Advanced Internet Security

Malware

Adrian Dabrowski
Aljosha Judmayer
Christian Kudera
Georg Merzdovnik

News from the Field

- Facebook hack: People's accounts appear for sale on dark web
  - Hacked Facebook accounts are being sold on the dark web, just days after the social network revealed 50 million of its users had been compromised
  - The hacked accounts are selling for between $3 and $12
  - If sold individually at these prices, the value of the stolen data on the black market would be somewhere between $150m and $600m

Source: https://www.independent.co.uk/life-style/gadgets-and-tech/news/facebook-hack-data-dark-web-login-details-cost-dream-market-a8564671.html

News from the Field

- Torii botnet - Not another Mirai variant
  - Avast observed a new malware strain, which they call Torii
  - It differs from Mirai and other botnets, particularly in the advanced techniques it uses
    - It tries to be more stealthy and persistent once the device is compromised
    - At the moment it stays silence and it does not (yet) do the usual stuff a botnet does
  - Torii can infect a wide range of devices and it provides support for a wide range of target architectures, including MIPS, ARM, x86, x64, PowerPC, SuperH, and others
  - Further details at the Avast Blog

    https://blog.avast.com/new-torii-botnet-threat-research

Introduction

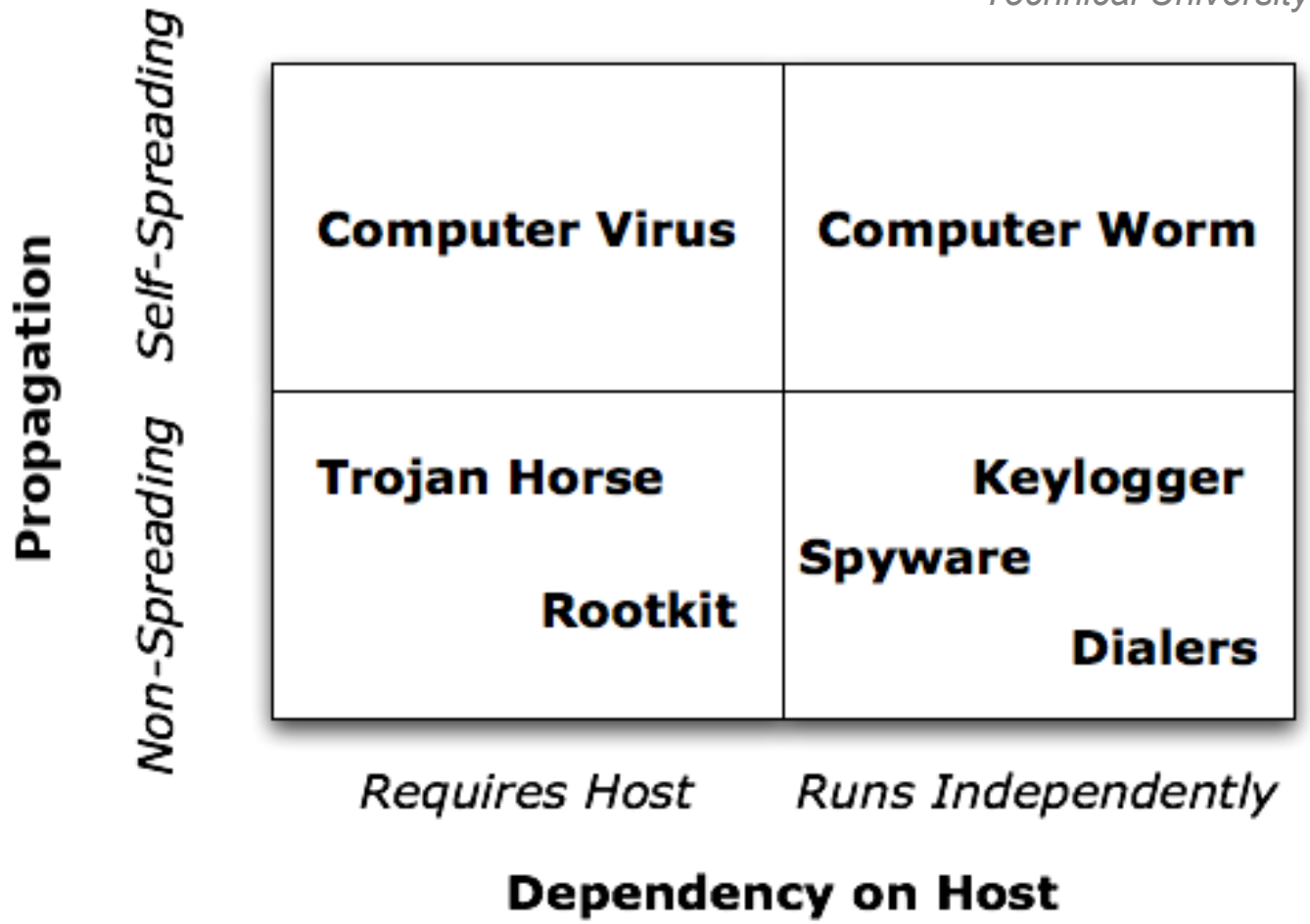
- Malware
  - Stands for "**Mal**icious Soft**ware**"
  - Software specifically designed to harm the user's computer or data
  - Term often incorrectly used equivalent with term virus (due to media coverage)
    - However, many different types exist!

Malware Taxonomy

Malware Types

- Virus
  - Self-replicating
  - Requires host file
- Worm
  - Self-replicating
  - Stand-alone software
- Trojan (Horse)
- Rootkit / Bootkit
- Keylogger
- Spyware / Adware / Crimeware / Ransomware

Malware Types



Propagation Channels

- Drive-by download
  - Unintended download of computer software from the Internet
- Unsolicited E-Mail
  - Unwanted attachments or embedded links in electronic mail
- Physical media
  - Integrated or removable media such as USB drives
- Self propagation
  - Ability of malware to move itself from computer to computer or network to network, thus spreading on its own

Reasons for Malware Propagation

- Mixing of data and code
  - Violates important design property of secure systems
  - Unfortunately very frequent (good example: JIT)
- Homogeneous computing base
  - Windows is just a very tempting target with huge market share
- Unprecedented connectivity
  - Easy to attack from safety of home
- Clueless user base
  - Many targets available, social engineering successful
- Malicious code has become profitable
  - Compromised computers can be sold (e.g. spam, DoS attack, steal sensitive information, crypto mining, ...)

Virus

Main Virus Parts

- Infection mechanism / Infection vector
  - How the virus spreads or propagates
  - A virus typically has a search routine, which locates new files or new disks for infection
- Trigger / Logic bomb
  - Determines when the payload of the virus should be executed (particular date, particular time, particular presence of another program, ...)
- Payload
  - The payload is the part that perform the malicious behavior

Virus Phases / Virus Life Cycle

- Dormant phase
  - The virus is idle during this stage and does not take any action
- Propagation phase
  - The virus starts multiplying and replicating itself
  - The copy may not be identical to the propagating version to evade detection
- Triggering phase
  - The Virus is activated to perform the function for which it was intended
- Execution phase
  - The payload is executed

Virus Infection Strategies

- Boot viruses
  - Master boot record (MBR) of hard disk (first sector on disk)
  - Boot sector of partitions
- File infectors
  - Simple overwrite virus (damages original program)
  - Parasitic virus
    - Append virus code and modify program entry point
    - Changes program size
  - Cavity virus
    - Inject code into unused regions of program code and modify program entry point
    - Program size stays the same

File Infector Techniques

- Anti virus developers quickly discovered to search around entry point

- Entry Point Obfuscation
  - Virus hijacks control later (after program is launched)
  - Overwrite import table addresses
  - Overwrite function call instructions

- Code Integration
  - Merge virus code with program
  - Requires disassembly of target
    - Difficult task on x86 machines

Worms

Computer Worms

A self-replicating program able to propagate itself across networks, typically having a detrimental effect.

(Oxford English Dictionary)

- Worms either
  - Exploit vulnerabilities that affect large number of hosts
  - Send copies of worm body via email
- Difference to classic virus is *autonomous* spread over network
- Speed of spreading is constantly increasing
- Make use of techniques known by virus writers for a long time

Computer Worm Components

- Target locator
  - How to choose new victims
- Infection propagator
  - How to obtain control of victim
  - How to transfer worm body to target system
- Life cycle manager
  - Controls the life time of a worm
    - E.g. suicide on particular date
- Payload
  - The payload is the part that perform the malicious behavior
  - Today, often a botnet client

Target Locator

- Email harvesting
  - Consult address books
  - Files might contain email addresses
    - Inbox of email client
    - Internet Explorer cache and personal directories
  - Even Google searches are possible

- Network share enumeration
  - Windows discovers local computers, which can be attacked
  - Some worms attack everything, including network printers
    - Prints random garbage

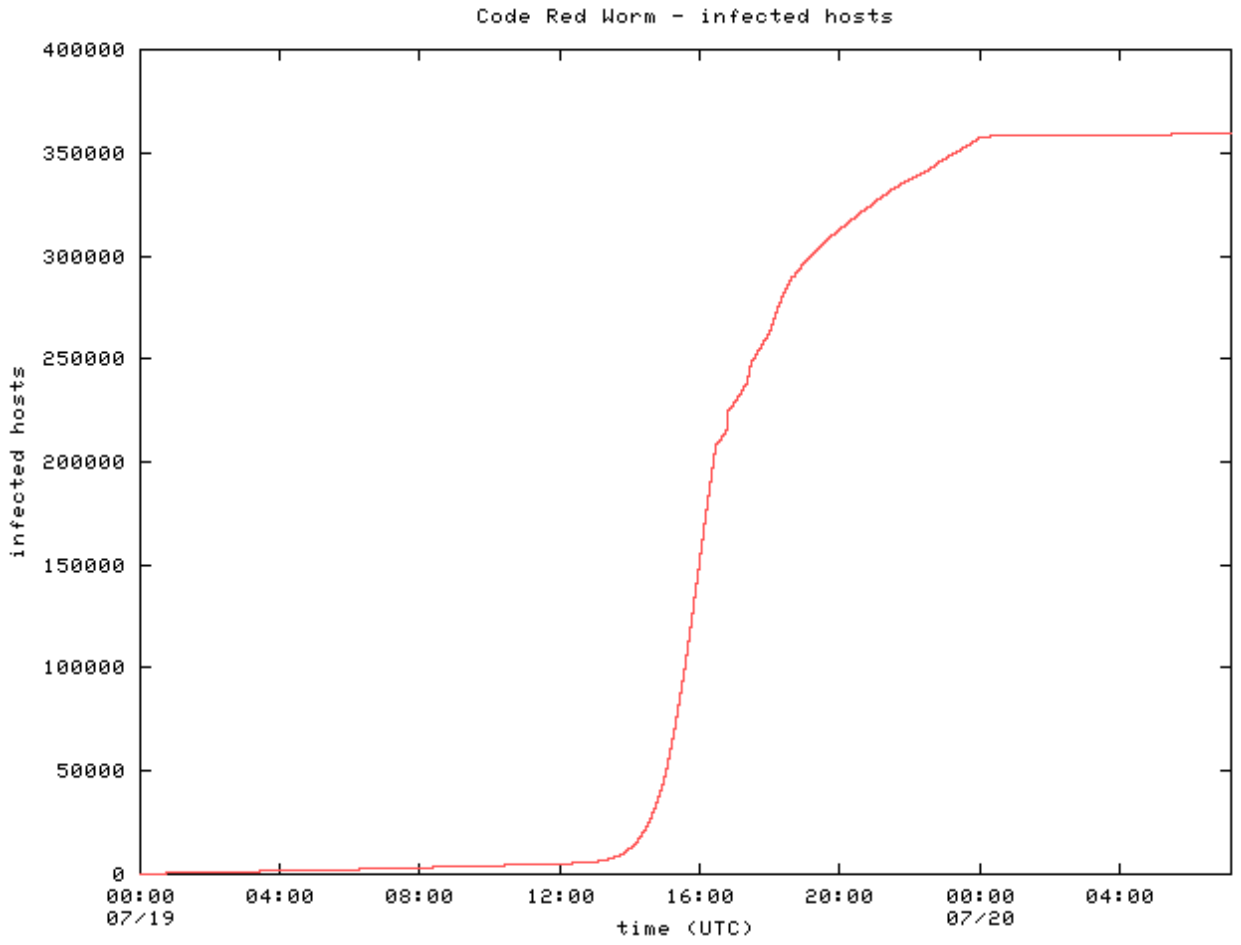
Target Locator

- Scanning
  - Randomly generate IP addresses and send probes
  - Interestingly, many random number generators are buggy
    - Static seed
    - Not complete coverage of address space
- Service discovery and OS fingerprinting performed as well

Exploit-Based Worms

- Require no human interaction
  - Typically exploit well-known network services
  - Can spread much faster
- Propagation speed limited either
  - By network latency
    Worm thread has to establish TCP connection
  - By bandwidth
    Worm can send (UDP) packets as fast as possible
- Spread can be modeled using classic disease model
  - Worm starts slow (only few machines infected)
  - Enters phase of exponential growth
  - Final phase where only few uncompromised machines left

Exploit-Based Worms



Other classes of Malware

Other classes of Malware

- Ransomware
  - Infects host computer
    - May encrypts user files (User can't use encrypted file)
    - May encrypts boot sector (User can't start computer)
  - Demands user to pay for decryption
  - Vast support network with help-desk!

  - Remarkable incidents
    - Chernobyl's radiation monitoring system hit by Ransomware Petya (Monitoring was then performed manually)
    - The city of Atlanta spent $2.6M to recover from a $52,000 ransomware Scare (Payment portal was offline)

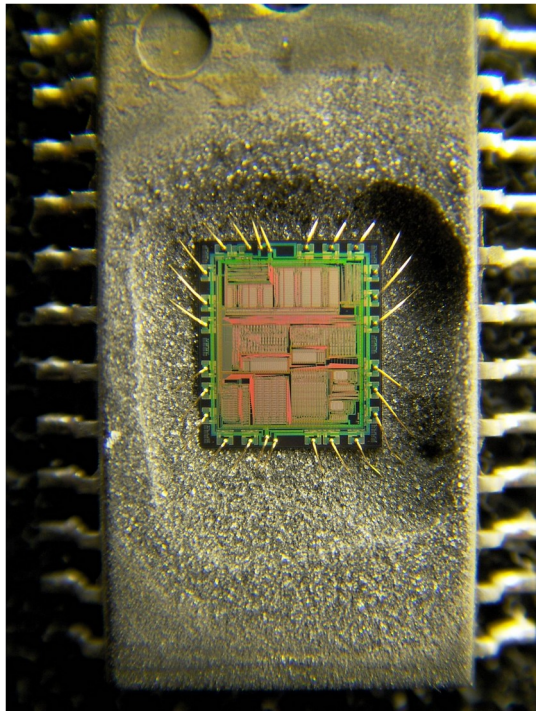
Other classes of Malware

- Hardware implants / Firmware injections
  - BIOS/UEFI, HDD Controller, PCI BootROMs, NICs etc...
  - Very early and low level access
  - Hard to detect and to remove (very resilient, requires reflashing of chip, may be self sustaining using sw-flash tools)

Other classes of Malware

- Backdoors in Integrated Circuits
  - “Our Trojan is capable of reducing the security of the produced random number from 128 bits to n bits, where n can be chosen. Despite these changes, the modified Trojan RNG passes not only the Built-In-Self-Test (BIST) but also generates random numbers that pass the NIST test suite for random numbers.”

“Stealthy Dopant-Level Hardware Trojans” - Becker, CHES 2013”



Integrated Circuit Example

Camouflage Malware: Bypassing Antivirus Software

AV-Software Detection Mechanisms

- Signature-based detection
  - Database of byte-level or instruction-level signatures that match malware
  - Wildcards and regular expressions can be used

- Heuristics-based detection
  - Aims at generically detecting new malware by statically examining files for suspicious characteristics
    - Code execution starts in last section
    - Incorrect header size in PE header
    - Suspicious code section name
    - Patched import address table

AV-Software Detection Mechanisms

- Behavioral detection
  - Observes how the program executes
  - Looking for suspicious behaviors, such as unpacking of malcode, modifying the hosts file or observing keystrokes

- Cloud-based detection
  - Identifies malware by collecting data from protected computers while analyzing it on the provider's infrastructure
  - The vendor's cloud engine can derive patterns and behavior by correlating data from multiple systems
  - A cloud-based engine allows users of the antivirus tool to benefit from the experiences of other members

Camouflage Evolution Overview

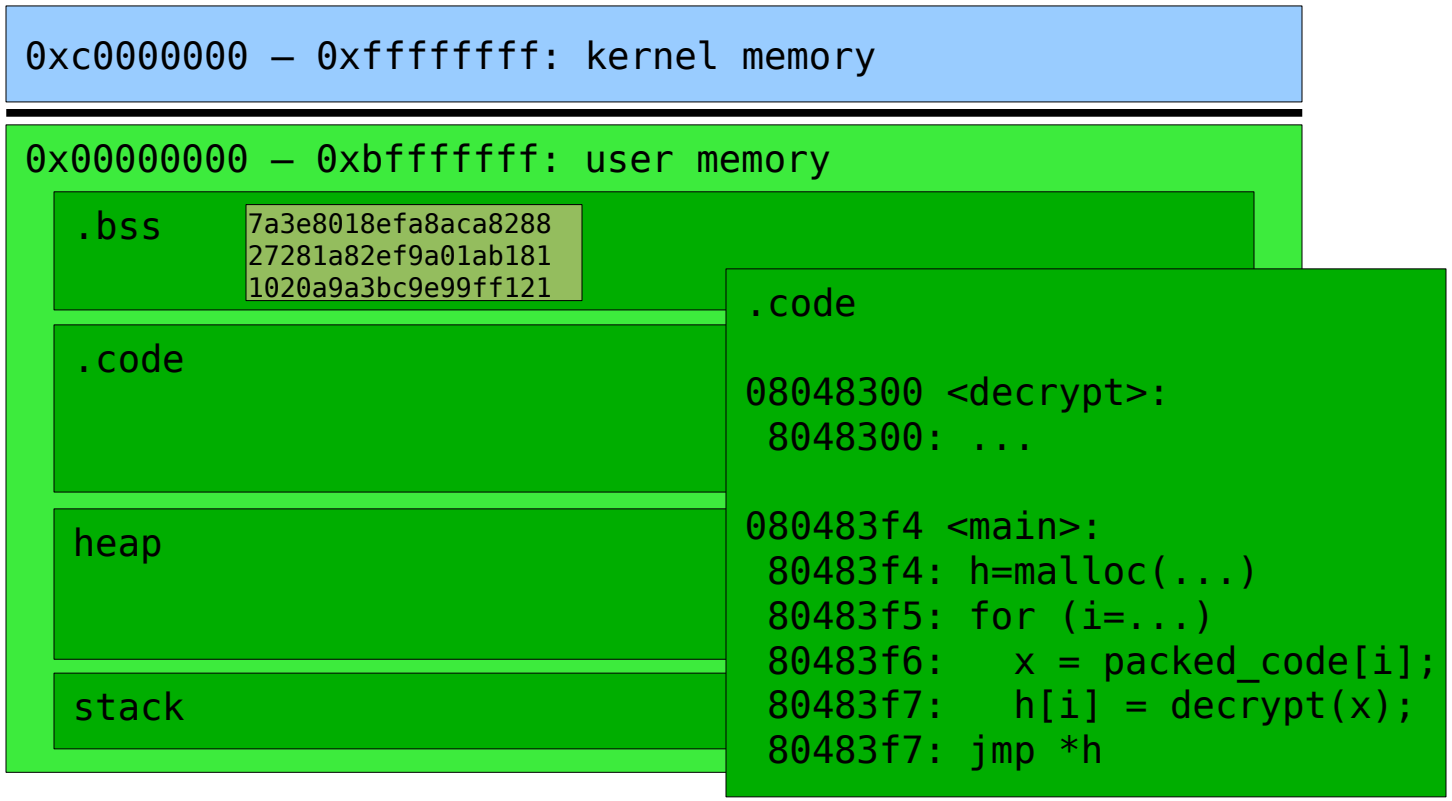
- Encryption (Packed code)
- Semi-Polymorphism (Oligomorphism)
- Polymorphism
- Metamorphism

Encryption (Packed code)

- Encrypted malware has two basic sections
  - Decryption loop
    - Short piece of code, which is responsible to encrypt and decrypt the code of main body
  - Main body
    - The main body is the actual code of the malware, encrypted, and is not meaningful before it is being decrypted

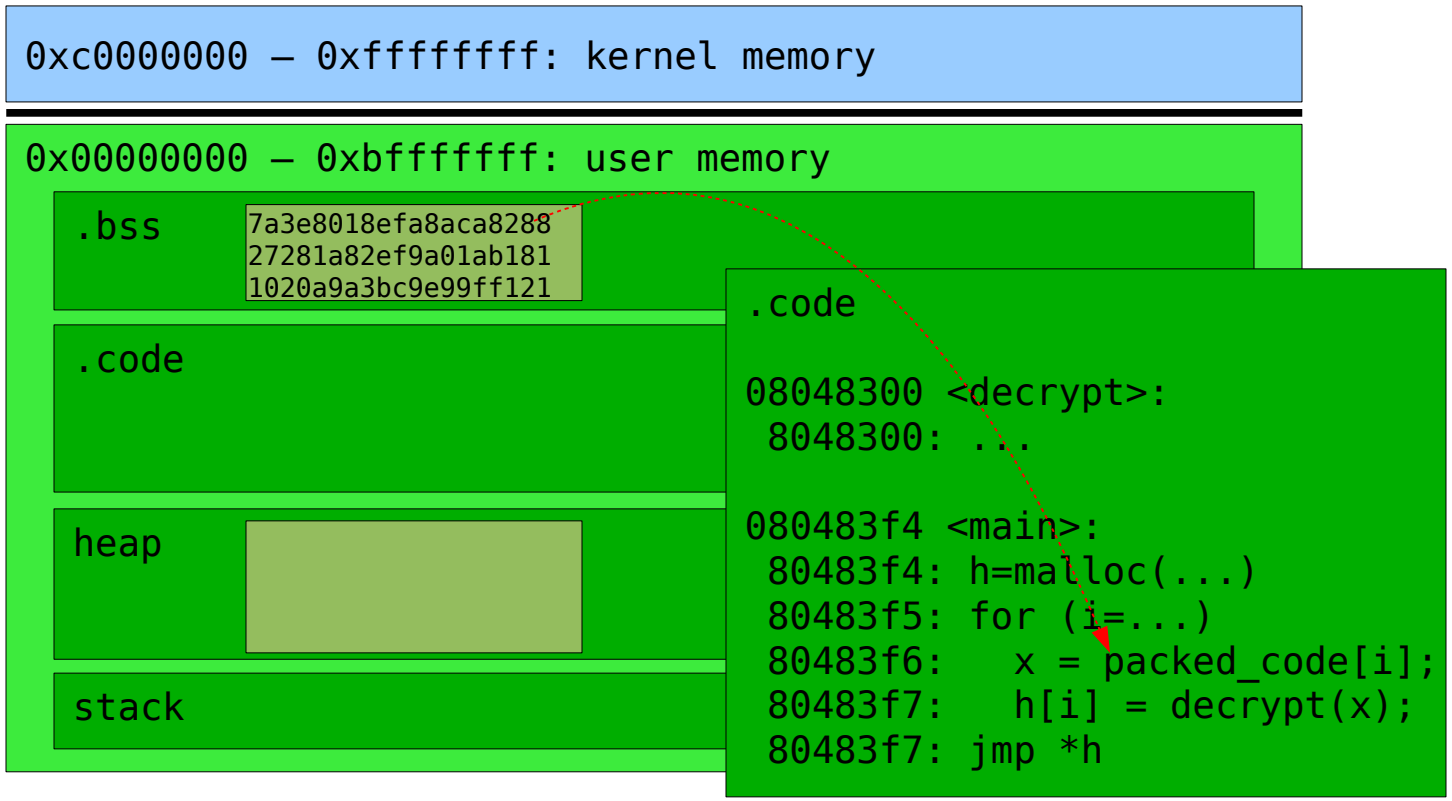
Encryption (Packed code)

- Packed code (*dynamic unpacking*)



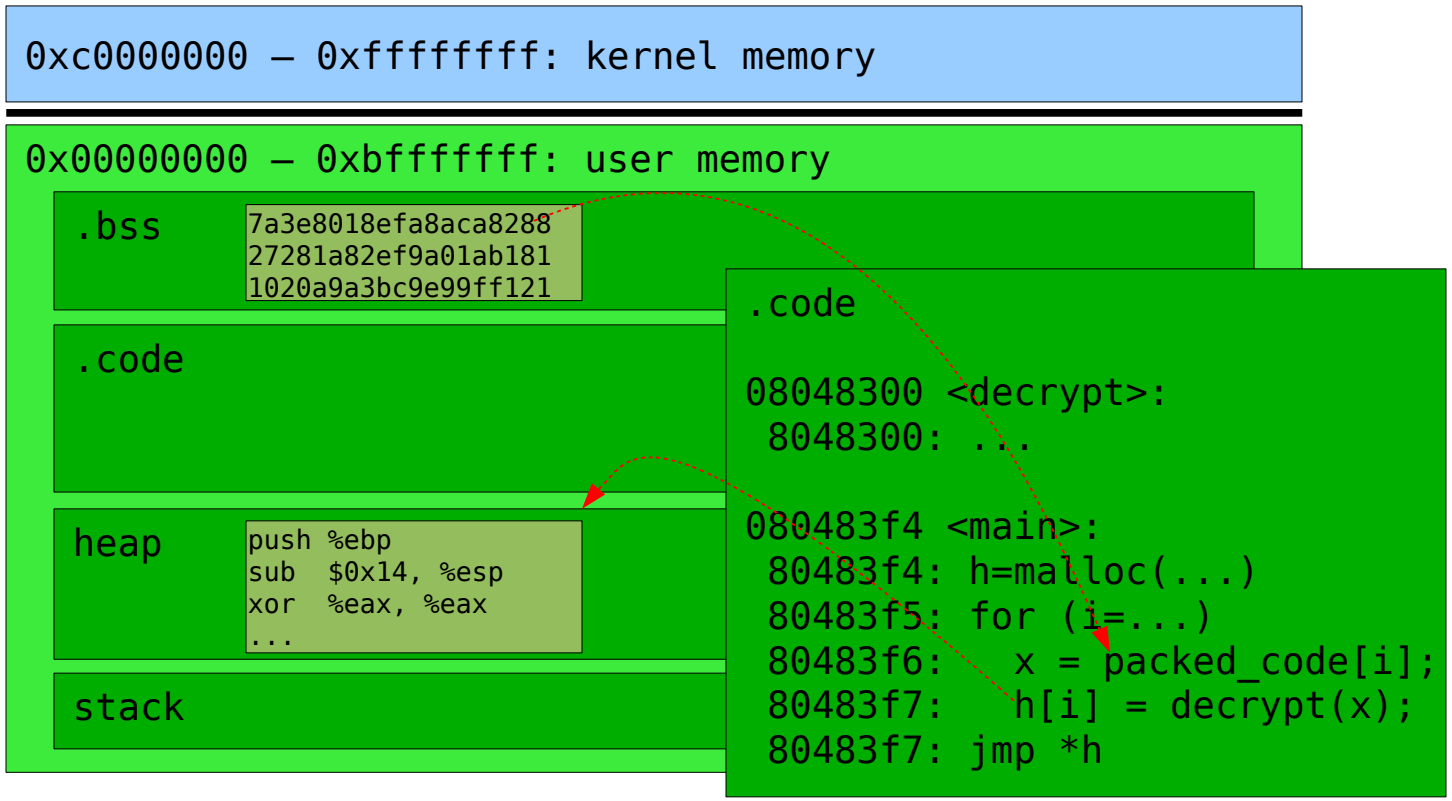
Encryption (Packed code)

- Packed code (*dynamic unpacking*)



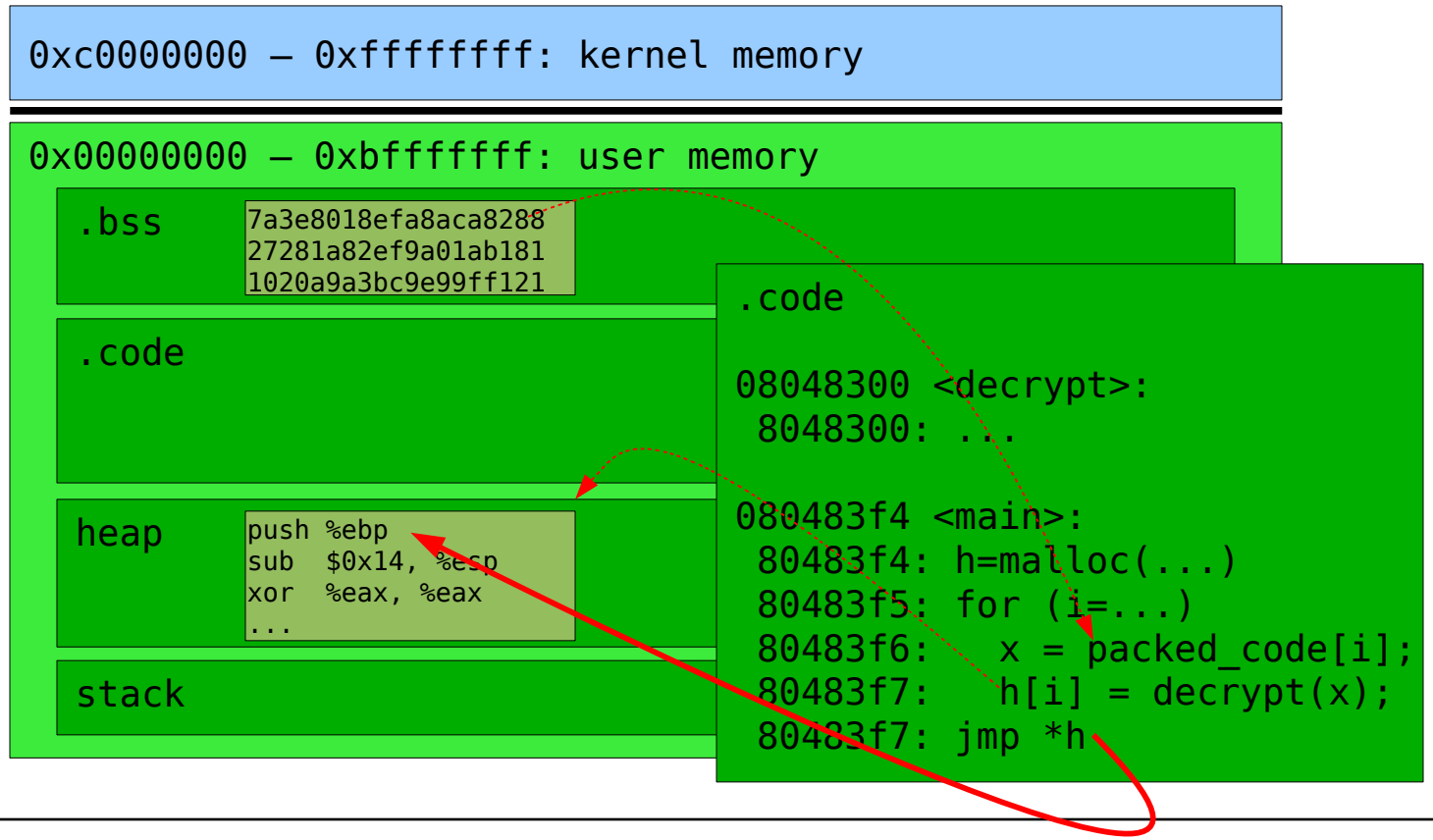
Encryption (Packed code)

- Packed code (*dynamic unpacking*)



Encryption (Packed code)

- Packed code (*dynamic unpacking*)



(Semi-) Polymorphism

- Semi-Polymorphism (Oligomorphism)
  - Contains a collection of different decryptors, which are randomly chosen for a new victim
  - Therefore, the decryptor code is not identical in various instances

- Polymorphism
  - Polymorphic malware are similar to encrypted and oligomorphic malware in usage of code encryption
  - The difference is that polymorphics able to create an unlimited number of new different decryptors (**Mutation Engine**)
    - Utilizes **code obfuscation** techniques to build a new decryptor

Metamorphism

- Metamorphic technique
  - Create different “versions” of malware that look different but have the same semantics (i.e., do the same)
  - A metamorphic engine is used to build the different versions
    - Components: Disassembler, Code analyzer, Code transformer, Assembler
    - Utilizes **code obfuscation** techniques
  - Metamorphic malware does not need encrypted parts

Malware Code Obfuscation Techniques

Obfuscation Techniques Overview

- Dead Code Insertion
- Instruction Reordering
- Instruction Substitution

Dead Code Insertion


```
5B 00 00 00 00      pop ebx
8D 4B 42            lea ecx, [ebx + 42h]
51                  push ecx
50                  push eax
50                  push eax
0F 01 4C 24 FE      sidt [esp - 02h]
5B                  pop ebx
83 C3 1C            add ebx, 1Ch
FA                  cli
8B 2B               mov ebp, [ebx]
```

```
5B 00 00 00 00 8D 4B 42 51 50 50 0F 01 4C 24 FE 5B
83 C3 1C FA 8B 2B
```

Dead Code Insertion


```
5B 00 00 00 00    pop ebx
8D 4B 42          lea ecx, [ebx + 42h]
51                push ecx
50                push eax
90                nop
50                push eax
40                inc eax
0F 01 4C 24 FE    sidt [esp - 02h]
48                dec eax
5B                pop ebx
83 C3 1C          add ebx, 1Ch
FA                cli
8B 2B             mov ebp, [ebx]
```

```
5B 00 00 00 00 8D 4B 42 51 50 90 50 40 0F 01 4C 24 FE
48 5B 83 C3 1C FA 8B 2B
```

Dead Code Insertion


```
5B 00 00 00 00    pop ebx
8D 4B 42          lea ecx, [ebx + 42h]
51                push ecx
50                push eax
90                nop
50                push eax
40                inc eax
0F 01 4C 24 FE    sidt [esp - 02h]
48                dec eax
5B                pop ebx
83 C3 1C          add ebx, 1Ch
FA                cli
8B 2B             mov ebp, [ebx]
```

```
5B 00 00 00 00 8D 4B 42 51 50 90 50 40 0F 01 4C 24 FE
48 5B 83 C3 1C FA 8B 2B
```

Instruction Reordering


```
5B 00 00 00 00      pop ebx
EB 09               jmp <S1>
                    S2:
50                  push eax
0F 01 4C 24 FE      sidt [esp - 02h]
5B                  pop ebx
EB 07               jmp <S3>
                    S1:
8D 4B 42            lea ecx, [ebx + 42h]
51                  push ecx
50                  push eax
EB F0               jmp <S2>
                    S3:
83 C3 1C            add ebx, 1Ch
FA                  cli
8B 2B               mov ebp, [ebx]
```

```
5B 00 00 00 00 EB 09 50 0F 01 4C 24 FE 5B EB 07 8D
4B 42 51 50 EB F0 83 C3 1C FA 8B 2B
```

Instruction Reordering


```
5B 00 00 00 00      pop ebx                    1
EB 09               jmp <S1>
                    S2:
50                  push eax
0F 01 4C 24 FE      sidt [esp - 02h]
5B                  pop ebx
EB 07               jmp <S3>
                    S1:
8D 4B 42            lea ecx, [ebx + 42h]
51                  push ecx
50                  push eax
EB F0               jmp <S2>
                    S3:
83 C3 1C            add ebx, 1Ch
FA                  cli
8B 2B               mov ebp, [ebx]
```

```
5B 00 00 00 00 EB 09 50 0F 01 4C 24 FE 5B EB 07 8D
4B 42 51 50 EB F0 83 C3 1C FA 8B 2B
```

Instruction Reordering


```
5B 00 00 00 00      pop ebx
EB 09               jmp <S1>
                    S2:
50                  push eax
0F 01 4C 24 FE      sidt [esp - 02h]
5B                  pop ebx
EB 07               jmp <S3>
                    S1:
8D 4B 42            lea ecx, [ebx + 42h]
51                  push ecx
50                  push eax
EB F0               jmp <S2>
                    S3:
83 C3 1C            add ebx, 1Ch
FA                  cli
8B 2B               mov ebp, [ebx]
```

2

```
5B 00 00 00 00 EB 09 50 0F 01 4C 24 FE 5B EB 07 8D
4B 42 51 50 EB F0 83 C3 1C FA 8B 2B
```

Instruction Reordering


```
5B 00 00 00 00      pop ebx
EB 09               jmp <S1>
                    S2:
50                  push eax
0F 01 4C 24 FE      sidt [esp - 02h]
5B                  pop ebx
EB 07               jmp <S3>
                    S1:
8D 4B 42            lea ecx, [ebx + 42h]
51                  push ecx
50                  push eax
EB F0               jmp <S2>
                    S3:
83 C3 1C            add ebx, 1Ch
FA                  cli
8B 2B               mov ebp, [ebx]
```

3

```
5B 00 00 00 00 EB 09 50 0F 01 4C 24 FE 5B EB 07 8D
4B 42 51 50 EB F0 83 C3 1C FA 8B 2B
```

Instruction Reordering


```
5B 00 00 00 00      pop ebx
EB 09               jmp <S1>
                    S2:
50                  push eax
0F 01 4C 24 FE      sidt [esp - 02h]
5B                  pop ebx
EB 07               jmp <S3>
                    S1:
8D 4B 42            lea ecx, [ebx + 42h]
51                  push ecx
50                  push eax
EB F0               jmp <S2>
                    S3:
83 C3 1C            add ebx, 1Ch
FA                  cli                          4
8B 2B               mov ebp, [ebx]
```

```
5B 00 00 00 00 EB 09 50 0F 01 4C 24 FE 5B EB 07 8D
4B 42 51 50 EB F0 83 C3 1C FA 8B 2B4
```

Instruction Reordering


```
5B 00 00 00 00      pop ebx
EB 09               jmp <S1>
                    S2:
50                  push eax
0F 01 4C 24 FE      sidt [esp - 02h]          3
5B                  pop ebx
EB 07               jmp <S3>
                    S1:
8D 4B 42            lea ecx, [ebx + 42h]
51                  push ecx                  2
50                  push eax
EB F0               jmp <S2>
                    S3:
83 C3 1C            add ebx, 1Ch
FA                  cli
8B 2B               mov ebp, [ebx]
```

```
5B 00 00 00 00 EB 09 50 0F 01 4C 24 FE 5B EB 07 8D
4B 42 51 50 EB F0 83 C3 1C FA 8B 2B
```

Instruction Substitution


```
5B 00 00 00 00      pop ebx
8D 4B 42            lea ecx, [ebx + 42h]
51                  push ecx
50                  push eax
50                  push eax
0F 01 4C 24 FE      sidt [esp - 02h]
5B                  pop ebx
83 C3 1C            add ebx, 1Ch
FA                  cli
8B 2B               mov ebp, [ebx]
```

```
5B 00 00 00 00 8D 4B 42 51 50 50 0F 01 4C 24 FE 5B
83 C3 1C FA 8B 2B
```

Instruction Substitution

5B 00 00 00 00	pop ebx
8D 4B 42	lea ecx, [ebx + 42h]
51	push ecx
50	push eax
89 04 24	**mov eax, [esp]**
83 C4 04	**add 04h, esp**
0F 01 4C 24 FE	sidt [esp - 02h]
5B	pop ebx
83 C3 1C	add ebx, 1Ch
FA	cli
8B 2B	mov ebp, [ebx]

```
5B 00 00 00 00 8D 4B 42 51 50 89 04 24 83 C4 04 0F
01 4C 24 FE 5B 83 C3 1C FA 8B 2B
```

Instruction Substitution

5B 00 00 00 00	pop ebx
8D 4B 42	lea ecx, [ebx + 42h]
51	push ecx
50	push eax
89 04 24	**mov eax, [esp]**
83 C4 04	**add 04h, esp**
0F 01 4C 24 FE	sidt [esp - 02h]
83 04 24 0C	**add 1Ch, [esp]**
5B	pop ebx
FA	cli
8B 2B	mov ebp, [ebx]

```
5B 00 00 00 00 8D 4B 42 51 50 89 04 24 83 C4 04 0F
01 4C 24 FE 83 04 24 0C 5B FA 8B 2B
```

Detecting Execution Environments

Detecting Virtualization

- If a program is not run natively (uninstrumented) on a machine, chances are high that it
  - Is being analyzed (In a security lab)
  - Scanned (Inside a sandbox of an Antivirus product)
  - Debugged (By a security specialist)
- Modern malware detect execution environment to complicate analysis
- Detection generally easy, but depends on environment:
  - Virtual machine: *very* easy
  - Hardware supported virtual machine: adjusted techniques, still easy
  - Emulator: *theoretically* undetectable, practically also easy to detect

Detecting Virtualization Overview

- Logical Discrepancies
- Resource Discrepancies
- Timing Discrepancies

Logical Discrepancies

- CPU discrepancies
  - Difference in (x86) interface
  - Interface bugs
    - QEMU: Setting incorrect status bits in exception/corner cases
    - QEMU: Accepting malformed instructions (e.g., overly long instruction sequences)
  - Incomplete implementation of rarely used interface parts
    - QEMU: A*lignment check enforcement* missing
    - QEMU / VMware: Caching essential part of system performance; requests to disabling this ignored

Logical Discrepancies

- VMware: Detect VMM interface
  - Guest code can interact with VMM
  - Supported functionality: Query VMM version, access host clipboard, etc.
  - Implemented through privileged instruction that is caught by VMM
  - Crashes on physical host (when called from user-land) → code must catch *segmentation violation*

```
movl $0x564D5868, %eax   /* magic value */
movl $0x00000000, %ebx   /* initialize output variable */
movl $0x0000000a, %ecx   /* function: get VMWare version */
movl $0x00005658, %edx   /* magic value */

in %dx, %eax             /* read from VMware port */

movl %ebx, [result]      /* get result */
```

Logical Discrepancies

- VirtualPC: Detect VMM interface
  - Guest code can interact with VMM
  - Supported functionality: Query VMM version, access host clipboard, etc.
  - Implemented through illegal instruction (VMM called automatically)
  - Crashes on physical host (when called from user-land) → code must catch *exception*

```
movl $0, %ebx   /* initialize output variable */
movl $0, %eax   /* VPC function number */

.byte 0x0f      /* call VPC (magic invalid instruction) */
.byte 0x3f
.byte 0x07
.byte 0x0b

movl %ebx, [result]   /* get result from VPC (or crash) */
```

Logical Discrepancies

- Off-Chip
  - Most VMMs implement very few & basic hardware components
  - Motherboard, CPU, video card, NIC
  - Remain unchanged for simplicity / compatibility
  - Leads to *absurd* (unrealistic) hardware configurations

    "Two AMD Opteron CPUs and 8 GB of RAM in an Intel motherboard from the Clinton administration"

  - Can also be detected by checking for well-known driver names, serial numbers, etc.

Resource Discrepancies

- VMMs (host) share physical resources with guests
  - CPU cycles, physical memory, cache footprint, persistent storage, ...
- Detect VMM by measuring cache effectiveness
  - Example: TLB (translation lookaside buffer) for caching mapping between virtual pages and frames in RAM
  - Step 1: calculate base value
    - Access large amounts of memory areas in loop
    - Measure amount of TLB misses
  - Step 2: check for VMM
    - Restart loop, interleave memory access with instructions that trigger the VMM
    - VMM code will generate additional TLB misses
  - Difference of TLB misses indicates presence of VMM

Timing Discrepancies

- Absolute timing
  - Privileged instructions need intervention by the VMM
  - Additional latency added on virtualized guest code
  - Slowdown can be measured (with access to an external clock)

- Latency variance
  - Access to *physical* hardware (register) often requires fixed amount of time
  - Subsequent reads of *virtual* hardware (registers) might be cached
  - Virtual hardware access has higher run-to-run variance in latency

- Additional latency
  - VMM must protect memory resources from direct access through guest code (e.g., RAM used by VMM)
  - Direct access cause hidden page faults that must be handled by VMM
  - Orders of magnitude slower than on non-virtualized host

Analysis Tools

Specific Malware Analysis Tools

- Cookoo (Malwr.com)
  - Open Source Toolkit
  - http://www.cuckoosandbox.org/

- REMnux (Lenny Zeltser)
  - A Linux Toolkit for Reverse-Engineering and Analyzing Malware
  - https://remnux.org/

Thank you for your attention