OS Security

Malware

Radboud University, Nijmegen, The Netherlands

Winter 2016/2017

A short recap

- Processes access memory through virtual addresses
- Mapping between virtual and physical addresses is (typically) done in hardware, but managed through the OS
- OS separates memory space of different processors
- Memory attack: Write shellcode to buffer, overflow buffer, overwrite return address with pointer to shell code
  - Countermeasure: NX (or $W \oplus X$)
- Advanced attack: return to libc, generalization: ROP
  - Countermeasure: Address Space Layout Randomization

Ad-hoc solutions for better OS security

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

Ad-hoc solutions for better OS security

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

Ad-hoc solutions for better OS security

- Completely re-designing an OS is expensive
- More feasible: Add-on security for existing OS
- Multiple techniques:
  - Memory protection (NX bit) and ASLR (last week)
  - Detect (or prevent) malware and intrusions (this lecture)

Ad-hoc solutions for better OS security

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

Ad-hoc solutions for better OS security

- Completely re-designing an OS is expensive
- More feasible: Add-on security for existing OS
- Multiple techniques:
  - Memory protection (NX bit) and ASLR (last week)
  - Detect (or prevent) malware and intrusions (this lecture)
  - Add mandatory access control (next week)
  - Compartmentalization and virtualization (Dec 5 & 12)

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 (or payload) 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 (or payload) 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 (or payload) 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 (or payload) 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)
  - 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, capture 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, capture screenshots, fetch upgrades remotely
  - Communication from the trojan was encrypted with AES in ECB mode
  - Communication to the trojan was unencrypted!

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, capture screenshots, fetch upgrades remotely
  - Communication from the trojan was encrypted with AES in ECB mode
  - Communication to the trojan was unencrypted!
  - Trojan was nicknamed *R2D2* because the string "C3PO-r2d2-POE" was found in its code

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
  - Can communicate via *covert channels*

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
  - Can communicate via *covert channels*
    - Any information flow not considered by the reference monitor is a covert channel.

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
  - Can communicate via *covert channels*
    - Any information flow not considered by the reference monitor is a covert channel.
    - Examples: Existence of a file, file access permissions, CPU usage, temperature sensor

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
  - Can communicate via *covert channels*
    - Any information flow not considered by the reference monitor is a covert channel.
    - Examples: Existence of a file, file access permissions, CPU usage, temperature sensor
    - (i) Timing Channels (e.g. CPU load)
    - (ii) Storage Channels (e.g. existence of files)

Rootkits (continued)

- Possible countermeasure: cryptographically sign all kernel modules and drivers

Rootkits (continued)

- Possible countermeasure: cryptographically sign all kernel modules and drivers
- This went horribly wrong with Flame in 2012

Rootkits (continued)

- Possible countermeasure: cryptographically sign all kernel modules and drivers
- This went horribly wrong with Flame in 2012
  - Flame tried to blend in with legitimate Microsoft applications by cloaking itself with an older cryptography algorithm that Microsoft used to digitally sign programs
  - Weaknesses in the MD5 hash function allowed malware to obtain valid signature

Rootkits (continued)

- Possible countermeasure: cryptographically sign all kernel modules and drivers
- This went horribly wrong with Flame in 2012
  - Flame tried to blend in with legitimate Microsoft applications by cloaking itself with an older cryptography algorithm that Microsoft used to digitally sign programs
  - 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 firmware 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 used to dial expensive numbers from the modem (not common anymore)

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 used to dial expensive numbers from the modem (not common anymore)
- 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 used to dial expensive numbers from the modem (not common anymore)
- 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
  - *Behavior-based 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 or sequence of system calls
- 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 or sequence of system calls
- 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
- Useful tools, e.g., VirusTotal(https://www.virustotal.com/en/), IDA Pro (https://www.hex-rays.com/products/ida/index.shtml)

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
- Useful tools, e.g., VirusTotal(https://www.virustotal.com/en/), IDA Pro (https://www.hex-rays.com/products/ida/index.shtml)
- More advanced: self-mutating code (metamorphism)
- Virus that prints mutated copies of itself

Packers

- Other technique to evade malware detection: *packers*

Packers

- Other technique to evade malware detection: *packers*
- Packer: A piece of software that takes the original malware and compresses it, thus making all the original code and data unreadable.

Packers

- Other technique to evade malware detection: *packers*
- Packer: A piece of software that takes the original malware and compresses it, thus making all the original code and data unreadable.
  - At runtime, a wrapper program will take the packed program and decompress it in memory, revealing the program's original code.

Packers

- Other technique to evade malware detection: *packers*
- Packer: A piece of software that takes the original malware and compresses it, thus making all the original code and data unreadable.
  - At runtime, a wrapper program will take the packed program and decompress it in memory, revealing the program's original code.
- Packing can be simple XOR or bit-flipping or advanced encryption with AES

Packers

- Other technique to evade malware detection: *packers*
- Packer: A piece of software that takes the original malware and compresses it, thus making all the original code and data unreadable.
  - At runtime, a wrapper program will take the packed program and decompress it in memory, revealing the program's original code.
- 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*
- Packer: A piece of software that takes the original malware and compresses it, thus making all the original code and data unreadable.
  - At runtime, a wrapper program will take the packed program and decompress it in memory, revealing the program's original code.
- 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)

Packers

- Other technique to evade malware detection: *packers*
- Packer: A piece of software that takes the original malware and compresses it, thus making all the original code and data unreadable.
  - At runtime, a wrapper program will take the packed program and decompress it in memory, revealing the program's original code.
- 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)
- An interesting research area

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 causes 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 causes 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 causes 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

Behavior-based malware detection

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

Behavior-based malware detection

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

Behavior-based malware detection

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

Antivirus software (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)

Antivirus software (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

Antivirus software (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

Antivirus software (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

Antivirus software (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 carefully

Antivirus software (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 carefully
  5. AV software can actively degrade security (e.g. Kaspersky):

Antivirus software (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 carefully
  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

Antivirus software (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 carefully
  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
     - POODLE - Padding Oracle On Downgraded Legacy Encryption
     - MiTM attack exploiting web browser vulnerability when web browsers and servers will downgrade to SSL 3.0 if there are problems negotiating a TLS session

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

Part II

Smartphone Malware

Evolution of Malware: From PC to Smartphone

Evolution of Malware: From PC to Smartphone

- Larger attack surface for malware authors; easy-to-deploy attacks; many forms of attack vectors

Evolution of Malware: From PC to Smartphone

- Larger attack surface for malware authors; easy-to-deploy attacks; many forms of attack vectors
- Motivation: 'low risk, high reward'

Evolution of Malware: From PC to Smartphone

- Larger attack surface for malware authors; easy-to-deploy attacks; many forms of attack vectors
- Motivation: 'low risk, high reward'
  - Various app markets: official (e.g. Google Play) and non-official (e.g. Pandaapp)
  - Decentralized: anyone can become an app developer; no proper vetting of new apps

Early days of smartphone malware

Early days of smartphone malware

- Back in 2004, a group known as **29A** released `Cabir` - a malware (worm) for Symbian
  - Propagate via Bluetooth
  - Bluetooth was the most used technology to transfer information between 2 devices at the time

Early days of smartphone malware

- Back in 2004, a group known as **29A** released `Cabir` - a malware (worm) for Symbian
  - Propagate via Bluetooth
  - Bluetooth was the most used technology to transfer information between 2 devices at the time
  - Countermeasure: simply turn Bluetooth off or switch it to the "invisible" mode

Early days of smartphone malware

- Back in 2004, a group known as **29A** released `Cabir` - a malware (worm) for Symbian
  - Propagate via Bluetooth
  - Bluetooth was the most used technology to transfer information between 2 devices at the time
  - Countermeasure: simply turn Bluetooth off or switch it to the "invisible" mode
- Trojan, `Qdial`, targeting Symbian users, was released in same year
  - Malware sent text messages to premium rate services, for which the handset owner would be charged, thus making an income for the malware author.

Early days of smartphone malware

- Back in 2004, a group known as **29A** released `Cabir` - a malware (worm) for Symbian
  - Propagate via Bluetooth
  - Bluetooth was the most used technology to transfer information between 2 devices at the time
  - Countermeasure: simply turn Bluetooth off or switch it to the "invisible" mode
- Trojan, `Qdial`, targeting Symbian users, was released in same year
  - Malware sent text messages to premium rate services, for which the handset owner would be charged, thus making an income for the malware author.
- In 2005, a variant of `Cabir` was released - `Pbstealer`
  - It copied all the information from an infected device's address book and attempted to transmit it to any Bluetooth-enabled device within range.

Early days of smartphone malware

- Back in 2004, a group known as **29A** released `Cabir` - a malware (worm) for Symbian
  - Propagate via Bluetooth
  - Bluetooth was the most used technology to transfer information between 2 devices at the time
  - Countermeasure: simply turn Bluetooth off or switch it to the "invisible" mode
- Trojan, `Qdial`, targeting Symbian users, was released in same year
  - Malware sent text messages to premium rate services, for which the handset owner would be charged, thus making an income for the malware author.
- In 2005, a variant of `Cabir` was released - `Pbstealer`
  - It copied all the information from an infected device's address book and attempted to transmit it to any Bluetooth-enabled device within range.
  - Malware included the string: *".:: Good artist copy, Great artist steal ::."*

Current state of smartphone malware

- All major smartphone platforms have been infected

Current state of smartphone malware

- All major smartphone platforms have been infected
  - iOS: `WireLurker` (2014), can install malicious third-party applications to an iOS device through an infected Mac via a USB connection

Current state of smartphone malware

- All major smartphone platforms have been infected
  - iOS: `WireLurker` (2014), can install malicious third-party applications to an iOS device through an infected Mac via a USB connection
  - Windows Phone: Proof-of-concept for Windows Phone 8 presented at MalCon 2013; `FinSpy Mobile` spyware (2013)

Current state of smartphone malware

- All major smartphone platforms have been infected
  - iOS: `WireLurker` (2014), can install malicious third-party applications to an iOS device through an infected Mac via a USB connection
  - Windows Phone: Proof-of-concept for Windows Phone 8 presented at MalCon 2013; `FinSpy Mobile` spyware (2013)
  - Blackberry: Trojans use a technique referred to as 'BackStab'; steal unencrypted backups of phones from computers; does not require higher-level privileges or root access to the phone or computer

Current state of smartphone malware

- All major smartphone platforms have been infected
  - iOS: `WireLurker` (2014), can install malicious third-party applications to an iOS device through an infected Mac via a USB connection
  - Windows Phone: Proof-of-concept for Windows Phone 8 presented at MalCon 2013; `FinSpy Mobile` spyware (2013)
  - Blackberry: Trojans use a technique referred to as 'BackStab'; steal unencrypted backups of phones from computers; does not require higher-level privileges or root access to the phone or computer
- Android OS - most infected platform to date

Popular Android Malware

- First proof-of-concept malware released in 2008.
  - Causes the phone to accept all incoming calls
  - Turns off the radio, preventing outgoing/incoming calls
  - Causes the phone to end all calls
  - Gathers sensitive information and sends it to the attacker

Popular Android Malware

- First proof-of-concept malware released in 2008.
  - Causes the phone to accept all incoming calls
  - Turns off the radio, preventing outgoing/incoming calls
  - Causes the phone to end all calls
  - Gathers sensitive information and sends it to the attacker
- `Mobile Spy` spyware, 2009
  - Monitored infected device via web browser, phone calls, text messages, photos, videos, GPS locations

Popular Android Malware

- First proof-of-concept malware released in 2008.
  - Causes the phone to accept all incoming calls
  - Turns off the radio, preventing outgoing/incoming calls
  - Causes the phone to end all calls
  - Gathers sensitive information and sends it to the attacker
- `Mobile Spy` spyware, 2009
  - Monitored infected device via web browser, phone calls, text messages, photos, videos, GPS locations
  - Ran in 'stealth mode', no visible icon

Popular Android Malware

- First proof-of-concept malware released in 2008.
  - Causes the phone to accept all incoming calls
  - Turns off the radio, preventing outgoing/incoming calls
  - Causes the phone to end all calls
  - Gathers sensitive information and sends it to the attacker
- `Mobile Spy` spyware, 2009
  - Monitored infected device via web browser, phone calls, text messages, photos, videos, GPS locations
  - Ran in 'stealth mode', no visible icon
- `DroidKungFu`, capable of root-level access on vulnerable Android devices and evade the detection of security software by encrypting its exploits using AES.

Popular Android Malware

- First proof-of-concept malware released in 2008.
  - Causes the phone to accept all incoming calls
  - Turns off the radio, preventing outgoing/incoming calls
  - Causes the phone to end all calls
  - Gathers sensitive information and sends it to the attacker
- `Mobile Spy` spyware, 2009
  - Monitored infected device via web browser, phone calls, text messages, photos, videos, GPS locations
  - Ran in 'stealth mode', no visible icon
- `DroidKungFu`, capable of root-level access on vulnerable Android devices and evade the detection of security software by encrypting its exploits using AES.
  - One of the exploits used was the `RageAgainstTheCage` (RATC) exploit.
  - Also known as *adb setuid exhaustion attack*
  - A race condition between RATC and adb-server

Popular Android Malware

- First proof-of-concept malware released in 2008.
  - Causes the phone to accept all incoming calls
  - Turns off the radio, preventing outgoing/incoming calls
  - Causes the phone to end all calls
  - Gathers sensitive information and sends it to the attacker
- `Mobile Spy` spyware, 2009
  - Monitored infected device via web browser, phone calls, text messages, photos, videos, GPS locations
  - Ran in 'stealth mode', no visible icon
- `DroidKungFu`, capable of root-level access on vulnerable Android devices and evade the detection of security software by encrypting its exploits using AES.
  - One of the exploits used was the `RageAgainstTheCage` (RATC) exploit.
  - Also known as *adb setuid exhaustion attack*
  - A race condition between RATC and adb-server
  - See https://thesnkchrmr.wordpress.com/2011/03/24/rageagainstthecage/ for more details about the exploit and its source code

Rootkits & Bootkits

- One of the first Android rootkit was presented at DEF CON 18 (2010)

Rootkits & Bootkits

- One of the first Android rootkit was presented at DEF CON 18 (2010)
- Rootkit was used to track location of smartphone's owner, read SMS and redirect calls

Rootkits & Bootkits

- One of the first Android rootkit was presented at DEF CON 18 (2010)
- Rootkit was used to track location of smartphone's owner, read SMS and redirect calls
- A demo of a clickjacking rootkit targeting Android 4.0, https://www.youtube.com/watch?v=RxpMPrqnxC0

Rootkits & Bootkits

- One of the first Android rootkit was presented at DEF CON 18 (2010)
- Rootkit was used to track location of smartphone's owner, read SMS and redirect calls
- A demo of a clickjacking rootkit targeting Android 4.0, https://www.youtube.com/watch?v=RxpMPrqnxC0
- Bootkit, `Android.Oldboot` (2014) has the capability of reinstalling itself even after all of its working components have been deleted. Primary targets were rooted Android devices.

Bitcoin Mining malware

- In 2014, several malicious apps found on Google Play store were used in a large-scale crypto currency mining operation

Bitcoin Mining malware

- In 2014, several malicious apps found on Google Play store were used in a large-scale crypto currency mining operation
- Contained a hidden crypto miner that stealthily exploit users' device for computational resources

Bitcoin Mining malware

- In 2014, several malicious apps found on Google Play store were used in a large-scale crypto currency mining operation
- Contained a hidden crypto miner that stealthily exploit users' device for computational resources
- Malware was deployed through Wallpaper apps, with more than 500 downloads

Tools to analyze Android Malware

- Mobile Malware can be analyzed in 2 ways: *statically* and *dynamically*

Tools to analyze Android Malware

- Mobile Malware can be analyzed in 2 ways: *statically* and *dynamically*
- Static Analysis: Analyze suspicious app through reverse-engineering

Tools to analyze Android Malware

- Mobile Malware can be analyzed in 2 ways: *statically* and *dynamically*
- Static Analysis: Analyze suspicious app through reverse-engineering
- Dynamic Analysis: Execute the suspicious app in a controlled environment and monitor its behaviors

Tools to analyze Android Malware

- Mobile Malware can be analyzed in 2 ways: *statically* and *dynamically*
- Static Analysis: Analyze suspicious app through reverse-engineering
- Dynamic Analysis: Execute the suspicious app in a controlled environment and monitor its behaviors
- Tools: IDA Pro, JD-Gui, Dex2Jar, Android SDK

Tools to analyze Android Malware

- Mobile Malware can be analyzed in 2 ways: *statically* and *dynamically*
- Static Analysis: Analyze suspicious app through reverse-engineering
- Dynamic Analysis: Execute the suspicious app in a controlled environment and monitor its behaviors
- Tools: IDA Pro, JD-Gui, Dex2Jar, Android SDK
- Countermeasures against Android malware:

Tools to analyze Android Malware

- Mobile Malware can be analyzed in 2 ways: *statically* and *dynamically*
- Static Analysis: Analyze suspicious app through reverse-engineering
- Dynamic Analysis: Execute the suspicious app in a controlled environment and monitor its behaviors
- Tools: IDA Pro, JD-Gui, Dex2Jar, Android SDK
- Countermeasures against Android malware:
  - There is no single solution!
  - Download apps from official markets only
  - Read permissions carefully before downloading and installing an app

Antivirus apps for Smartphones

- Useful as a first-line of defense
  - Can detect popular malware families and their variants

[1] https://vvdveen.com/publications/drammer.pdf
[2] https://arxiv.org/pdf/1611.03748v1.pdf

Antivirus apps for Smartphones

- Useful as a first-line of defense
  - Can detect popular malware families and their variants
  - But...

[1] https://vvdveen.com/publications/drammer.pdf
[2] https://arxiv.org/pdf/1611.03748v1.pdf

Antivirus apps for Smartphones

- Useful as a first-line of defense
  - Can detect popular malware families and their variants
  - But...
  - App developers have fine-grained access (mostly via permissions) to hardware and software resources of the device
  - As a result, many malicious attack go undetected by antivirus apps

[1] https://vvdveen.com/publications/drammer.pdf
[2] https://arxiv.org/pdf/1611.03748v1.pdf

Antivirus apps for Smartphones

- Useful as a first-line of defense
  - Can detect popular malware families and their variants
  - But...
  - App developers have fine-grained access (mostly via permissions) to hardware and software resources of the device
  - As a result, many malicious attack go undetected by antivirus apps
- DRAMMER[1] attack

[1] https://vvdveen.com/publications/drammer.pdf
[2] https://arxiv.org/pdf/1611.03748v1.pdf

Antivirus apps for Smartphones

- Useful as a first-line of defense
  - Can detect popular malware families and their variants
  - But...
  - App developers have fine-grained access (mostly via permissions) to hardware and software resources of the device
  - As a result, many malicious attack go undetected by antivirus apps
- DRAMMER[1] attack
  - Deterministic Rowhammer attack; allows attackers to manipulate data stored in memory chips, resulting in gaining root access of the device

[1] https://vvdveen.com/publications/drammer.pdf
[2] https://arxiv.org/pdf/1611.03748v1.pdf

Antivirus apps for Smartphones

- Useful as a first-line of defense
  - Can detect popular malware families and their variants
  - But...
  - App developers have fine-grained access (mostly via permissions) to hardware and software resources of the device
  - As a result, many malicious attack go undetected by antivirus apps
- DRAMMER[1] attack
  - Deterministic Rowhammer attack; allows attackers to manipulate data stored in memory chips, resulting in gaining root access of the device
- Side-channel attacks on smartphones
  - Apps often (unintentionally) leak sensitive information which can be exploited by malware

[1] https://vvdveen.com/publications/drammer.pdf
[2] https://arxiv.org/pdf/1611.03748v1.pdf

Antivirus apps for Smartphones

- Useful as a first-line of defense
  - Can detect popular malware families and their variants
  - But...
  - App developers have fine-grained access (mostly via permissions) to hardware and software resources of the device
  - As a result, many malicious attack go undetected by antivirus apps
- DRAMMER[1] attack
  - Deterministic Rowhammer attack; allows attackers to manipulate data stored in memory chips, resulting in gaining root access of the device
- Side-channel attacks on smartphones
  - Apps often (unintentionally) leak sensitive information which can be exploited by malware
  - Leaks via in-built sensors, such as accelerometers, GPS, motion sensors, etc..
  - See this paper[2] for more details

[1] https://vvdveen.com/publications/drammer.pdf
[2] https://arxiv.org/pdf/1611.03748v1.pdf

Intrusion Detection & Prevention

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

Intrusion Detection & Prevention

- Two kinds of intrusion detection systems (IDS):
  - Network-based IDS (NIDS)
  - Host-based IDS (HIDS)
    - Special kind of HIDS: antivirus software (AV)
    - AV is typically more generally anti-malware software (aka virus scanners, malware scanners)

Intrusion Detection & Prevention

- Two kinds of intrusion detection systems (IDS):
  - Network-based IDS (NIDS)
  - Host-based IDS (HIDS)
    - Special kind of HIDS: antivirus software (AV)
    - AV is typically more generally anti-malware software (aka virus scanners, malware scanners)
- 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)

Intrusion Detection & Prevention

- Two kinds of intrusion detection systems (IDS):
  - Network-based IDS (NIDS)
  - Host-based IDS (HIDS)
    - Special kind of HIDS: antivirus software (AV)
    - AV is typically more generally anti-malware software (aka virus scanners, malware scanners)
- 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 tool: SNORT (more on this later)

Network-based intrusion detection system

- NIDS monitors traffic on its network segment as a data source
- This is achieved by placing the network interface in *promiscuous mode* to capture all traffic that crosses its network segment

Network-based intrusion detection system

- NIDS monitors traffic on its network segment as a data source
- This is achieved by placing the network interface in *promiscuous mode* to capture all traffic that crosses its network segment
- Different detection methods:
  - Signature-based detection: Signatures are attack patterns predetermined and pre-configured. This detection method monitors the network traffic and compares it with the pre-configured signatures so as to find a match.

Network-based intrusion detection system

- NIDS monitors traffic on its network segment as a data source
- This is achieved by placing the network interface in *promiscuous mode* to capture all traffic that crosses its network segment
- Different detection methods:
  - Signature-based detection: Signatures are attack patterns predetermined and pre-configured. This detection method monitors the network traffic and compares it with the pre-configured signatures so as to find a match.
  - Anomaly-based detection: This method of detection creates a baseline on average network conditions. Once a baseline has been created, the system intermittently samples network traffic on the basis of statistical analysis and compares the sample to the created baseline.

Network-based intrusion detection system

- NIDS monitors traffic on its network segment as a data source
- This is achieved by placing the network interface in *promiscuous mode* to capture all traffic that crosses its network segment
- Different detection methods:
  - Signature-based detection: Signatures are attack patterns predetermined and pre-configured. This detection method monitors the network traffic and compares it with the pre-configured signatures so as to find a match.
  - Anomaly-based detection: This method of detection creates a baseline on average network conditions. Once a baseline has been created, the system intermittently samples network traffic on the basis of statistical analysis and compares the sample to the created baseline.
  - Protocol state analysis detection: This type of detection method identifies deviations of protocol states by comparing observed events with predefined profiles

Signature-based detection

- Only works for known attacks

Signature-based detection

- Only works for known attacks
- Packets can be matched against three different types of signatures:

Signature-based detection

- Only works for known attacks
- Packets can be matched against three different types of signatures:
  - String signatures: look for a text string that indicates a possible attack. For example: `"cat "+ +" > /.rhosts"` might cause a UNIX system to become extremely vulnerable to network attack

Signature-based detection

- Only works for known attacks
- Packets can be matched against three different types of signatures:
  - String signatures: look for a text string that indicates a possible attack. For example: `"cat "+ +" > /.rhosts"` might cause a UNIX system to become extremely vulnerable to network attack
  - Port signatures: monitor connection attempts to well-known, frequently attacked ports. Examples of these ports include telnet (TCP port 23), FTP (TCP port 21/20), SUNRPC (TCP/UDP port 111), and IMAP (TCP port 143)

Signature-based detection

- Only works for known attacks
- Packets can be matched against three different types of signatures:
  - String signatures: look for a text string that indicates a possible attack. For example: `"cat "+ +" > /.rhosts"` might cause a UNIX system to become extremely vulnerable to network attack
  - Port signatures: monitor connection attempts to well-known, frequently attacked ports. Examples of these ports include telnet (TCP port 23), FTP (TCP port 21/20), SUNRPC (TCP/UDP port 111), and IMAP (TCP port 143)
  - Header signatures: watch for suspicious combinations in packet headers. For example:s a TCP packet with both the SYN and FIN flags set, signifying that the requester wishes to start and stop a connection at the same time

Host-based intrusion detection system

- 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

Host-based intrusion detection system

- 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!

Host-based intrusion detection system

- 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
  - Behavior-based intrusion detection

Host-based intrusion detection system

- 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
  - Behavior-based intrusion detection
- Problem of signature-based IDS: same as with AV

Host-based intrusion detection system

- 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
  - Behavior-based intrusion detection
- Problem of signature-based IDS: same as with AV
- Problem of behavior-based IDS: hard to obtain good detection rate at low false-positive rate in highly dynamic systems

SNORT

- Can be used for network intrusion detection and prevention
- Free and open source

SNORT

- Can be used for network intrusion detection and prevention
- Free and open source
- Uses a simple rules description language to create rules
- Snort rules are divided into 2 logical sections: rule header and rule options

SNORT

- Can be used for network intrusion detection and prevention
- Free and open source
- Uses a simple rules description language to create rules
- Snort rules are divided into 2 logical sections: rule header and rule options
  - The rule header contains the rule's action (e.g., log, alert, drop), protocol, source and destination IP addresses and netmasks, and the source and destination ports information

SNORT

- Can be used for network intrusion detection and prevention
- Free and open source
- Uses a simple rules description language to create rules
- Snort rules are divided into 2 logical sections: rule header and rule options
  - The rule header contains the rule's action (e.g., log, alert, drop), protocol, source and destination IP addresses and netmasks, and the source and destination ports information
  - The rule option section contains alert messages and information on which parts of the packet should be inspected to determine if the rule action should be taken.

SNORT

- Can be used for network intrusion detection and prevention
- Free and open source
- Uses a simple rules description language to create rules
- Snort rules are divided into 2 logical sections: rule header and rule options
  - The rule header contains the rule's action (e.g., log, alert, drop), protocol, source and destination IP addresses and netmasks, and the source and destination ports information
  - The rule option section contains alert messages and information on which parts of the packet should be inspected to determine if the rule action should be taken.
- action proto src_ip src_port direction dst_ip dst_port (options)

SNORT

- Can be used for network intrusion detection and prevention
- Free and open source
- Uses a simple rules description language to create rules
- Snort rules are divided into 2 logical sections: rule header and rule options
  - The rule header contains the rule's action (e.g., log, alert, drop), protocol, source and destination IP addresses and netmasks, and the source and destination ports information
  - The rule option section contains alert messages and information on which parts of the packet should be inspected to determine if the rule action should be taken.
- action proto src_ip src_port direction dst_ip dst_port (options)
- Example: `log tcp any :1024 -> 192.168.1.0/24 500:`
- Log tcp traffic from priveleged ports less than or equal to 1024 going to ports greater than or equal to 500

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 was compromised and how (forensics)
  - Restore to a clean state (reinstall, restore clean data backup)