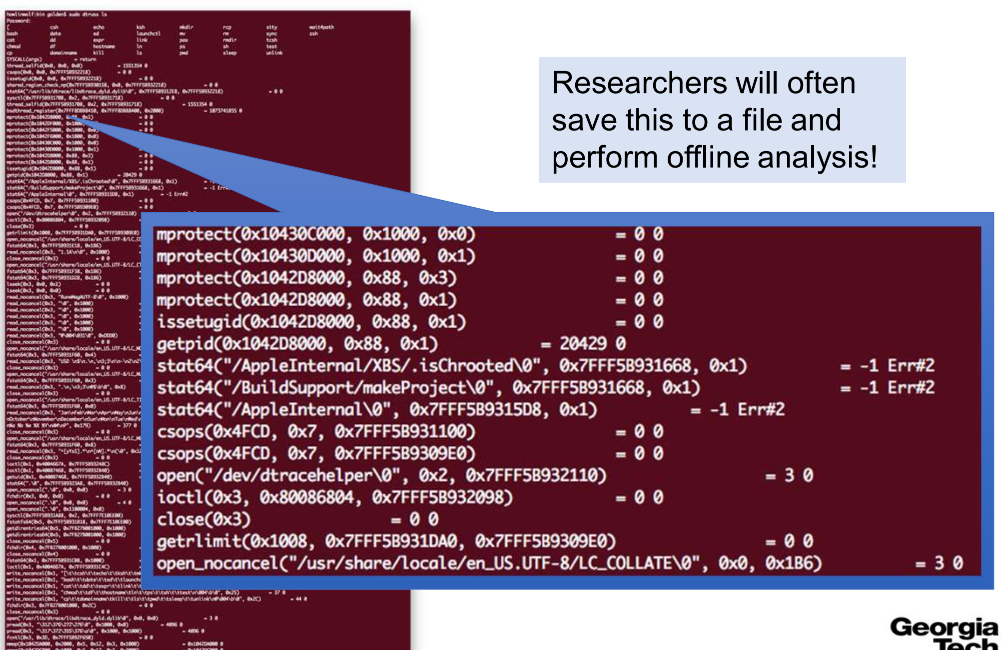
6

Basic Dynamic Analysis Tools

- Process/Filesystem/Registry analysis
  - Live analysis of modification of Windows registry
  - Filesystem activity monitoring
    - Basic: $ watch ls –l
    - Periodically execute a command & show the changing output
  - System call traces
    - dtruss, ptrace, etc.
  - Library call traces
    - ltrace
  - Process monitoring
    - Best Ever: procmon

7

dtruss

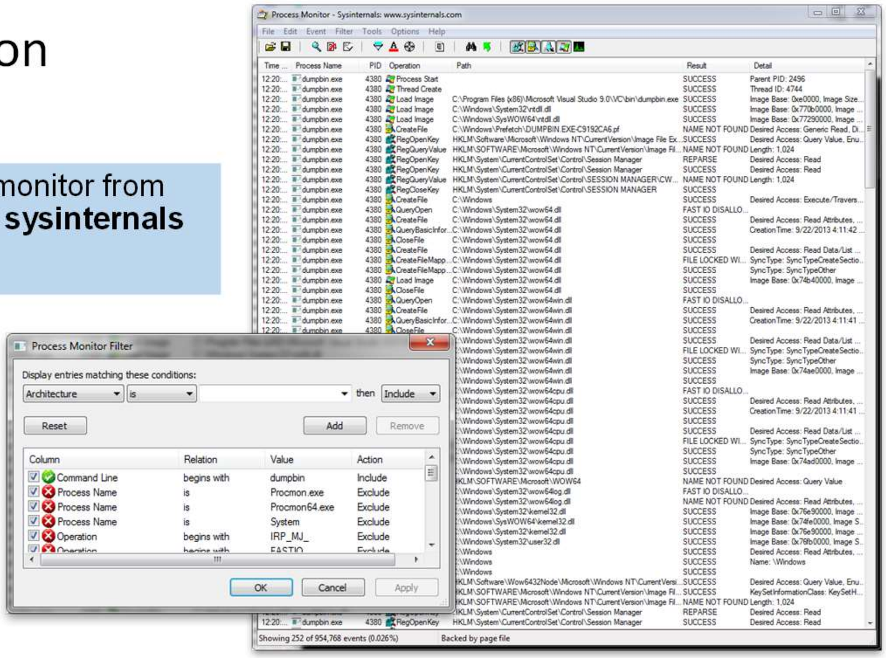


Researchers will often save this to a file and perform offline analysis!

8

Procmon

Process monitor from Windows **sysinternals** tool suite



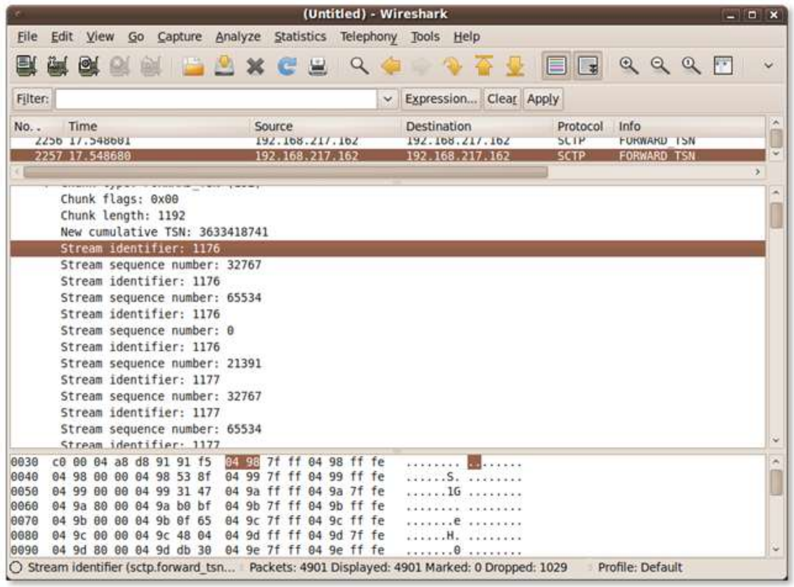
9

More BASIC Dynamic Analysis Tools

- **Network Trace Capture/Analysis**
  - Discover data transmitted/received by application
    - e.g., WireShark
  - Discover remote peers
  - Allows reverse engineering of network protocols used by an application
    - A lot of research done on this!
    - **Highly Recommended Read:** Comparetti, P. M., Wondracek, G., Kruegel, C., & Kirda, E. (2009). Prospex: Protocol Specification Extraction. *2009 30th IEEE Symposium on Security and Privacy*.
- **Serial/Parallel Port Monitoring**
  - Reverse engineer serial/ parallel protocols
  - E.g., portmon
  - Not just old malware!
  - Newer cyber-physical (“close to the hardware”) malware

10

Wireshark



Georgia Tech

11

Major Advantage of Dynamic Malware Analysis

- Analysis of **encrypted**/**packed**/**obfuscated** malware
- Obfuscation or encryption of parts of the malware body **makes static analysis much harder** or impossible!
  - This buys the malware author time to keep the infection & spread going!
- Reverse engineers will often have to supplement static analysis with **dynamic analysis** or **emulation techniques**
- There are some basic techniques
  - Debuggers
  - IDA support
  - Extracting decrypted/unpacked executable code from memory
  - We will see next…

Georgia Tech

12

Advanced Topics in Malware Analysis

Dynamic Malware Analysis Tools and Techniques

Brendan Saltaformaggio, PhD

Assistant Professor

School of Electrical and Computer Engineering

Debuggers

Georgia Tech

13

The Dynamic Analysis Fan Favorite: Debuggers

- Not just for fixing your buggy C pointers anymore!
- Reverse engineers can analyze the malware executable at runtime
- Provides a disassembly of executable
  - Like static disassembly but local to the execution path!
- Also allows reverse engineers to monitor and **modify** execution
  - Modify = change the "call kill_your_machine" instruction to "nop"
- Monitor CPU registers, memory, flags, and control flow
- Trace access to specific memory regions
- Set breakpoints to avoid extremely fine-grained analysis

Georgia Tech

14

Kernel vs. User-mode Debugging

- **User-mode debuggers** analyze only user-mode side of application
- Examples: OllyDbg, debuggers in IDA Pro, GDB & WinDbg (can be both!)
- Often sufficient for most malware

- God Mode Activate: **Kernel-mode debuggers**
  - Often used by kernel developers
- Examples: SoftICE (gone) , KD/NTKD, GDB & WinDbg (can be both!)
- Allow analysis of user-level code, kernel code, & low-level breakpoints

Georgia Tech

15

More on Kernel-Mode Debuggers

- In the old days, required a second system connected via serial cable for inspecting kernel instructions & data

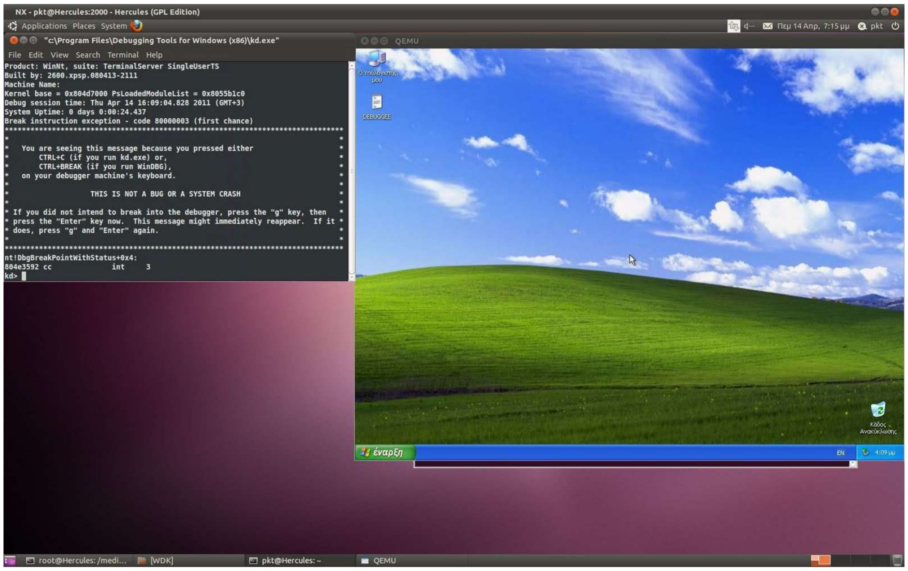
- Nowadays we can simply attach the debugger to a guest VM from the host

- Example from malware analysis:
  - Low level breakpoint on kernel code that provides window movement
  - This allows window movement action to be located regardless of which high-level API is used

Georgia Tech

16

Debugging Windows Kernel in a VM



17

SoftICE Screenshot (Rest in Peace)

- Tricking Skype to run while being debugged by SoftICE
- Ref: http://gcasiez.pagesperso-orange.fr/skypeandsoftice.html
- Skype has many anti-debugging tricks, maybe it is a malware?
- We will see anti-debugging tricks later…

18

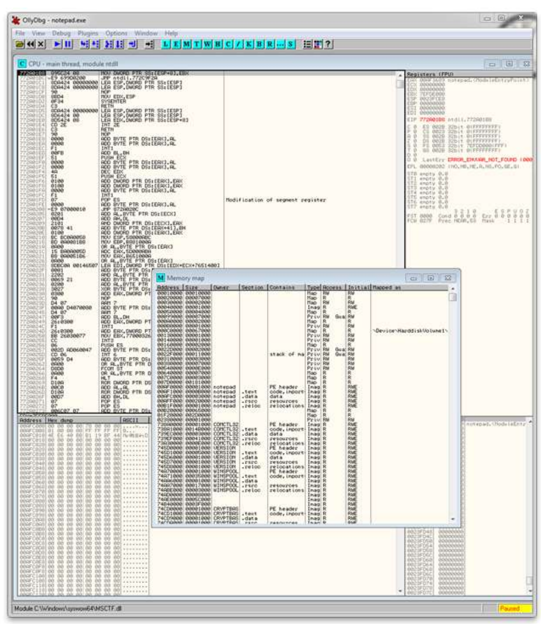
The Big Kahuna: GDB

- GDB is the default debugger on Linux machines and part of the GNU toolchain
- `disas` symbol to disassemble a symbol
- `break *0xf00` to set a break point (sets breakpoints anywhere)
- `bt` prints a backtrace of the stack (either frame pointers or debug information is needed)
- `info registers` displays current register contents
- `set` can update memory cells, variables, even change code
- Infinitely extensible via plug-ins!

19

OLLYDBG

- The fan favorite of user-mode debuggers
- So many features… ☺
- Decent GUI ☺
- Becoming totally outdated ☹



20

Ollydbg Basics

- User-mode debugger
- **Typical functionality**:
  - Single stepping, step into, step over, etc.
  - Examination of memory, registers
- Supports several kinds of breakpoints
  - **Software** (unlimited # via int 3, writes "0xCC" into code)
    - Easy for malware to defeat
  - **Hardware** (only four at a time, CPU watched for code addresses stored in registers)
    - Harder to defeat
  - **Memory** (one and only one at a time)
    - OllyDbg single steps and catches access to memory location!
    - Less likely to be defeated

21

Ollydbg Basics (Cont.)

- For some packed/encrypted executables, you can isolate decryptor, execute it, and then analyze unpacked executable
- Careful not to execute the viral payload!

22

Debugging Malware, Easy?

- No
- Reverse engineering modern malware is never easy
- Many layers of obscured code, anti-debugger, anti-emulation, anti-VM
  - Malware detects use of software/hardware breakpoints
  - Use of rdtsc
  - Side effects of debuggers
  - APIs for debugger detection
  - See http://www.securityfocus.com/infocus/1893 for lots more on Windows debugger detection
- Recommended read: Chapter 17 in Volume 3B of the Intel manuals for detailed information on debugging
- Often requires using multiple tools at the same time to get a high-level idea of what is going on BEFORE doing fine-grained analysis

Georgia Tech

23

Advanced Topics in Malware Analysis

Dynamic Malware Analysis Tools and Techniques

Brendan Saltaformaggio, PhD

Assistant Professor

School of Electrical and Computer Engineering

Real World Malware Investigation

Georgia Tech

24

Real World Malware Investigation

Unlocking Ransomware Without Paying The Russians

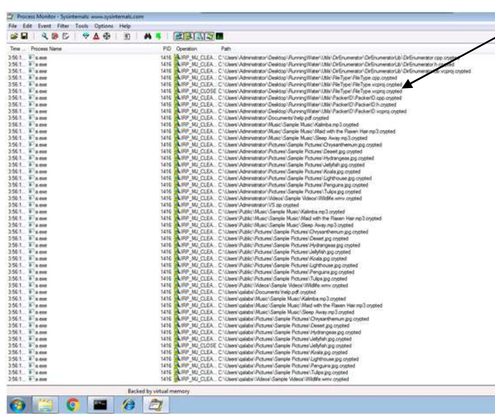
Georgia Tech

25

Ransomware Case Study

Step 1: Fire up a VM and fill it with "realistic" user files

Step 2: Detect RansomLock.AK with Procmon



- Victim files are given a ".crypted" extension
- File extensions and target files change among different families
- E.g., This family does encrypt source code (c, cpp, h files), but others do not
- No families encrypt executables --- They want the system to keep running!

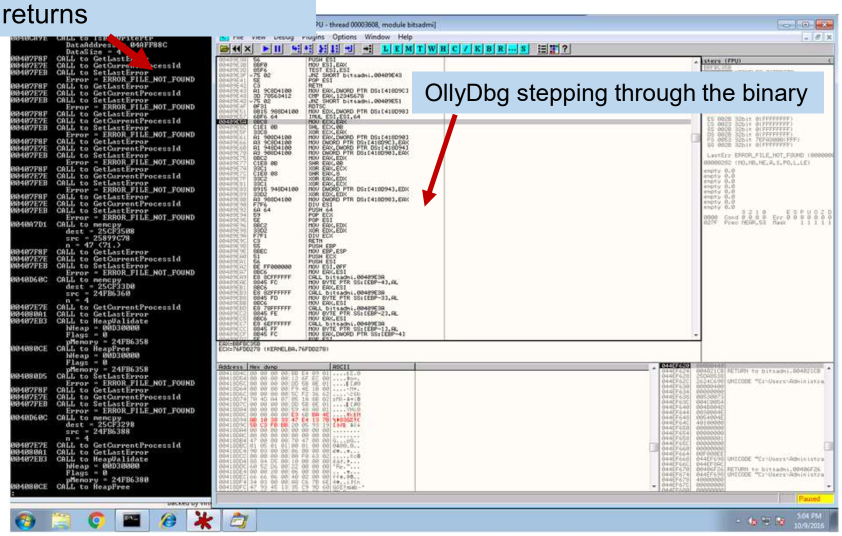
Georgia Tech

26

Finding The Decryption Loop in Cryptolocker.AH

Terminal logging library function calls & returns

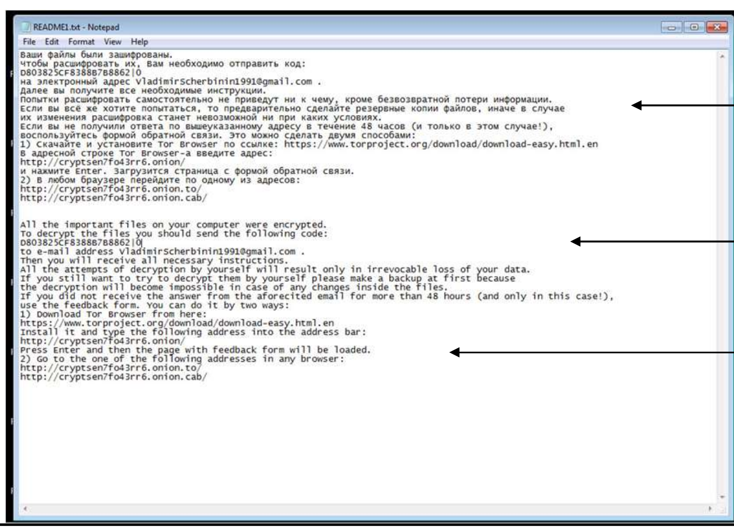
OllyDbg stepping through the binary



- Uses "baked-in" inline statically-compiled RSA encryption/decryption loop
  - Finding it was not fun ;)

27

I had to Reverse engineer it…
The readme.txt was not helpful!



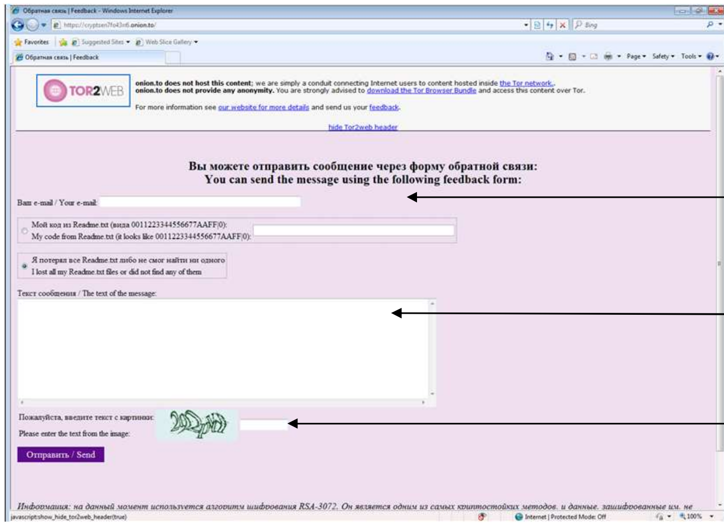
Russian?

Please email the following ID to vladimirscherbinin1991@gmail.com
(perhaps 25 years old?)

If you do not hear back in 48 hours then please use the Feedback Form!

28

But The Feedback Form Was Nice!



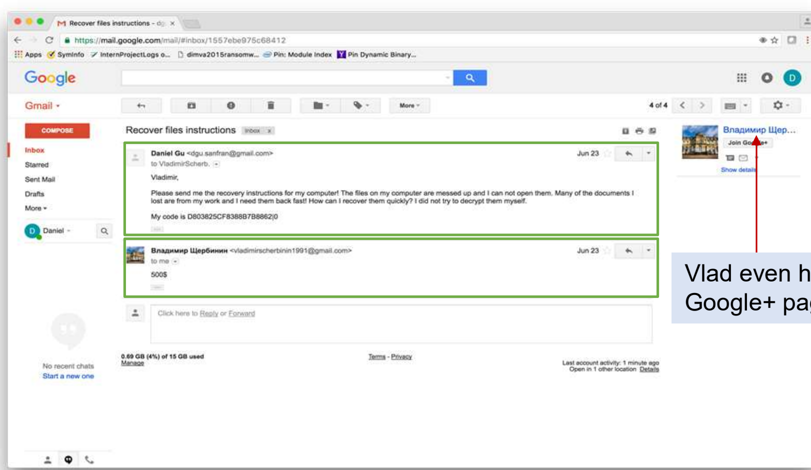
Enter your email

Personal message

And the CAPTCHA
(Vladimir doesn't
want spam!)

29

I had to Reverse engineer it…
I wasn't paying $500 to get Symantec's files back!



Vlad even had a Google+ page!

30

Advanced Topics in Malware Analysis

Dynamic Malware Analysis Tools and Techniques

Brendan Saltaformaggio, PhD

Assistant Professor

School of Electrical and Computer Engineering

Software/Hardware Breakpoints

Georgia Tech

31

Software Breakpoints

- Software breakpoints use **int 3**
- **int 3** opcode is a one byte instruction specifically to support easy replacement of other instructions by debugger
- Debugger replaces the target instruction with **int 3** opcode (0xCC)
- Executing **int 3** directly calls a system call which looks up if the process is being debugged
- Debugger regains control through **int 3** handler

Georgia Tech

32

Detection of Software Breakpoints

- Malware simply search their own code for the **0xCC opcode**
- Malware can also checksum/hash its own code to make sure **int 3** has not been inserted
- Malware can remove breakpoint, exit, etc.
- Obvious sign for reverse engineers: Debugging stops working/ exits/ crashes, whatever…

Georgia Tech

33

Detection of Hardware Breakpoints

- Debug registers **DR0**, **DR1**, **DR2**, **DR3** contain linear addresses for breakpoints
- Debug register **DR7** is debug status register
- Indicates which breakpoints **DR0 - DR3** are set and has other status info
- When the CPU detects **RIP == DR(0-3)**, then it raises an interrupt
- Does the same stuff as **int 3** handler
- Again: Chapter 17 in Volume 3B of the Intel manuals for detailed information on debugging

- Malware can scan debug registers to determine if breakpoints are set
- If set: can exit, perform alternate behavior, etc.

Georgia Tech

34

Use of rdtsc to Detect Debugging

- `rdtsc` (opcodes 0x0F 0x31) introduced for Pentium and beyond
- Read Time Stamp Counter
- Gets # of ticks since last CPU reset into rdx:rax
- However, this can reveal dynamic analysis to a malware!
- **Idea**: Debugged/monitored code runs slightly more slowly than un-debugged code
- Malware measures the passage of time using `rdtsc` and executes evasive action if its execution seems to be taking too long
- Try for yourself: set breakpoints on NOPs after `rdtsc` instructions

Georgia Tech

35

Detecting Side Effects of Debuggers

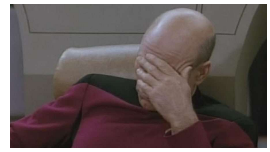
- Example: Detect **SoftICE** debugging your kernel from inside a malware!
- **SoftICE** replaces a few interrupt handlers in the kernel's **interrupt descriptor table** (IDT)
  - Namely, **int 1** and **int 3** in order to handle debugging the kernel itself!

```
mov eax, dword ptr [pIDT+2] ; eax -> IDT
add eax, 8                  ; eax -> int 1 vector
mov ebx, [eax]              ; ebx == int 1 vector
add eax, 16                 ; eax -> int 3 vector
mov eax, [eax]              ; eax == int 3 vector
and eax, 0ffffh             ; strip the selector
and ebx, 0ffffh             ; part of it
sub eax, ebx                ; find displacement
cmp eax, 10h
jne HackedVector            ; if it isn't equal, then chances are
                            ; SoftICE had tampered with these vectors
```

Georgia Tech

36

Windows Leaks for Debugger Detection



- Windows provides ways to if you're being debugged
- Offset 2 in PEB (**Process Environment Block**) structure is a flag for "**is debugger present**"
- Windows provides an API called "**IsDebuggerPresent()**" which simply checks this flag
- **PEB!NtGlobalFlags** determines how some heap management is performed and the flags get changed for debugged processes

- Many others, some are very obscure
- More info https://www.symantec.com/connect/articles/windows-anti-debug-reference

37

Advanced Topics in Malware Analysis

Dynamic Malware Analysis Tools and Techniques

Brendan Saltaformaggio, PhD
Assistant Professor
School of Electrical and Computer Engineering

Case Study: Armadilo

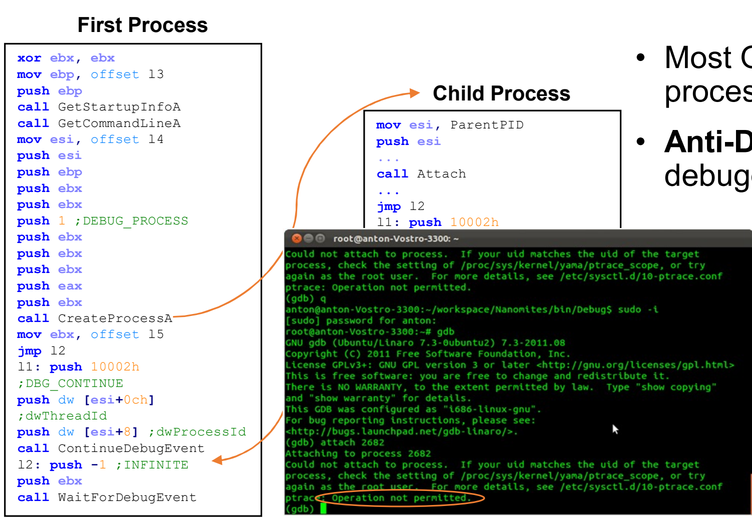
38

Brief Packer Case Study: Armadillo

- Malware (or their packers) may also include very complex custom techniques to thwart debuggers
- Case Study: Armadillo
- Commercial packer from Silicon Realms
  - They call it an "**executable compressor**"
- Widely used to protect expensive commercial software from being reverse engineered
- Three amazing features:
  - Double process debugging
  - "Magic" jumps (conditional branches instead of JMPs)
  - Encryption

39

Double process debugging

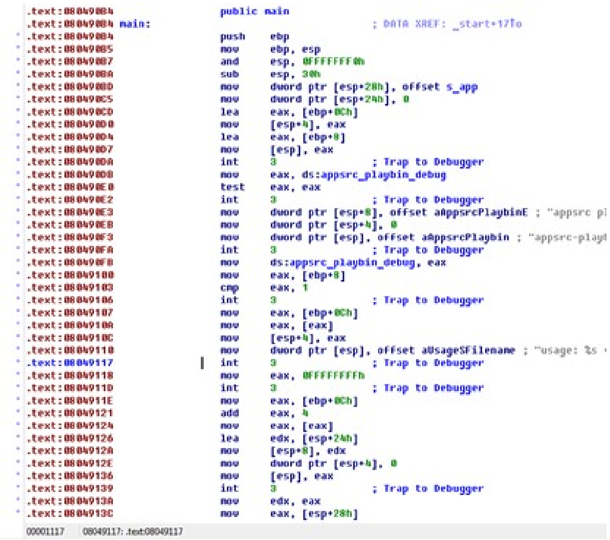


- Most OSs only allow 1 debugger process to debug 1 debugee process
- **Anti-Debug Solution**: 2 processes debugging each other!

40

Armadillo's Magic

- "**Nanomites**": Twist on double process debugging
  - Jump instructions replaced with int 3 software breakpoint opcodes in child process
  - Parent handles software breakpoints and looks up actual jump targets in an encrypted table
  - Decrypts the jump target & patches child code before resuming the child's execution



Georgia Tech

41

Defeating Nanomites

- Reverse engineers need to repeatedly & carefully terminate the child/parent debug relationship after monitoring behavior
  - Have to keep doing this until all needed nanomite patches have been detected!
- Manual unpacking very tedious
- Recommended read: https://www.apriorit.com/white-papers/293-nanomite-technology
- Constant war between crackers and Silicon Realms
- Evidence that all versions have been cracked, eventually
- Excellent additional read:
  - Ferrie, P. (2008). Anti-unpacker tricks- part one. *Virus Bulletin*.

Georgia Tech

42

Defeating Anti-Debugger Techniques

- To combat anti-debugger stuff, reverse engineers will generally have to patch the malware binary to kill one or more of:
  - Double process debugging
  - Debugger detection leaks
  - Breakpoint tampering
  - Use of rdtsc or other timing APIs
  - Illegal instructions

But… when an interpreter is used by the malware, you will probably have to keep it intact unless you want to reverse engineer the interpreter too ☹

Georgia Tech

43

Defeating Anti-Debugger Techniques (Cont.)

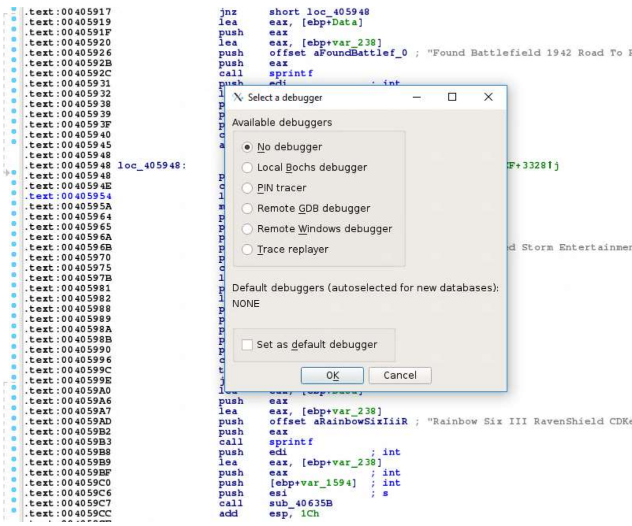
- Difficulty varies by packer
- Different versions of packer will likely require some different techniques
- Recommended Read:
  - Deng, Z., Zhang, X., & Xu, D. (2013). Spider. *Proceedings of the 29th Annual Computer Security Applications Conference on - ACSAC 13*

Georgia Tech

44

Debuggers in IDA

- IDA knows that you will be performing static and dynamic analysis
- So why not use the power of IDA to help with debugging…
- … and feed the knowledge gained from debugging back to IDA!
- So IDA integrates with many debuggers
- Plug-ins can add support for all others!

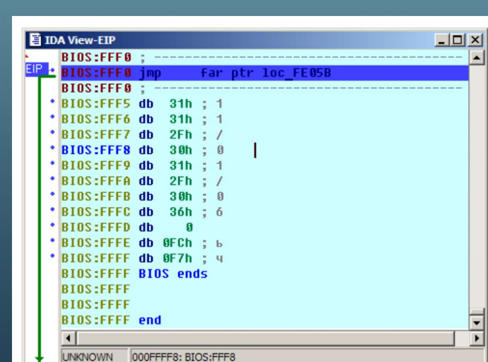


Georgia Tech

45

Debuggers in IDA (Cont.)

- IDA + Debuggers + Plugins = Extremely powerful
- https://www.hex-rays.com/products/ida/support/tutorials/remote-debugging.shtml
- IDA can connect to a remote debugger and supply all of that debugger's functionality within IDA
- Even remote debuggers on VMs (remember kernel debuggers?)
- Here is IDA debugging the BIOS of a Windows kernel during boot!

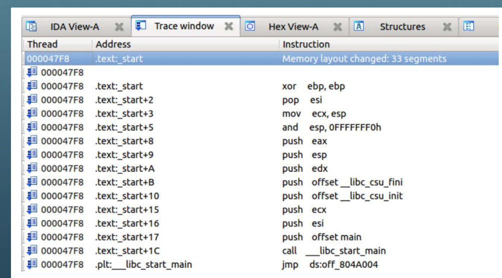


Georgia Tech

46

IDA Trace and Replay

- IDA can also record a trace of the instructions which execute during dynamic analysis
- Then replay those exact instructions over and over again!
- Allows for repeating analysis without having to re-execute the malware
  - E.g., Avoid miserable unpacking, Command and control server is gone, etc.
  - Research e.g., Differential analysis -> “What paths differ when I give this argument?”
- Much more on tracing in the next slide set…



Georgia Tech

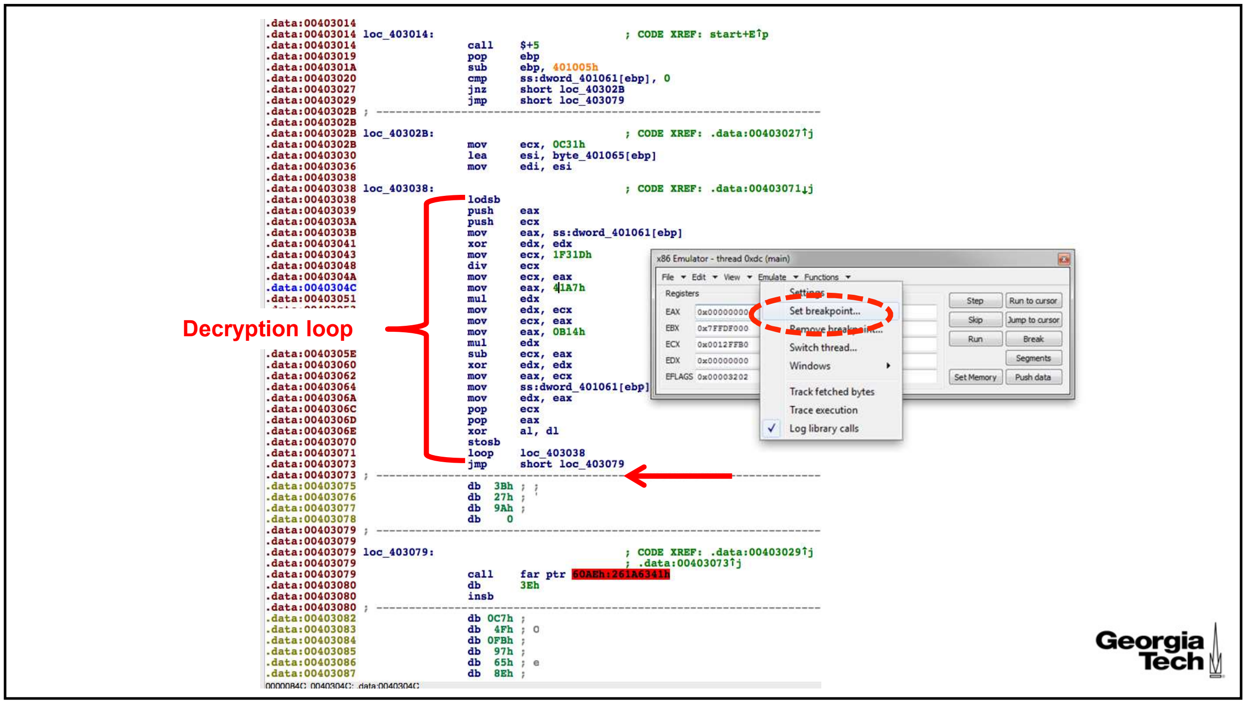
47

Instruction Emulation: x86emu

- Plug-in for IDA Pro written by Chris Eagle (author of The IDA Pro book!)
- Very IDA Pro SDK version specific, but has been updated!
- Available @ http://sourceforge.net/projects/ida-x86emu/
- Provides x86 emulation
- Safe, because it’s not really executing code that’s been loaded into IDA
- Memory/register/stack emulation
- Does not execute Win32 API functions
- Does allow you to specify their effects
- Often enough to let the decryptor in an encrypted/packed executable complete
- Automatically updates the IDA disassembly listing

Georgia Tech

48



49

Lesson Summary

- Integrate various types of dynamic malware analysis tools for analysis
- Identify data transmitted/received by applications
- Employed basic techniques of malware sandboxing
- Use debuggers to analyze malware executable at runtime

50