

ENTS 689i  
Part II: System Security

Lecture 7: Malicious software

# Today's Class

- Attacker overview
- Types of malware
- Malware techniques
- Malware mitigation
- Trust and challenges
- Exam and HW questions

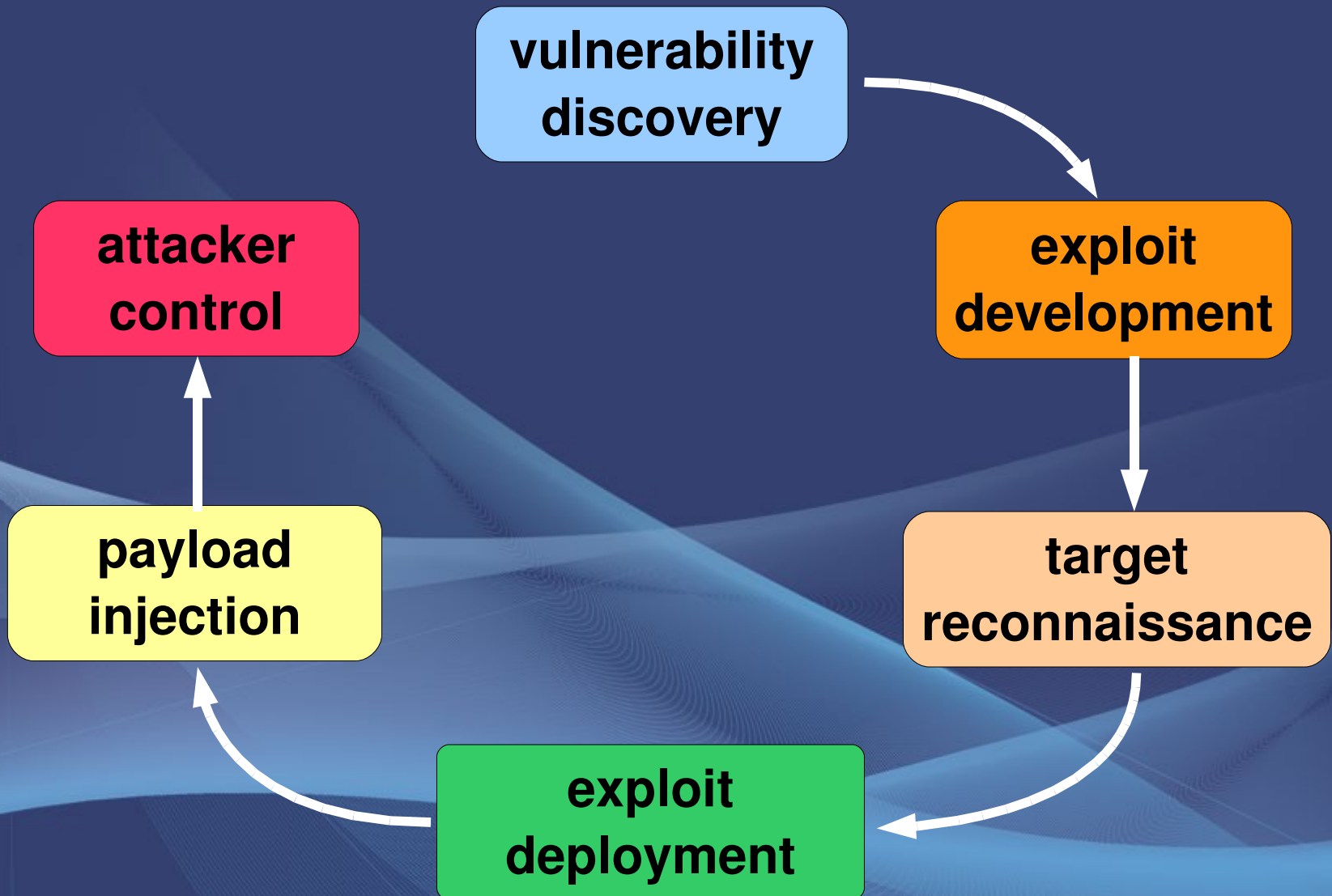
# Attacker profiles

- **Attackers come in many different forms**
  - Someone you don't know
  - Someone you know
  - Someone you trust (**insider**)
- **Attackers have different targets**
  - Random person/machine
  - A specific person or group
  - A specific organization
  - A specific technology

# Attacker profiles

- **Attackers have different tools/skills**
  - Download and use tools (“script kiddies”)
  - Modify/implement known techniques
  - Develop/use advanced toolsets
- **Attackers have different motivations**
  - Financial
  - Political
  - Ideological
  - Reputation or ego

# Lifecycle of attack



# Malware classifications

- Many ways to characterize
- **Functionality**: what the malware does
  - keystroke logger, packet sniffer, stealth, ...
- **Mechanism**: how it does it
  - API misuse, DLL injection, function hooking, code modification, ...
- **Injection**: how it got installed
  - e.g., trojan horse, virus infection, worm, ...

# Malware functionality

- **Malware can do many different things**
  - Like any other program
  - Depends on privilege level at which it runs
  - We will discuss some common ones
- **Hiding: remain on system undetected**
  - Can be passive or active
  - “hiding in the noise”
  - built-in hidden file features
  - rootkit techniques, OS modification
  - Antivirus subversion

# Malware functionality

- **Reconnaissance: collect user/system info**
  - Harvest email/IP addresses of new targets
  - Keystroke logging
  - Packet sniffing
  - Password cracking
  - Check if vulnerable to later stages of attack
  - Commonly associated with **spyware**
- **Attack: target other systems**
  - Use one machine as a **stepping stone**
  - Decreases chance of detection/attribution

# Malware functionality

- **Backdoor: insert a way back in**
  - Open network port
  - Add secret user account
  - Disable security checks
- **Privilege escalation:**
  - Give user a way to run as system/superuser
- **Persistence: find a way to run again**
  - Commonly modify startup folder or registry
  - Embed in/replace program
  - As early in load sequence as possible

# Malware functionality

- **Data destruction: remove or modify data**
  - Might delete critical system files
  - Or user data
  - Variant: encrypt/remove user data for ransom
- **Data distribution/storage**
  - Use compromised server as control node
  - Warez servers: illegal/stolen content

# Trojan horse

- Appears to do something good or expected
- Has unexpected or hidden functionality
- Frequently performs much/all of the anticipated functionality
- **Common examples:**
  - Sysadmin tools, e.g., `ls`, `ps`, `lsmod`
  - Screen savers, games, user “toy” programs
  - Email attachments

# Virus

- Common term for many forms of malware
- Originally: malicious logic inserted into another program or file
- “Attach” to the target without user knowing
- Most common form is document virus
  - e.g., Word, Powerpoint, PDF, images
- Difference from a trojan horse:
  - Infects other files/programs (**replication**)
- Typically launched by user or their program<sub>12</sub>

# Document Virus

- Commonly delivered via email
  - “Check out this important file”
- Many programs have rich content features
- Embed many types of functionality
- Problem enhanced by **auto launchers**
  - Open by file extension
  - Open by file content/magic headers
- Also can use program **macro** languages
  - Program-specific series of repeatable steps

# Boot Virus

- Modifies computer boot/load sequence
- Typical boot process:
  - BIOS -> boot loader -> OS kernel -> programs
- What if we load virus before kernel?
- Doesn't give OS chance to check/prevent installation
- Most loaders don't verify before loading
- Typical installation: replace boot section
- Can also happen at firmware level

# Worm

- Spreads from machine to machine
  - Typically, without user action
- Some worms do need user involvement
  - User clicks email, etc.
- Frequently used to describe fast or widespread outbreaks
- Examples:
  - Code red (IIS)
  - Blaster (DCOM RPC)

# Spyware

- Generally, software that collects user info
- May include many forms of collection
  - Keystroke logging
  - Tracking user web viewing
  - Collecting critical identification info
  - Email address harvesting
- *Many* ways to install, very hard to remove
- *Adware*: force user's attention to products
  - Comes included with some downloads

# Rootkit

- Provides stealth and persistence
- Refers to ability to maintain “root” privilege
- Typically involves some form of hiding
  - Files, processes, network connection
- Frequently includes backdoor for reentry
- Can occur at user or kernel level

# Common rootkit techniques

- Trojan of administrator program
  - e.g., hide files with malicious ls
- User process DLL injection
- Malicious driver installation
- Interrupt/system call/function hooking
- Code patching
- Data-only attacks

# Software trust

- How do we know which software to trust?
- Given to us by our friend?
  - Where did he or she get it?
- Created by a trustworthy vendor?
  - Software can still have accidental bugs
- Downloaded from public web site?
- How do we know what a program does?
  - All of the files it reads/writes
  - All of the packets it sends and to whom

# Software trust

- Hard to know what program **should** do
- Particularly difficult to identify all required access permissions
- Moving same code into new environments can change access requirements
- Two key trust questions:
  - What software is running on my own machine?
  - Can remote party trust other end of session?

# Confused deputy problem

- Imagine a compiler, shared by users
- Users pay for use of the compiler
- Compiler logs billing and statistics in `/var/billing`
- Compiler is run as follows:  
`/bin/compiler <input.c> <output>`
- What accesses does the compiler need?

# Confused deputy problem

- Compiler needs two sets of permissions
  - User's permissions for input/output files
  - Permission to write to the billing file
- What if the user does the following?  
`/bin/compiler myfile.c /var/billing`
- Compiler has permissions to write to billing!
- Compiler is a **confused deputy**
- Problem: assigning accesses is hard

# Malicious software mitigation

- **Virus scanners**
  - Look for instances of known bad software
- **File authentication**
  - Verify the file comes from a trusted source
- **Integrity verification**
  - Verify important info has not been modified
- **Least privilege**
  - Run new programs in constrained environment

# Virus scanners

- Most common form of malware mitigation
- Almost everybody has it and knows about it
- Contain large list of malware **signatures**
  - Match a particular piece/family of malware
- Generally, signatures are particular pattern
  - ASCII string or other data
  - Code sequence
- Can also define certain **heuristics**



# Polymorphism and obfuscation

- Malware detection is currently an **arms race**
- New malware -> detection -> new malware
- Virus scanners look for known signatures
- Attackers want to achieve the same goals
  - But not be detected by AV
- Malware writers try to introduce diversity into their binaries to avoid signatures
  - But still maintain the same functionality
  - This is referred to as **polymorphism**

# Polymorphism and obfuscation

- Frequently implemented using **encryption**
  - Same binary, encrypted with different keys
  - At runtime decrypt binary, start executing
- Using compression also common
  - Self-extracting binary
- Referred to generally as **packing**/unpacking
- Also makes reverse engineering harder
- Alternative: use different binaries with equivalent functionality (e.g., insert no-ops)<sub>77</sub>

# Polymorphism to heuristics

- Virus diversity leads to trying to identify more general “bad things”
- Example: any code that self-extracts or self-modifies may be flagged
  - Can be false positives
  - But easy to **white list** known good programs
- Can also look for changes to commonly-modified targets
  - E.g., certain registry keys or files

# Spyware/rootkit detection

- Similar goals to virus scanning
  - Detect evidence of malware
- Look for signs of known bad
- Look for signs of particular behavior
  - Registry keys
  - HTTP cookies/session info
  - Web browser settings

# Memory-only malware

- The arms race continues
- Traditional anti-malware programs check programs and persistent data stores
  - E.g., disks, registry
- Once in memory, malware may delete itself or never be disk resident
- “Go where they aren't looking”
- Next week we'll talk about memory analysis

# Deception

- Detect attackers by deceiving them
- Insert decoy information or resources
- Lure the attacker into interacting with them
- Trustworthy users won't try to access
- **Honeytoken**: invalid information such as login credentials or credit card
- **Honeypot**: deploy a system to be attacked
  - Any access is by definition illicit
- **Digital watermarking**: track files in the wild

# Host-based IDS/IPS

- **Intrusion detection**: monitor activity, look for signs of intrusion
- Alert administrator if something is detected
- HIDS: typically one or more software agents interpreting **observable events**
  - e.g., logs, file accesses, other system calls
- Concept can be extended to **prevention** if action point is in the right place (HIPS)

# Host-based IDS/IPS

- Requires a mechanism to distinguish “good” from “bad”
  - State-based: “once a process opens file X, it should never do Y”
  - Policy-based “no process except login should ever read passwords”
- **False positives**: IDS generates alert, but activity is not malicious
- **False negatives**: IDS does NOT alert on malicious activity

# HIPS in practice

- Most AV companies now sell HIPS
- Virus checking is done at **access time**, rather than periodic scans
- Other policies also enforced
- Most common: “should this program have access to this resource?”
  - Vista's well-known “confirm or deny” dialogs
- Modern HIPS look like ad-hoc MAC

# Mandatory access control

- OS can enforce least privilege and minimize effects of compromise
- Not all programs can do all things
  - Even if superuser
- Challenge: difficult to setup policy, hard to use system if setup poorly
- Assumes OS remains trustworthy
  - Otherwise, simply bypass checks

# Software authentication

- Many files are distributed with signatures
  - Computed over hash of file
  - Sometimes embedded in file
  - Sometimes in the form of a manifest
- Relies on public PKI
  - Trust in CA companies, such as Verisign
- Relies on user actually checking
- Implicitly relies on user's verification tools
  - How do we verify the verifiers?

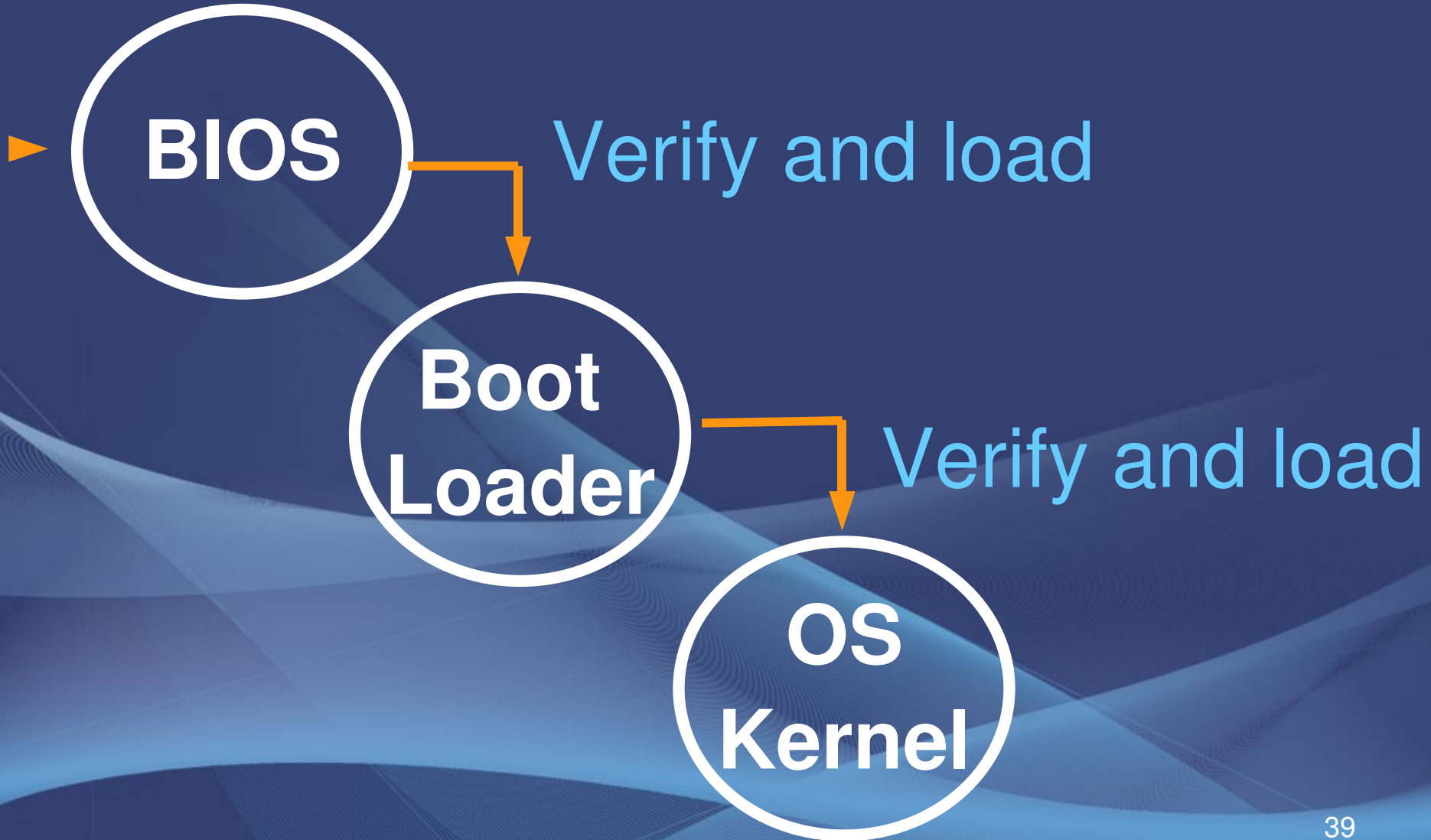
# File integrity

- Check to make sure file has not changed
- Easiest for invariant files, such as programs and libraries
- At installation: create catalog of file hashes
- Periodically: verify files on still match
- **Tripwire** is a common software tool
- For higher fidelity, boot from trusted media
  - Malware could lie to the tool

# System integrity

- More generally, we want to ensure whole system has integrity
- Static data integrity: measure at rest
- **Load integrity**: measure before loading
- **Runtime integrity**: measure running process
- Typically based on “layered” abstraction
- Lower layers measure higher layers

# Secure boot



# Trusted computing

- Large industry initiative
- Goal: make it easier to trust/verify software
- Developed a set of specifications
  - TPM: embedded security microcontroller
  - TSS: software stack for secure TPM use
  - TCN: trusted network connect
  - ...
- Mobile TPM (cell phones) in design stage

# Trusted Platform Module

- Microcontroller in many modern systems
- Set of primitives for secure systems to use
  - **Secure storage**: protect secrets with cryptography and separation
  - **Secure measurement**: unforgeable hashes that can be used to attest to a system's state
- Derive properties from these primitives
- Used by some security software
  - Microsoft BitLocker full disk encryption

# Trusted computing

- How can trusted computing help with malicious software?
- **Enables platform authentication**
  - User can prove what kind of device he is using
- **Enables platform state attestation**
  - User can prove what software has been loaded
- Remote or local entities can enact policies
- Example: **authenticated boot**
- Two steps: measurement and reporting

# Trusted computing challenges

- **Will users adopt it?**
  - Some concerns about who controls machine
  - Some concerns about DRM
  - Many concerns about usability
- **Will it be implemented correctly?**
  - Most solutions currently rely on large measurement components, e.g., OS kernel
  - Load time guarantees may not translate to runtime security