OS Security

Malware

Radboud University Nijmegen, The Netherlands

Winter 2014/2015

Last week. . .

A short recap

- Important concept to reduce covert channels and possible damage by an attack: compartmentalization
- Very strong implementation: virtualization

A short recap

- Important concept to reduce covert channels and possible damage by an attack: compartmentalization
- Very strong implementation: virtualization
- Multiple *virtual machines* running on the same physical hardware
- Different ways to implement this

A short recap

- Important concept to reduce covert channels and possible damage by an attack: compartmentalization
- Very strong implementation: virtualization
- Multiple *virtual machines* running on the same physical hardware
- Different ways to implement this
  - Full virtualization (guest OS in ring 1)

A short recap

- Important concept to reduce covert channels and possible damage by an attack: compartmentalization
- Very strong implementation: virtualization
- Multiple *virtual machines* running on the same physical hardware
- Different ways to implement this
  - Full virtualization (guest OS in ring 1)
  - HW-assisted virtualization (guest OS in ring 0, hypervisor in ring -1)

A short recap

- Important concept to reduce covert channels and possible damage by an attack: compartmentalization
- Very strong implementation: virtualization
- Multiple *virtual machines* running on the same physical hardware
- Different ways to implement this
  - Full virtualization (guest OS in ring 1)
  - HW-assisted virtualization (guest OS in ring 0, hypervisor in ring -1)
  - Paravirtualization: modified guest OS in ring 3, host OS in ring 1, hypervisor in ring 0

A short recap

- Important concept to reduce covert channels and possible damage by an attack: compartmentalization
- Very strong implementation: virtualization
- Multiple *virtual machines* running on the same physical hardware
- Different ways to implement this
  - Full virtualization (guest OS in ring 1)
  - HW-assisted virtualization (guest OS in ring 0, hypervisor in ring -1)
  - Paravirtualization: modified guest OS in ring 3, host OS in ring 1, hypervisor in ring 0
  - Host-based virtualization: hypervisor in ring 3, unmodified guest OS in ring 3

A short recap

- Important concept to reduce covert channels and possible damage by an attack: compartmentalization
- Very strong implementation: virtualization
- Multiple *virtual machines* running on the same physical hardware
- Different ways to implement this
  - Full virtualization (guest OS in ring 1)
  - HW-assisted virtualization (guest OS in ring 0, hypervisor in ring -1)
  - Paravirtualization: modified guest OS in ring 3, host OS in ring 1, hypervisor in ring 0
  - Host-based virtualization: hypervisor in ring 3, unmodified guest OS in ring 3
- Weaker way to compartmentalize: sandboxing
- Sandboxing limits access to resources

A short recap

- Important concept to reduce covert channels and possible damage by an attack: compartmentalization
- Very strong implementation: virtualization
- Multiple *virtual machines* running on the same physical hardware
- Different ways to implement this
  - Full virtualization (guest OS in ring 1)
  - HW-assisted virtualization (guest OS in ring 0, hypervisor in ring -1)
  - Paravirtualization: modified guest OS in ring 3, host OS in ring 1, hypervisor in ring 0
  - Host-based virtualization: hypervisor in ring 3, unmodified guest OS in ring 3
- Weaker way to compartmentalize: sandboxing
- Sandboxing limits access to resources
- Attacks typically aim at breaking out of the jail

Ad-hoc solutions for better security

- Completely re-designing an OS is expensive
- More feasible: Add-on security for existing OS
- Multiple techniques:

Ad-hoc solutions for better security

- Completely re-designing an OS is expensive
- More feasible: Add-on security for existing OS
- Multiple techniques:
  - Memory protection (NX bit) and ASLR (two weeks ago)

Ad-hoc solutions for better security

- Completely re-designing an OS is expensive
- More feasible: Add-on security for existing OS
- Multiple techniques:
  - Memory protection (NX bit) and ASLR (two weeks ago)
  - Compartmentalization and virtualization (last lecture)

Ad-hoc solutions for better security

- Completely re-designing an OS is expensive
- More feasible: Add-on security for existing OS
- Multiple techniques:
  - Memory protection (NX bit) and ASLR (two weeks ago)
  - Compartmentalization and virtualization (last lecture)
  - Add mandatory access control (next lecture)

Ad-hoc solutions for better security

- Completely re-designing an OS is expensive
- More feasible: Add-on security for existing OS
- Multiple techniques:
  - Memory protection (NX bit) and ASLR (two weeks ago)
  - Compartmentalization and virtualization (last lecture)
  - Add mandatory access control (next lecture)
  - Detect (or prevent) intrusions and malware (this lecture)

IDS, IPS, AV

- Two kinds of intrusion-detection systems (IDS):
  - Network-based IDS (NIDS)
  - Host-based IDS (HIDS)

IDS, IPS, AV

- Two kinds of intrusion-detection systems (IDS):
  - Network-based IDS (NIDS)
  - Host-based IDS (HIDS)
- Some systems have additional capabilities to *prevent* intrusion
- Those systems are called intrusion-prevention systems (IPS), again:
  - Network-based IPS (NIPS)
  - Host-based IPS (HIPS)

IDS, IPS, AV

- Two kinds of intrusion-detection systems (IDS):
  - Network-based IDS (NIDS)
  - Host-based IDS (HIDS)
- Some systems have additional capabilities to *prevent* intrusion
- Those systems are called intrusion-prevention systems (IPS), again:
  - Network-based IPS (NIPS)
  - Host-based IPS (HIPS)
- Special kind of HIDS: antivirus software (AV)
- AV is typically more generally anti-malware software (aka virus scanners, malware scanners)

Malware

Definition

Malware is malicious software or functionality that a user does not intend to run.

Malware

Definition

Malware is malicious software or functionality that a user does not intend to run.

- Typical features of malware:
  - Some way to trick the user into running it

Malware

Definition

Malware is malicious software or functionality that a user does not intend to run.

- Typical features of malware:
  - Some way to trick the user into running it
  - A damage routine performing the actual malicious behavior

Malware

Definition

Malware is malicious software or functionality that a user does not intend to run.

- Typical features of malware:
  - Some way to trick the user into running it
  - A damage routine performing the actual malicious behavior
  - Often a routine to spread to other computers

Malware

Definition

Malware is malicious software or functionality that a user does not intend to run.

- Typical features of malware:
  - Some way to trick the user into running it
  - A damage routine performing the actual malicious behavior
  - Often a routine to spread to other computers
  - Often functionality to hide from malware scanners

Malware

Definition

Malware is malicious software or functionality that a user does not intend to run.

- Typical features of malware:
  - Some way to trick the user into running it
  - A damage routine performing the actual malicious behavior
  - Often a routine to spread to other computers
  - Often functionality to hide from malware scanners
- Different ways to categorize malware:
  - By their malicious behavior (what they do)
  - By their spreading routine
  - By privilege of the malicious code

Viruses

- A virus infects a host program:
  - Copy itself into the host program
  - Change entry point to the entry point of the virus
  - Change the return address from the virus code to the original entry point

Viruses

- A virus infects a host program:
  - Copy itself into the host program
  - Change entry point to the entry point of the virus
  - Change the return address from the virus code to the original entry point
- Characteristic for a virus: it spreads by infecting other files

Viruses

- A virus infects a host program:
  - Copy itself into the host program
  - Change entry point to the entry point of the virus
  - Change the return address from the virus code to the original entry point
- Characteristic for a virus: it spreads by infecting other files
- Viruses traditionally need an executable host file (e.g., .exe, .bat, .vbs)
- More general: can also infect office files with macros (macro virus)

Viruses

- A virus infects a host program:
  - Copy itself into the host program
  - Change entry point to the entry point of the virus
  - Change the return address from the virus code to the original entry point
- Characteristic for a virus: it spreads by infecting other files
- Viruses traditionally need an executable host file (e.g., .exe, .bat, .vbs)
- More general: can also infect office files with macros (macro virus)
- The earliest viruses are from the 70s spreading in the ARPANET
- Originally most viruses spread over floppy disks
- Today obviously mainly spread over the Internet

Self-replicating code

- A virus needs to replicate (print) itself
- How do you write a program that prints itself?

Self-replicating code

- A virus needs to replicate (print) itself
- How do you write a program that prints itself?
- First attempt (in Python): `print "print 'hello'"`

Self-replicating code

- A virus needs to replicate (print) itself
- How do you write a program that prints itself?
- First attempt (in Python): `print "print 'hello'"`
- Output: `"print 'hello'"`

Self-replicating code

- A virus needs to replicate (print) itself
- How do you write a program that prints itself?
- First attempt (in Python): `print "print 'hello'"`
- Output: `"print 'hello'"`
- Next attempt: `s = 'print %s'; print s % repr(s)`

Self-replicating code

- A virus needs to replicate (print) itself
- How do you write a program that prints itself?
- First attempt (in Python): `print "print 'hello'"`
- Output: `"print 'hello'"`
- Next attempt: `s = 'print %s'; print s % repr(s)`
- Output: `print 'print %s'`

Self-replicating code

- A virus needs to replicate (print) itself
- How do you write a program that prints itself?
- First attempt (in Python): `print "print 'hello'"`
- Output: `"print 'hello'"`
- Next attempt: `s = 'print %s'; print s % repr(s)`
- Output: `print 'print %s'`
- This works:
  `s = 's = %s; print s %% repr(s)'; print s % repr(s)`

Self-replicating code

- A virus needs to replicate (print) itself
- How do you write a program that prints itself?
- First attempt (in Python): `print "print 'hello'"`
- Output: `"print 'hello'"`
- Next attempt: `s = 'print %s'; print s % repr(s)`
- Output: `print 'print %s'`
- This works:
  `s = 's = %s; print s %% repr(s)'; print s % repr(s)`
- Output:
  `s = 's = %s; print s %% repr(s)'; print s % repr(s)`

Self-replicating code

- A virus needs to replicate (print) itself
- How do you write a program that prints itself?
- First attempt (in Python): `print "print 'hello'"`
- Output: `"print 'hello'"`
- Next attempt: `s = 'print %s'; print s % repr(s)`
- Output: `print 'print %s'`
- This works:
  `s = 's = %s; print s %% repr(s)'; print s % repr(s)`
- Output:
  `s = 's = %s; print s %% repr(s)'; print s % repr(s)`
- The central ingredient is recursion!

Worms

- A *worm* is a stand-alone malware program, which spreads without a host program

Worms

- A *worm* is a stand-alone malware program, which spreads without a host program
- Two different ways of spreading:
  1. With user interaction (e.g., by e-mail)
  2. Without user interaction through software vulnerabilities

Worms

- A *worm* is a stand-alone malware program, which spreads without a host program
- Two different ways of spreading:
  1. With user interaction (e.g., by e-mail)
  2. Without user interaction through software vulnerabilities
- Famous example of the first type of worm: Loveletter (aka ILOVEYOU)

Worms

- A *worm* is a stand-alone malware program, which spreads without a host program
- Two different ways of spreading:
  1. With user interaction (e.g., by e-mail)
  2. Without user interaction through software vulnerabilities
- Famous example of the first type of worm: Loveletter (aka ILOVEYOU)
  - Worm that started spreading in May 2000
  - Spread by e-mail with subject line "I love you"
  - Read address book of infected host and sent to the address book (from the user's mail address)
  - Malicious Attachment had filename LOVE-LETTER-FOR-YOU.TXT.vbs (Windows by default did not show the vbs)

Worms

- A *worm* is a stand-alone malware program, which spreads without a host program
- Two different ways of spreading:
  1. With user interaction (e.g., by e-mail)
  2. Without user interaction through software vulnerabilities
- Famous example of the first type of worm: Loveletter (aka ILOVEYOU)
  - Worm that started spreading in May 2000
  - Spread by e-mail with subject line "I love you"
  - Read address book of infected host and sent to the address book (from the user's mail address)
  - Malicious Attachment had filename LOVE-LETTER-FOR-YOU.TXT.vbs (Windows by default did not show the vbs)
  - Deleted all files ending on .jpg, .jpeg, .vbs, .vbe, .js, .jse, .css, .wsh, .sct and .hta and replaced them by a copy of itself (with additional ending .vbs)

Worms

- A *worm* is a stand-alone malware program, which spreads without a host program
- Two different ways of spreading:
  1. With user interaction (e.g., by e-mail)
  2. Without user interaction through software vulnerabilities
- Famous example of the first type of worm: Loveletter (aka ILOVEYOU)
  - Worm that started spreading in May 2000
  - Spread by e-mail with subject line "I love you"
  - Read address book of infected host and sent to the address book (from the user's mail address)
  - Malicious Attachment had filename LOVE-LETTER-FOR-YOU.TXT.vbs (Windows by default did not show the vbs)
  - Deleted all files ending on .jpg, .jpeg, .vbs, .vbe, .js, .jse, .css, .wsh, .sct and .hta and replaced them by a copy of itself (with additional ending .vbs)
  - Caused an estimated damage of US$\$10,000,000,000$

Worms

- A *worm* is a stand-alone malware program, which spreads without a host program
- Two different ways of spreading:
  1. With user interaction (e.g., by e-mail)
  2. Without user interaction through software vulnerabilities
- Example of the second type: Sasser
  - Spread through a buffer overflow in the "Local Security Authority Subsystem Service" (LSASS) in Windows XP and 2000
  - Communication through TCP on ports 445 and 139
  - Services running by default on Windows (and reachable from outside)

Trojans

- *Trojans* offer useful functionality and hidden malicious functionality
- Unlike viruses and worms, trojans are not self-replicating

Trojans

- *Trojans* offer useful functionality and hidden malicious functionality
- Unlike viruses and worms, trojans are not self-replicating
- Trojans can be used for a variety of criminal actions
- Trojans can be used for targeted attacks

Trojans

- *Trojans* offer useful functionality and hidden malicious functionality
- Unlike viruses and worms, trojans are not self-replicating
- Trojans can be used for a variety of criminal actions
- Trojans can be used for targeted attacks
- Trojans are also used by governments to wiretap Internet telephony
- Probably most famous example: German “Staatstrojaner” (aka R2D2 or 0zapftis)

Trojans

- *Trojans* offer useful functionality and hidden malicious functionality
- Unlike viruses and worms, trojans are not self-replicating
- Trojans can be used for a variety of criminal actions
- Trojans can be used for targeted attacks
- Trojans are also used by governments to wiretap Internet telephony
- Probably most famous example: German "Staatstrojaner" (aka R2D2 or 0zapftis)
  - German police may use malware only to wiretap Internet telephony
  - Staatstrojaner was analyzed by Chaos Computer Club in 2011

Trojans

- *Trojans* offer useful functionality and hidden malicious functionality
- Unlike viruses and worms, trojans are not self-replicating
- Trojans can be used for a variety of criminal actions
- Trojans can be used for targeted attacks
- Trojans are also used by governments to wiretap Internet telephony
- Probably most famous example: German "Staatstrojaner" (aka R2D2 or 0zapftis)
  - German police may use malware only to wiretap Internet telephony
  - Staatstrojaner was analyzed by Chaos Computer Club in 2011
  - Staatstrojaner was found to allow remote control, caputure screenshots, fetch upgrades remotely
  - Communication from the trojan was encrypted with AES in ECB mode

Trojans

- *Trojans* offer useful functionality and hidden malicious functionality
- Unlike viruses and worms, trojans are not self-replicating
- Trojans can be used for a variety of criminal actions
- Trojans can be used for targeted attacks
- Trojans are also used by governments to wiretap Internet telephony
- Probably most famous example: German "Staatstrojaner" (aka R2D2 or 0zapftis)
  - German police may use malware only to wiretap Internet telephony
  - Staatstrojaner was analyzed by Chaos Computer Club in 2011
  - Staatstrojaner was found to allow remote control, caputure screenshots, fetch upgrades remotely
  - Communication from the trojan was encrypted with AES in ECB mode
  - Communication to the trojan was unencrypted!

Rootkits

- After compromising a computer, malware (or attackers) typically try to hide their traces
- Software that hides traces of an attack is called *rootkit*

Rootkits

- After compromising a computer, malware (or attackers) typically try to hide their traces
- Software that hides traces of an attack is called *rootkit*
- Most powerful: rootkits running in the kernel:
  - Can hide existence of files by modifying the file-system driver
  - Can hide existence of processes by modifying process management
  - Can create hidden filesytem to store data
  - Can temper with malware scanners

Rootkits

- After compromising a computer, malware (or attackers) typically try to hide their traces
- Software that hides traces of an attack is called *rootkit*
- Most powerful: rootkits running in the kernel:
  - Can hide existence of files by modifying the file-system driver
  - Can hide existence of processes by modifying process management
  - Can create hidden filesytem to store data
  - Can temper with malware scanners
- Possible countermeasure: cryptographically sign all kernel modules and drivers

Rootkits

- After compromising a computer, malware (or attackers) typically try to hide their traces
- Software that hides traces of an attack is called *rootkit*
- Most powerful: rootkits running in the kernel:
  - Can hide existence of files by modifying the file-system driver
  - Can hide existence of processes by modifying process management
  - Can create hidden filesytem to store data
  - Can temper with malware scanners
- Possible countermeasure: cryptographically sign all kernel modules and drivers
- This went horribly wrong with Flame in 2012
- Weaknesses in the MD5 hash function allowed malware to obtain valid signature

Rootkits

- After compromising a computer, malware (or attackers) typically try to hide their traces
- Software that hides traces of an attack is called *rootkit*
- Most powerful: rootkits running in the kernel:
  - Can hide existence of files by modifying the file-system driver
  - Can hide existence of processes by modifying process management
  - Can create hidden filesytem to store data
  - Can temper with malware scanners
- Possible countermeasure: cryptographically sign all kernel modules and drivers
- This went horribly wrong with Flame in 2012
- Weaknesses in the MD5 hash function allowed malware to obtain valid signature
- Can detect and remove a kernel rootkit only when booting another clean OS

Bootkits

- Malware can compromise the boot process of a computer
- Rootkits that modify the bootloader are called *bootkits*
- Bootkits are typically installed in the MBR of the hard drive
- Bootkits can make sure to re-infect a computer at each reboot

Firmware malware

- So far, malware was in software (user space, kernel space, boot loader)
- How about hardware malware?

Firmware malware

- So far, malware was in software (user space, kernel space, boot loader)
- How about firmware malware?

Firmware malware

- So far, malware was in software (user space, kernel space, boot loader)
- How about firmware malware?
- Close to impossible to detect (or remove) by malware scanners
- Survives full re-installation of the operating system

Firmware malware

- So far, malware was in software (user space, kernel space, boot loader)
- How about firmware malware?
- Close to impossible to detect (or remove) by malware scanners
- Survives full re-installation of the operating system
- Example 1: badBIOS (malware infecting the BIOS)

Firmware malware

- So far, malware was in software (user space, kernel space, boot loader)
- How about firmware malware?
- Close to impossible to detect (or remove) by malware scanners
- Survives full re-installation of the operating system
- Example 1: badBIOS (malware infecting the BIOS)
- Example 2: badUSB (malicious USB device firmware)

Firmware malware

- So far, malware was in software (user space, kernel space, boot loader)
- How about firmware malware?
- Close to impossible to detect (or remove) by malware scanners
- Survives full re-installation of the operating system
- Example 1: badBIOS (malware infecting the BIOS)
- Example 2: badUSB (malicious USB device firmware)
- Example 3: IRATEMONK (NSA malware to infect harddrive firmware)
  http://leaksource.files.wordpress.com/2013/12/nsa-ant-iratemonk.jpg

Firmware malware

- So far, malware was in software (user space, kernel space, boot loader)
- How about firmware malware?
- Close to impossible to detect (or remove) by malware scanners
- Survives full re-installation of the operating system
- Example 1: badBIOS (malware infecting the BIOS)
- Example 2: badUSB (malicious USB device firmware)
- Example 3: IRATEMONK (NSA malware to infect harddrive firmware)
  http://leaksource.files.wordpress.com/2013/12/nsa-ant-iratemonk.jpg
- Impressive piece of work on firmware malware: DAGGER
  - Infects computer through Intel's Advanced Management Technology (AMT)
  - Includes keylogger, sends all keystrokes over the network
  - Operating system cannot see any of this
  - For a great talk, see
    http://www.youtube.com/watch?v=Ck8bIjAUJgE

Malware functionality

- Can also classify malware by its damage routines:

Malware functionality

- Can also classify malware by its damage routines:
- Many worms and viruses turn infected computers into *botnet zombie hosts*
- Primary target: obtain network for DOS attacks and spamming

Malware functionality

- Can also classify malware by its damage routines:
- Many worms and viruses turn infected computers into *botnet zombie hosts*
- Primary target: obtain network for DOS attacks and spamming
- *Ransomware* encrypts part of the harddrive, requests money for decryption key

Malware functionality

- Can also classify malware by its damage routines:
- Many worms and viruses turn infected computers into *botnet zombie hosts*
- Primary target: obtain network for DOS attacks and spamming
- *Ransomware* encrypts part of the harddrive, requests money for decryption key
- *Spyware* is used to exfiltrate information (e.g., banking data)

Malware functionality

- Can also classify malware by its damage routines:
- Many worms and viruses turn infected computers into *botnet zombie hosts*
- Primary target: obtain network for DOS attacks and spamming
- *Ransomware* encrypts part of the harddrive, requests money for decryption key
- *Spyware* is used to exfiltrate information (e.g., banking data)
- *Dialer* were (maybe still are?) used to dial expensive numbers from the modem

Malware functionality

- Can also classify malware by its damage routines:
- Many worms and viruses turn infected computers into *botnet zombie hosts*
- Primary target: obtain network for DOS attacks and spamming
- *Ransomware* encrypts part of the harddrive, requests money for decryption key
- *Spyware* is used to exfiltrate information (e.g., banking data)
- *Dialer* were (maybe still are?) used to dial expensive numbers from the modem
- Targeted malware can have very specific damage routines
- Example: Stuxnet sabotaged the Iranian nuclear program

Malware functionality

- Can also classify malware by its damage routines:
- Many worms and viruses turn infected computers into *botnet zombie hosts*
- Primary target: obtain network for DOS attacks and spamming
- *Ransomware* encrypts part of the harddrive, requests money for decryption key
- *Spyware* is used to exfiltrate information (e.g., banking data)
- *Dialer* were (maybe still are?) used to dial expensive numbers from the modem
- Targeted malware can have very specific damage routines
- Example: Stuxnet sabotaged the Iranian nuclear program
- Finally, some malware just destroys data (digital vandalism)

Malware detection

- Idea: look at incoming files before they are stored on the hard drive
- Scan for malware, stop if malware detected
- Alternative: full scan of all files on the hard drive

Malware detection

- Idea: look at incoming files before they are stored on the hard drive
- Scan for malware, stop if malware detected
- Alternative: full scan of all files on the hard drive
- Important malware-scanner characteristics:
  - *Detection rate*: percentage of malware that is detected
  - Undetected malware is called *false negatives*

Malware detection

- Idea: look at incoming files before they are stored on the hard drive
- Scan for malware, stop if malware detected
- Alternative: full scan of all files on the hard drive
- Important malware-scanner characteristics:
  - *Detection rate*: percentage of malware that is detected
  - Undetected malware is called *false negatives*
  - Files that are incorrectly classified as malware are *false positives*
  - Typical requirement: no false positives!

Malware detection

- Idea: look at incoming files before they are stored on the hard drive
- Scan for malware, stop if malware detected
- Alternative: full scan of all files on the hard drive
- Important malware-scanner characteristics:
  - *Detection rate*: percentage of malware that is detected
  - Undetected malware is called *false negatives*
  - Files that are incorrectly classified as malware are *false positives*
  - Typical requirement: no false positives!
- Mainly two techniques to detect malware:
  - *Signature-based detection*: (look for known patterns in files)
  - *Heuristic detection:* Analyse behavior and make decision

Signature-based malware detection

- Signature-based malware detection only detects *known* malware
- Essential requirement: update the signature database daily
- Still cannot detect zero-day (next-generation) malware

Signature-based malware detection

- Signature-based malware detection only detects *known* malware
- Essential requirement: update the signature database daily
- Still cannot detect zero-day (next-generation) malware
- Signatures can be as simple as a cryptographic hash
- Typically look for certain code sequences (less susceptible to minor changes)

Signature-based malware detection

- Signature-based malware detection only detects *known* malware
- Essential requirement: update the signature database daily
- Still cannot detect zero-day (next-generation) malware
- Signatures can be as simple as a cryptographic hash
- Typically look for certain code sequences (less susceptible to minor changes)
- Generally powerful technique against known malware
- Used by all major anti-malware software

Code polymorphism

- Idea to defeat signature-based malware detection: *polymorphic code*
- Use automated engine to generate many versions of a virus
- All have the same functionality, but look different

Code polymorphism

- Idea to defeat signature-based malware detection: *polymorphic code*
- Use automated engine to generate many versions of a virus
- All have the same functionality, but look different
- In principle there is an infite number of ways to mutate a program and keep functionality
- Trivial example: insert NOP instructions

Code polymorphism

- Idea to defeat signature-based malware detection: *polymorphic code*
- Use automated engine to generate many versions of a virus
- All have the same functionality, but look different
- In principle there is an infite number of ways to mutate a program and keep functionality
- Trivial example: insert NOP instructions
- More advanced: permute independent instructions

Code polymorphism

- Idea to defeat signature-based malware detection: *polymorphic code*
- Use automated engine to generate many versions of a virus
- All have the same functionality, but look different
- In principle there is an infite number of ways to mutate a program and keep functionality
- Trivial example: insert NOP instructions
- More advanced: permute independent instructions
- Can even check that polymorphic versions are not detected
- Use online tools, e.g., `https://www.virustotal.com/en/`

Code polymorphism

- Idea to defeat signature-based malware detection: *polymorphic code*
- Use automated engine to generate many versions of a virus
- All have the same functionality, but look different
- In principle there is an infite number of ways to mutate a program and keep functionality
- Trivial example: insert NOP instructions
- More advanced: permute independent instructions
- Can even check that polymorphic versions are not detected
- Use online tools, e.g., `https://www.virustotal.com/en/`
- More advanced: self-mutating code (metamorphism)
- Virus that prints mutated copies of itself

Packers

- Other technique to evade malware detection: *packers*
- Idea: modify actual malware, use packer to "unpack" true malware in memory
- Packing can be simple XOR or bit-flipping or advanced encryption with AES
- Can even use multiple layers of packing

Packers

- Other technique to evade malware detection: *packers*
- Idea: modify actual malware, use packer to "unpack" true malware in memory
- Packing can be simple XOR or bit-flipping or advanced encryption with AES
- Can even use multiple layers of packing
- Can also unpack (decrypt) blockwise, such that full malware is never in memory

Packers

- Other technique to evade malware detection: *packers*
- Idea: modify actual malware, use packer to "unpack" true malware in memory
- Packing can be simple XOR or bit-flipping or advanced encryption with AES
- Can even use multiple layers of packing
- Can also unpack (decrypt) blockwise, such that full malware is never in memory
- Essentially two ways to detect packed malware:
  - Static detection: Try known packers on the payload
  - Dynamic detection: Run the malware (including unpacking routine) itself in a safe environment (sandbox)

Moving to the GPU

- Usually malware (and the packer) runs on the CPU
- Idea to hide from scanners: use the graphics processing unit (GPU) for unpacking
- Proof-of-concept presented by Vasiliadis, Polychronakis, and Ioannidis in 2010: "GPU assisted malware"

Moving to the GPU

- Usually malware (and the packer) runs on the CPU
- Idea to hide from scanners: use the graphics processing unit (GPU) for unpacking
- Proof-of-concept presented by Vasiliadis, Polychronakis, and Ioannidis in 2010: "GPU assisted malware"
- Problem for static detection:
  - Malware can use computational power of the GPU for unpacking
  - Trying to unpack on the CPU incurs significant slowdown

Moving to the GPU

- Usually malware (and the packer) runs on the CPU
- Idea to hide from scanners: use the graphics processing unit (GPU) for unpacking
- Proof-of-concept presented by Vasiliadis, Polychronakis, and Ioannidis in 2010: "GPU assisted malware"
- Problem for static detection:
  - Malware can use computational power of the GPU for unpacking
  - Trying to unpack on the CPU incurs significant slowdown
- Problem for dynamic detection:
  - Sandboxes don't support GPU binaries
  - Cannot run the malware in a safe environment

Moving to the GPU

- Usually malware (and the packer) runs on the CPU
- Idea to hide from scanners: use the graphics processing unit (GPU) for unpacking
- Proof-of-concept presented by Vasiliadis, Polychronakis, and Ioannidis in 2010: "GPU assisted malware"
- Problem for static detection:
  - Malware can use computational power of the GPU for unpacking
  - Trying to unpack on the CPU incurs significant slowdown
- Problem for dynamic detection:
  - Sandboxes don't support GPU binaries
  - Cannot run the malware in a safe environment
- Obviously, the GPU can also be used for malware detection (signature matching)
- Seamans and Alexander described GPU extension to ClamAV in 2007
- Speedup of signature detection on Nvidia GTX 7800 compared to 3-GHz Pentium 4:
  - 27× for 0% match rate
  - 17× for 1% match rate
  - 11× for 50% match rate

Heuristic malware detection

- Approach to detect unknown (variants of) malware: heuristics
- Simple case: use wildcards in signatures
- Advanced case: run the malware in a safe environment (virtual machine, sandbox), study behavior

Heuristic malware detection

- Approach to detect unknown (variants of) malware: heuristics
- Simple case: use wildcards in signatures
- Advanced case: run the malware in a safe environment (virtual machine, sandbox), study behavior
- Heuristic analysis relies on experience
- Good at detecting malware with behavior that "has been seen before"

Heuristic malware detection

- Approach to detect unknown (variants of) malware: heuristics
- Simple case: use wildcards in signatures
- Advanced case: run the malware in a safe environment (virtual machine, sandbox), study behavior
- Heuristic analysis relies on experience
- Good at detecting malware with behavior that "has been seen before"
- Typically not good at detecting really new malware
- Certainly not *reliable* at detecting new malware

AV can't hurt, or can it?

- Common security recommendation for end users: "Use a malware scanner (AV) and keep it up to date"
- "Wisdom" behind this recommendation: AV certainly makes security better (even if it doesn't detect everything)

AV can't hurt, or can it?

- Common security recommendation for end users: "Use a malware scanner (AV) and keep it up to date"
- "Wisdom" behind this recommendation: AV certainly makes security better (even if it doesn't detect everything)
- Multiple problems with this wisdom:
  1. AV software can seriously degrade system performance

AV can't hurt, or can it?

- Common security recommendation for end users: "Use a malware scanner (AV) and keep it up to date"
- "Wisdom" behind this recommendation: AV certainly makes security better (even if it doesn't detect everything)
- Multiple problems with this wisdom:
  1. AV software can seriously degrade system performance
  2. False positives can break system functionality

AV can't hurt, or can it?

- Common security recommendation for end users: "Use a malware scanner (AV) and keep it up to date"
- "Wisdom" behind this recommendation: AV certainly makes security better (even if it doesn't detect everything)
- Multiple problems with this wisdom:
  1. AV software can seriously degrade system performance
  2. False positives can break system functionality
  3. AV software is highly trusted (needs privileged access), but not necessarily trustworthy

AV can't hurt, or can it?

- Common security recommendation for end users: "Use a malware scanner (AV) and keep it up to date"
- "Wisdom" behind this recommendation: AV certainly makes security better (even if it doesn't detect everything)
- Multiple problems with this wisdom:
  1. AV software can seriously degrade system performance
  2. False positives can break system functionality
  3. AV software is highly trusted (needs privileged access), but not necessarily trustworthy
  4. Users may *feel* secure and behave less careful

AV can't hurt, or can it?

- Common security recommendation for end users: "Use a malware scanner (AV) and keep it up to date"
- "Wisdom" behind this recommendation: AV certainly makes security better (even if it doesn't detect everything)
- Multiple problems with this wisdom:
  1. AV software can seriously degrade system performance
  2. False positives can break system functionality
  3. AV software is highly trusted (needs privileged access), but not necessarily trustworthy
  4. Users may *feel* secure and behave less careful
  5. AV software can actively degrade security (e.g. Kaspersky):

AV can't hurt, or can it?

- Common security recommendation for end users: "Use a malware scanner (AV) and keep it up to date"
- "Wisdom" behind this recommendation: AV certainly makes security better (even if it doesn't detect everything)
- Multiple problems with this wisdom:
  1. AV software can seriously degrade system performance
  2. False positives can break system functionality
  3. AV software is highly trusted (needs privileged access), but not necessarily trustworthy
  4. Users may *feel* secure and behave less careful
  5. AV software can actively degrade security (e.g. Kaspersky):
     - Kaspersky has man-in-the-middle functionality for SSL connections
     - Kaspersky still speaks SSL 3.0 (although the browser may have it disabled)
     - SSL 3.0 is vulnerable to the POODLE attack

Zip bombs

- Malware scanners (AV) needs to unpack zipped files
- Unpacked copy needs to sit somewhere in memory or on disk

Zip bombs

- Malware scanners (AV) needs to unpack zipped files
- Unpacked copy needs to sit somewhere in memory or on disk
- Can use this as attack against AV:
  - Create small zip file, which expands to huge unpacked data
  - Can also use multiple levels of zipping

Zip bombs

- Malware scanners (AV) needs to unpack zipped files
- Unpacked copy needs to sit somewhere in memory or on disk
- Can use this as attack against AV:
  - Create small zip file, which expands to huge unpacked data
  - Can also use multiple levels of zipping
- Famous example: 42.zip
  - Packed size: 42 KB
  - Fully unpacked (after 5 levels of unzip): 4.5 PB
  - Expansion factor of $>$100,000,000,000

Zip bombs

- Malware scanners (AV) needs to unpack zipped files
- Unpacked copy needs to sit somewhere in memory or on disk
- Can use this as attack against AV:
  - Create small zip file, which expands to huge unpacked data
  - Can also use multiple levels of zipping
- Famous example: 42.zip
  - Packed size: 42 KB
  - Fully unpacked (after 5 levels of unzip): 4.5 PB
  - Expansion factor of $>$100,000,000,000
- Recall self-replicating code, how about a self replicating zip?
- Idea: create a zip file that contains itself
- Virus scanners will keep unpacking forever

Zip bombs

- Malware scanners (AV) needs to unpack zipped files
- Unpacked copy needs to sit somewhere in memory or on disk
- Can use this as attack against AV:
  - Create small zip file, which expands to huge unpacked data
  - Can also use multiple levels of zipping
- Famous example: 42.zip
  - Packed size: 42 KB
  - Fully unpacked (after 5 levels of unzip): 4.5 PB
  - Expansion factor of >100,000,000,000
- Recall self-replicating code, how about a self replicating zip?
- Idea: create a zip file that contains itself
- Virus scanners will keep unpacking forever
- This exists, for details see http://research.swtch.com/zip
- Not restricted to zip, also works with gzip

HIDS

- HIDS goes beyond malware scanning (although there may be some overlap)
- Typically register certain resources with the IDS, those resources are monitored
- Examples of resources: system files, Windows registry entries, network ports

HIDS

- HIDS goes beyond malware scanning (although there may be some overlap)
- Typically register certain resources with the IDS, those resources are monitored
- Examples of resources: system files, Windows registry entries, network ports
- Idea: remember state of resource, detect modifications
- Typically store hash values of resources
- Crucial to protect the table of hashes!

HIDS

- HIDS goes beyond malware scanning (although there may be some overlap)
- Typically register certain resources with the IDS, those resources are monitored
- Examples of resources: system files, Windows registry entries, network ports
- Idea: remember state of resource, detect modifications
- Typically store hash values of resources
- Crucial to protect the table of hashes!
- Additionally, analyze log files (e.g., `/var/log/syslog`)
- For log-file analysis, two possibilities:
  - Signature-based intrusion detection
  - Anomaly-based intrustion detection

HIDS

- HIDS goes beyond malware scanning (although there may be some overlap)
- Typically register certain resources with the IDS, those resources are monitored
- Examples of resources: system files, Windows registry entries, network ports
- Idea: remember state of resource, detect modifications
- Typically store hash values of resources
- Crucial to protect the table of hashes!
- Additionally, analyze log files (e.g., `/var/log/syslog`)
- For log-file analysis, two possibilities:
  - Signature-based intrusion detection
  - Anomaly-based intrustion detection
- Problem of signature-based IDS: same as with AV

HIDS

- HIDS goes beyond malware scanning (although there may be some overlap)
- Typically register certain resources with the IDS, those resources are monitored
- Examples of resources: system files, Windows registry entries, network ports
- Idea: remember state of resource, detect modifications
- Typically store hash values of resources
- Crucial to protect the table of hashes!
- Additionally, analyze log files (e.g., `/var/log/syslog`)
- For log-file analysis, two possibilities:
  - Signature-based intrusion detection
  - Anomaly-based intrustion detection
- Problem of signature-based IDS: same as with AV
- Problem of anomaly-based IDS: hard to obtain good detection rate at low false-positive rate in highly dynamic systems

Recover after intrusion

- Easy situation: download a file from the Internet, AV complains.
  $\Rightarrow$ Don't run/open file, but stop download (or delete file).

Recover after intrusion

- Easy situation: download a file from the Internet, AV complains.
  $\Rightarrow$ Don't run/open file, but stop download (or delete file).
- Hard situation: AV complains about *old* files (or IDS reports intrusion)

Recover after intrusion

- Easy situation: download a file from the Internet, AV complains.
  $\Rightarrow$ Don't run/open file, but stop download (or delete file).
- Hard situation: AV complains about *old* files (or IDS reports intrusion)
- AV software typically offers to "remove the virus/worm/trojan"
- Question: Is that enough?

Recover after intrusion

- Easy situation: download a file from the Internet, AV complains.
  $\Rightarrow$ Don't run/open file, but stop download (or delete file).
- Hard situation: AV complains about *old* files (or IDS reports intrusion)
- AV software typically offers to "remove the virus/worm/trojan"
- Question: Is that enough?
- There is only one responsible answer: **No.**

Recover after intrusion

- Easy situation: download a file from the Internet, AV complains.
  $\Rightarrow$ Don't run/open file, but stop download (or delete file).
- Hard situation: AV complains about *old* files (or IDS reports intrusion)
- AV software typically offers to "remove the virus/worm/trojan"
- Question: Is that enough?
- There is only one responsible answer: **No.**
- Once a system has been compromised, you don't know what else is broken
- Only reasonable recovery from intrusion:
  - Isolate the system (to prevent further damage)
  - Analyze what what compromised and how (forensics)
  - Restore to a clean state (reinstall, restore clean data backup)