Alternative Medicine: The Malware Analyst's Blue Pill

Paul Royal

Damballa, Inc

Black Hat USA 2008
August 6th, 2008

DAMBALLA

Agenda

- **About**
- **Malware Analysis**
  - Approaches, Challenges
- **Malware Analysis Using Intel VT**
  - Virtual Machine Introspection
  - Fine- and Coarse- Grained Tracing
- **Design/Implemention**
  - Azure, a PoC Malware Analysis Tool
- **Experimentation/Evaluation**
  - Automated Unpacking
- **Conclusion/Future Work**
- **Q&A**

DAMBALLA

About

- **Damballa**
  - Botnet detection and remediation in large enterprise networks
- **Paul Royal**
  - Principal Researcher at Damballa
    - Focus on sandboxes, sensors and analyzers used for the discovery and identification of bot behavior
  - BS/MS CS from Georgia Tech
    - Studied automated malware processing and transformation

Malware Analysis

- **Static Analysis**
  - Attempts to understand what a program would do if executed
  - Requires: An unobstructed view of program code
- **Dynamic Analysis**
  - Attempts to understand what a program does when executed
  - Requires: Ability to trace the actions of the binary (with fine- or coarse- granularity)

DAMBALLA

Analysis Challenges

- **Dynamic Analysis**
  - Must handle anti-debugging, anti-instrumentation, anti-VM
- **Static Analysis**
  - Must overcome code obfuscations (e.g., packing)
  - Solutions transitively dependent on dynamic analysis

Dynamic Analysis Approaches

- **In-Guest**
  - Implemented using Win Debugging API, API hooking, Custom Handlers (e.g., pagefault, debug exception)
  - Examples: CWSandbox, Saffron, VAMPiRE
- **Whole-System Emulation**
  - Often created by modifying/extending existing system emulator (e.g., QEMU)
  - Examples: Anubis, Renovo
- **Often vulnerable to detection**

DAMBALLA

Detecting In-Guest Tools

- **CWSandbox**
  - Hooks WinAPI calls; does not hide hooks

```
#include <windows.h>
#include <stdio.h>

int main(int argc, char* argv[]){
            HMODULE kernel32 = NULL;
            void *createfile_function_pointer = NULL;
            unsigned char opcodes[2];

            kernel32 = LoadLibrary("kernel32");
            createfile_function_pointer =
                        (void*)GetProcAddress(kernel32, "CreateFileA");

            memcpy(opcodes, createfile_function_pointer, sizeof(opcodes));

             if(opcodes[0] == 0xFF && opcodes[1] == 0x25){
                        fopen("in_cwsandbox", "w");
                        exit(-1);
             }

 return 0;
}
```

Credit: Artem Dinaburg

DAMBALLA

Detecting System Emulators

- ## QEMU
  - Vulnerable to attacks that exploit inaccurate/ incomplete system emulation

```
#include <stdio.h>
#include <windows.h>

int seh handler(struct EXCEPTION RECORD *exception record,
                void *established frame,
                struct CONTEXT *context recorcd,
                void *dispatcher context){
 printf("Not QEMU\n");
 exit(0);
}

int main(int argc, char *argv[]){
 uint32 t handler = (uint32 t)seh handler;
 printf("Attempting detection\n");
 asm("movl %0, %%eax\n\t"
               "pushl %%eax\n\t"::"r" (handler): "%eax");
 asm("pushl %fs:0\n\t"
               "movl %esp, %fs:0\n\t");
 asm(".byte 0xf3,0xf3,0xf3,0xf3,0xf3,0xf3,"
               "0xf3,0xf3,0xf3,0xf3,0xf3,0xf3,"
               "0xf3,0xf3,0xf3,0x90");
 asm("movl %esp, %eax");
 asm("movl %eax, %fs:0");
 asm("addl $8, %esp");

 printf("QEMU Detected\n");
 return -1;
}
```

Credit: Peter Ferrie, Artem Dinaburg

DAMBALLA

An Alternative Approach

- **Current Approaches**
  - In-Guest
    - Always some instrumentation/side effect to detect
  - Whole-System Emulation
    - Always some inconsistency to exploit
  - Detection/Detection-Prevention Arms Race
- **Need external, baremetal-like platform for malware analysis**
  - What about using hardware virtualization extensions (e.g., Intel VT)?

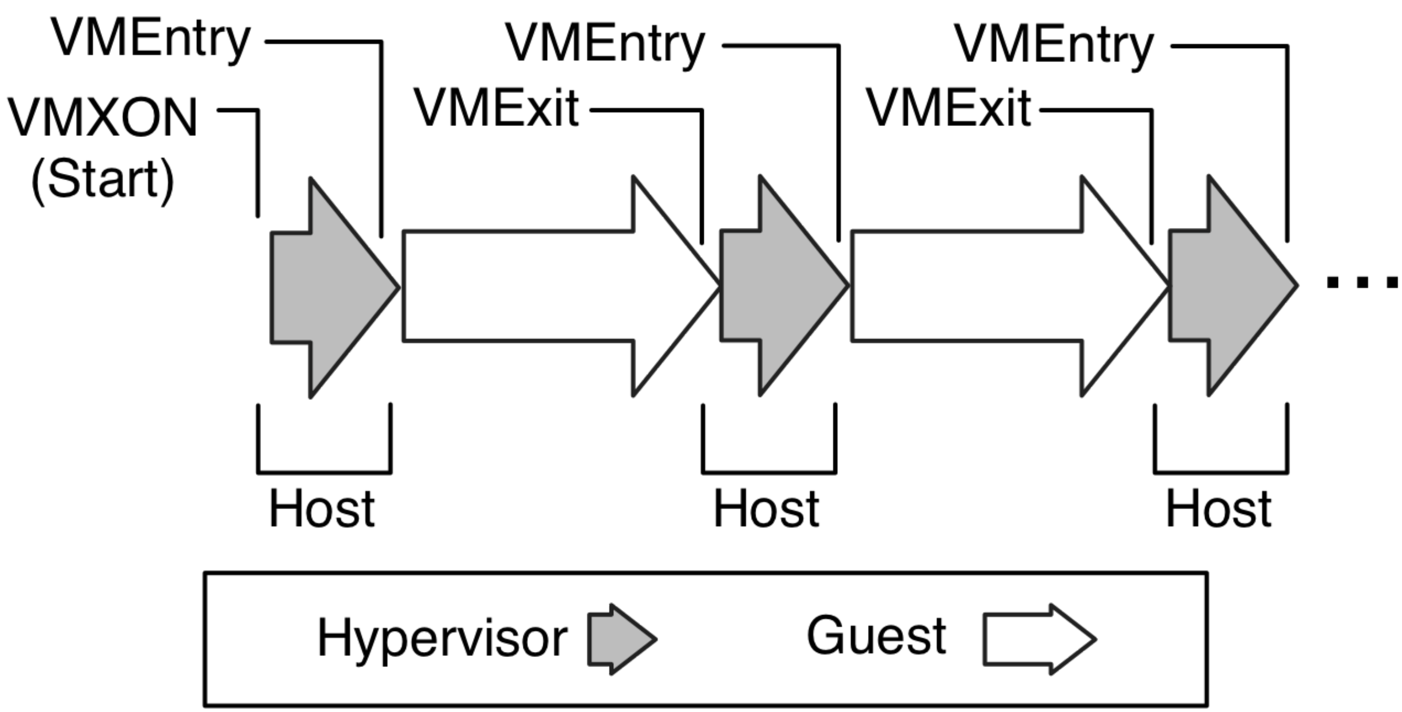
Intel VT

- **Hardware-assisted means to virtualize x86 instruction set**
- **Operation**
  - Hardware elements (e.g., VMCS)
  - Virtualization instructions (e.g., VMXON, VMLAUNCH, VMRESUME)
  - Administrative software component
    - Host can read from, write to, preempt receipt of notification for certain guest events
    - Preemption causes a VMExit (guest is frozen)
- **Allows for the execution of unmodified guests**

DAMBALLA

Intel VT Cont'd

- **Operation**



DAMBALLA

Intel VT for Malware Analysis

- **Positives**
  - External
    - No in-guest components to detect
  - Capable
    - Functionality suggests potential use in analysis
  - “Equivalent”
    - Hardware-assisted nature offers transparency
- **Negatives**
  - Not made for analyzing malware
    - Any functionality (e.g., coarse-grained tracing) must be derived
  - Intel VT/administrative software component vulnerable to detection

DAMBALLA

Discussion Preface

- **Next sets of slides discuss three malware analysis requirements**
  - Virtual Machine Introspection
  - Fine-Grained Tracing
  - Coarse-Grained Tracing
- **Format**
  - Requirement's description
  - x86 background
  - Leveraging Intel VT to fulfill requirement

Virtual Machine Introspection

- **Garfinkel and Rosenblum**
  - Inspecting a guest process externally for the purpose of analysis
- **Example use of VMI**
  - External identification of a target process in the guest
  - In malware analysis, target process must be identified after loading but before execution
- **VMI through Intel VT?**
  - Possible by leveraging host's MMU responsibilities

DAMBALLA

x86 Memory Management

- **Virtual Memory in x86**
  - Uses paging to provide processes with the appearance of an exclusive address space
  - Each process has its own page directory pointer
  - Page directory pointer of active process stored in CR3
- **Context Switches**
  - Process switched in or out by the OS
  - Page directory must be changed to the upcoming process
  - Change occurs as a MOV to CR3

VMI through Intel VT

- **Exploit host's MMU duties**
  - During guest context switch, guest attempts MOV to CR3
  - Causes VMExit; guest is frozen until resumed by host
- **Guest reads can be used to identify the upcoming process**
  - Requires a bit of reverse-engineering kernel data structures
  - More on this later

DAMBALLA

Fine-Grained Tracing

- **Monitoring the behavior of a process at the instruction-level**
- **In malware analysis, fine-grained is used for**
  - Dynamic taint analysis
    - Example: Panorama
  - Multi-path exploration
  - Precision automated unpacking
    - Examples: PolyUnpack, Renovo

DAMBALLA

x86 Debugging

- **FLAGS register**
  - Contains set of processor status, control, and system flags
  - Read from/written to using PUSHF/POPF
- **FLAGS: trap flag**
  - System flag use to enable “single-stepping” or debug mode
  - When set, a debug exception is thrown immediately after execution of the next instruction

Fine-Grained via Intel VT

- **Previous in-guest analysis tools have used the trap flag**
  - VAMPiRE
    - Installs its own debug exception handler
    - Repeatedly sets the trap flag and preempts the resulting exception
- **Intel VT can do the same externally**
  - Host sets the guest's trap flag in FLAGS
  - Host uses Intel VT to preempt receipt of the corresponding exception
    - No in-guest debug exception handler

DAMBALLA

Coarse-Grained Tracing

- **Monitoring the behavior of a process at the API or system call level**
  - Discrete events are often easily recognizable actions
    - Examples: File or registry access, process or thread creation, network activity
- **In malware analysis, use for**
  - Behavioral Antivirus
    - Examples: ThreatFire, Norton AntiBot
  - Malware Analysis Services
    - Examples: Anubis, CWSandbox

DAMBALLA

x86 Fast System Call Facility

- **SYSENTER instruction**
  - Executed when a process makes a Native API or system call
  - Used to transition from ring 3 (user space) to ring 0 (kernel space)
- **SYSENTER_EIP_MSR**
  - Used by SYSENTER to set the instruction pointer to the address of the system call handler's entrypoint

Coarse-Grained via Intel VT

- **Idea: Combine fast system call facility with host's MMU duties**
  - Proposed by Dinaburg
- **External coarse-grained tracing**
  - Host sets SYSENTER_EIP_MSR to unallocated kernel memory address
  - Guest makes system call
    - After SYSENTER is executed, a page fault occurs that is preempted by the host
    - Host then restores guest's instruction pointer to the original value and resumes guest

DAMBALLA

Azure

- **Named after the rootkit that relies on similar principles for operation**
- **Proof of concept malware analysis tool for Windows XP-based guests**
  - Operates through Intel VT
  - Implemented using KVM
- **Uses**
  - VMI to identify target process
  - Fine-grained tracing to monitor its behavior
- **Coarse-grained tracing left for future work**

DAMBALLA

Azure: VMI

- **Starting with guest context switch**
  - Fixed offset from FS:[0] contains guest address of ETHREAD kernel structure
  - Fixed offset into ETHREAD contains address of EPROCESS kernel structure
  - EPROCESS contains process name, other useful pointers
- **On match, records**
  - CR3 of target process
  - Information from structures such as the PEB (process entrypoint, imagebase, etc.)

Azure: Fine-Grained

- **Upon identifying target process**
  - Sets guest’s trap flag
  - Updates exception bitmap to receive preemptive notification of corresponding debug exception
- **When guest is resumed**
  - Debug exception thrown immediately after execution of next instruction
  - Preempted by host, which repeats the above process until next context switch

DAMBALLA

Azure: Fine-Grained Cont'd

- **Implementation Corner Cases**
  - Interrupt-disabling instructions (e.g., MOV:SS and HLT)
    - Prevent interruptions during execution of next instruction
    - Must modify guest interruptability state
  - Target process' use of PUSHF, POPF and the trap flag
    - Trap flag may need to be filtered out when FLAGS is read by the target (Azure does naïve filtering)
    - Debug exception should be forwarded when target process has set the trap flag

DAMBALLA

Experimentation

- **Azure could be extended into a precision automated unpacker**
  - While performing fine-grained tracing read, disassemble each instruction
    - Track memory-write instructions
  - If the instruction pointer contains an address in the set of written locations
    - Use guest reads to snapshot the unpacked code
    - Clear the set of write locations but continue execution to see if multiple packing layers are present

Experiment Setup

- **Azure's ability to act as an automated unpacker evaluated alongside other approaches**
  - Saffron (in-guest)
  - Renovo (whole-system emulation)
- **Acquired synthetically packed sample set used to test Renovo**
  - Represents 15 packers used to obfuscate vast majority of modern malware

DAMBALLA

Test Criteria

- **Determined whether a sample was successfully unpacked by searching for the original program's code**
  - Used a 32 byte string representing instructions at a fixed offset from the original program's entry point
  - Offset used due to avoid instruction and API virtualization
- **Saffron/Renovo**
  - Searched unpacked layer(s) for the presence of the 32 byte string
- **Azure**
  - Due to time limitations, Azure was instead modified to read 32 bytes starting at the address of the guest instruction pointer following execution of each instruction
  - Data read is then compared to the 32 byte string found in the original program
  - A match indicates Azure traced the target through execution of the original program's code

Results

Packer	Azure	Renovo	Saffron
Armadillo	Yes	No	No
Aspack	Yes	Yes	Yes
Asprotect	Yes	Yes	Yes
FSG	Yes	Yes	Yes
MEW	Yes	Yes	Yes
Molebox	Yes	Yes	Part
Morphine	Yes	Yes	Yes
Obsidium	Yes	No	Part
PECompact	Yes	Yes	Yes
Themida	Yes	Yes	Part
Themida VM	Yes	Part	Part
UPX	Yes	Yes	Yes
UPX S	Yes	Yes	Yes
WinUPack	Yes	Yes	Yes
Yoda's Prot	Yes	Yes	Yes

Label	Meaning
Yes	String found in unpacked code
No	No unpacked code found
Part	Unpacked code found, but string not present

DAMBALLA

Conclusion

- **Analyzing modern malware can be difficult**
- **Intel VT can be used to perform external, transparent malware analysis**
  - Virtual Machine Introspection
  - Fine-Grained Tracing
  - Coarse-Grained Tracing
- **Experiments with Azure show that this approach offers significant transparency**

DAMBALLA

Future Work

- **Ether**
  - In-development malware analyzer based on Xen (with Intel VT)
    - Includes complete automated unpacker and system call tracer
  - Based off joint research between GTISC and Damballa
- **Upcoming paper on Ether in ACM CCS**
  - Ether: Malware Analysis via Hardware Virtualization Extensions
    - Artem Dinaburg, Paul Royal, Monirul Sharif, Wenke Lee
  - Publication will coincide with source code release
    - See http://ether.gtisc.gatech.edu

DAMBALLA

Questions?

Azure Source Download

http://code.google.com/p/azurema