System Security – Malware Defense I

TDDE62 – Information Security: Privacy, System and Network Security

Ulf Kargén

Department of Computer and Information Science
Linköping University

LINKÖPING UNIVERSITY

Malware Defense – Agenda

- Two lectures
  - **Lecture I**: Malware basics, malware on the PC, antivirus techniques
  - **Lecture II**: Mobile malware and machine learning for malware detection

- Today's agenda:
  - Basic concepts and terminology
  - Types of malware
  - The malware detection cat-and-mouse game
    - Common techniques used by antivirus software
    - Common obfuscation used by malware to evade detection

325,000 new unique malware samples per day according to AV-TEST

- Around 1.1 billion known unique malware samples exist today

Estimated cost of cyberattacks 2023 was **$8 trillion**

Definition and Terminology

Malware is software designed with the *intention* of causing some harmful effects.

Basic terminology

- A piece of malware typically belong to an entire ***family*** of malicious software with similar functionality and code structure
  - New variants of a family appear as malware authors update their code to add new functionality, or to evade existing malware defenses
- An individual member of a family is called a ***variant***
- A specific malware binary is typically referred to as a ***sample***

Most PC malware target the Windows platform due to its large market share

- Mac and Linux malware also exist, but is comparably more rare
- Today, smartphones are also heavily targeted by malware authors – more about this in next lecture

Malware Naming

Antivirus (AV) companies often assign names to malware

- For example, “W32/Zeus.B” is the name given to a variant of the Zeus malware by a particular AV company

Common that different AV companies use different names and naming schemes

- For example, “Trojan-Spy:W32/Zbot” is an alias for the Zeus malware (assigned by a different AV company)
- File hashes (e.g. MD5, SHA1 or SHA256) are typically used to uniquely identify individual samples

Malware Nomenclature

Malware is divided into different types according to an “informal” nomenclature

- Not entirely consistent…

Based on either

- The malware’s goal/functionality
- The method of infection

Malware Types based on **Functionality**

- **Spyware** extracts sensitive information from victim system and sends it to attacker. Logs keystrokes or scrapes screen contents for stealing e.g.:
  - Credentials for email/social media accounts,
  - Credit card numbers,
  - Banking details
- **Adware** modifies e.g. browser settings to litter user with ad popups.
- **Botnet clients** silently turn victim machines into a remotely controlled node in a botnet
  - Malware connects to a Command & Control (C&C) server to receive instructions from botnet operator
  - Botnet can be used to stage DDoS attacks for e.g. extortion
  - Operators of botnet frequently also rent out DDoS capacity to other criminals

Malware Types based on **Functionality**

- **Cryptojackers** use infected computer's hardware to mine cryptocurrency for attackers
- **Ransomware** encrypts all files on hard drive and then requires a ransom to be paid for restoring the system.
  - Typically use public-key crypto ⇨ Only operators have secret to decrypt files
  - After ransom is paid (typically in Bitcoin), operators use C&C channel to instruct malware to decrypt files
  - Also common to threaten to release sensitive data if victim doesn't pay
- **Droppers** are simple executables designed to "drop" other malware onto a computer. Payload malware can either be contained inside dropper itself, or be downloaded
- **Remote Access Tools (RATs)** provide remote "back door" access into infected machines
- **"Advanced Persistent Threats" (APTs)** are advanced malware designed to evade detection for an extended period of time. Used for e.g. espionage (nation state or corporate) or "cyber warfare".

Malware Types based on **Method of Infection**

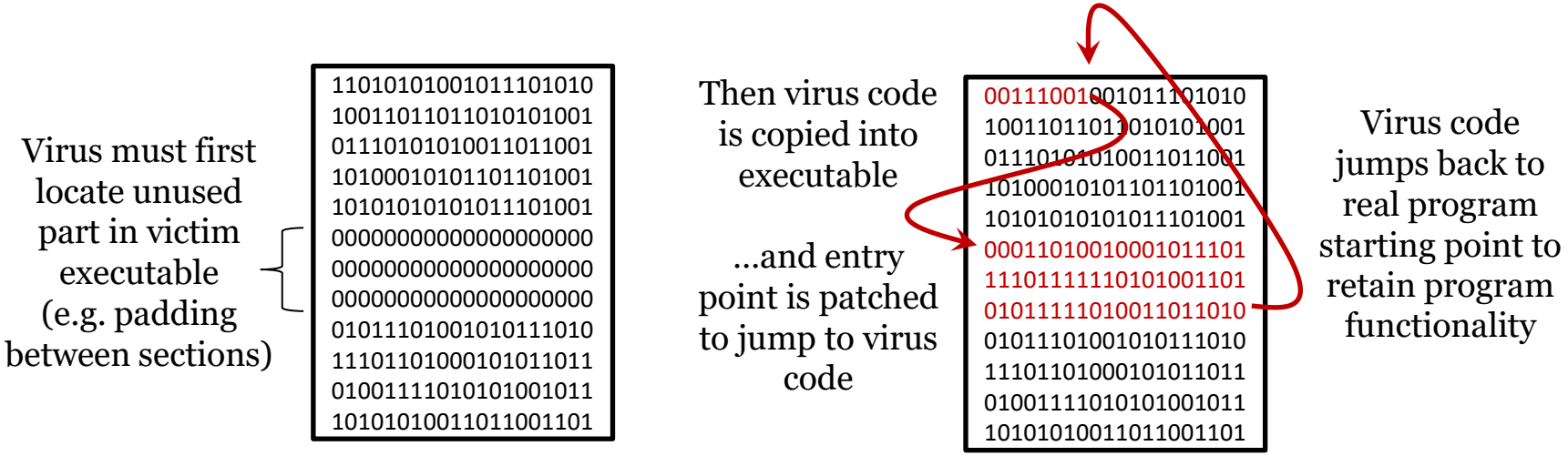
Three main types based on infection strategy

- Viruses
- Worms
- Trojans

Malware Types based on **Method of Infection**

True **viruses** are the earliest form of malware

- Emerged during the 1980s – basically extinct today
- Needs a "host" program to be able to function
  - When executed, virus will splice its own code into other executables in system



Malware Types based on **Method of Infection**

Virus code must be small to allow piggybacking on existing executables

- Viruses typically had simple functionality
- Written mostly as digital “pranks” (though some were extremely destructive)
- Motivation was mostly the challenge itself and to “show off” to others in the hacker community

Today, malware writers are almost exclusively motivated by some kind of gains (economical, political, etc.)

- Viruses too simple to support “useful” functionality – therefore basically unheard of nowadays

Malware Types based on **Method of Infection**

Worms are standalone malicious programs capable of *automatically spreading from system to system*

- Most prevalent from mid-early 2000s until around 2010
- Typically exploits unprotected network shares or unpatched vulnerabilities in network protocols to spread
- More rarely seen today
  - Modern systems have sufficiently hardened default configuration to avoid *automatically* exploitable flaws in most cases
  - This means too few infectable systems to support worm "business model"

For example the *Conficker* worm exploited a buffer overflow in a Windows service to spread and form a botnet.

Later versions attempted to spread via poorly configured network shares, using a dictionary attack to attempt to break password-protected shares.

Malware Types based on **Method of Infection**

Trojans are malware that attempts to pose as useful software to trick victims into installing/running it

- A very broad term…
- In practice, used for most malware that doesn't contain functionality for automatically spreading to new systems (i.e. everything that isn't a virus or worm)

Trojans frequently pose as, for example

- Seemingly legitimate documents with malicious macros – drops malware onto system if opened and macro execution is allowed
- Fake video codecs/players
- Fake antivirus software
- Fake pirated software/games or fake game "cracks" (DRM bypasses)
- Trojanized versions of real software
- … among others

Malware Types based on **Method of Infection**

According to recent statistics, > 90% of malware is delivered via email (e.g. malicious Word documents)

Another common infection vector are **drive-by-downloads**

- Automatically infect users who visit a malicious web page

Often performed using an **exploit kit** (EK)

- Web app specifically designed to infect visitors with malware
- Either made for in-house use by organized crime group, or sold to others on the black market

Exploit Kits

Attacks typically happen like this:

1. Attackers either manage to get an ad distributor to show malicious ads on a web page, or they hack a legitimate web page
   - Malicious ad or hacked page opens an iframe or redirects to exploit kit landing page
2. Visitors of affected web page is redirected to EK landing page
3. EK uses the “user agent” information to find out OS and browser version
   - Checks internal database for known exploits for the browser, and serves up the right exploit to victim
4. Exploit runs in victim browser, and installs malware of attacker’s choice

EKs usually uses relatively “old” known exploits, taking advantage of users who don’t install updates.

Malware Detection

Antivirus programs detect malware samples by scanning files of the computer and matching them against *signatures* and *heuristics*

Exact inner workings of AV software is mostly kept secret

- Knowledge of commercial AV techniques pieced together from public documentation, educated guesswork and reverse-engineering of AV products…

Malware Detection – AV Fundamentals

AV software scan files in filesystem both periodically and on-demand

- By “hooking” system APIs for opening files, a file can be scanned as soon as it is e.g. downloaded or the user attempts to access it

The database of malware signatures is updated typically several times a day

- AV companies typically receive millions of new samples every day
  - Filtered using data-mining techniques before manual analysis to create new signatures and updated heuristics to be sent out to clients
  - Signatures can match individual samples or entire families of malware

Malware Detection – Misclassification

The *false negative* rate (FNR) of an AV product determines how many malwares that are missed (i.e. detected as benign software)

The *false positive* rate (FPR) determines how frequently benign software is misclassified as malicious

- AV software must have a *reasonably* low FNR to be useful.
- It is crucial that AV have *extremely* low FPR!
  - For example, important system files mistakenly flagged as malicious and deleted by AV software can have disastrous effects!

Challenge: Malware authors often use *obfuscation* to increase chance of AV misclassifying their malware as benign

- Primary reason is to prevent/slow down *automatic* detection
- Also to slow down manual analysis, but this is a secondary goal (c.f., for example, DRM schemes where manual analysis is the main threat)

The Malware Detection Cat-and-Mouse Game

Malware authors constantly attempt new ways of evading AV software.

- AV companies constantly update their products to defeat evasion techniques…

Most basic detection technique: **Signatures**

- Simple string matching (binary or ASCII) at fixed offsets in files
  Example: "match the string `X50!P%@AP[4\PZX54(P^)7CC)7}` at offset 126"
- Hashes over the entire file are sometimes also used – matches a specific sample
- Still the "main line of defense" in many AV products

Malware countermeasure: Simple *polymorphism*

- Change a few bytes, append or prepend data to executable – changes offsets or content of matched sections

The Malware Detection Cat-and-Mouse Game

AV evolution: More complex signatures and static *heuristics*

- Signatures can be made more resilient to simple polymorphism by being format-aware
  - Parse headers of e.g. executable files and match against specific fields of header
  - However, this requires that AV software contains parsers for a huge number of complex formats (executables, documents, etc.)
    - Increases attack surface for software exploits against *the AV software itself* !
    - Several examples of exploitable vulnerabilities in AV in the past…

The Malware Detection Cat-and-Mouse Game

Fuzzy hashing is another type of advanced signature

- Hash functions constructed so that the difference between H(x) and H(y) is "proportional" to difference between x and y (minimal diffusion)
- This is the opposite of what we want for cryptographic hashes!

Computing a "diff" between fuzzy hashes gives a good approximation of the amount of difference between hash inputs

- Can be used as a kind of "soft" signatures to defeat simple polymorphism

The Malware Detection Cat-and-Mouse Game

Examples of *ssdeep* fuzzy hashes:

Proin sapien dolor, pellentesque tincidunt nulla id, aliquet porttitor ligula. Phasellus euismod quam nunc, id ultrices lorem aliquet eget. Nulla vel sem at sapien condimentum porttitor sed mattis dolor. Maecenas id faucibus risus, ut aliquet felis. Nulla efficitur nisi tellus, ut luctus massa volutpat quis. Pellentesque ultrices pretium imperdiet. Nulla laoreet ipsum turpis, id aliquam nisl volutpat sit amet. Nulla eget odio ut nibh fringilla consequat eget quis quam. Duis in lacinia cras amet.

12:08tK4FBFcKdQRsgYmK43MyGanjI
WUqQxozy:0jUFcKWjK431dUWVD+

Proin sapien dolor, pellentesque tincidunt nulla id, aliquet porttitor ligula. Phasellus euismod quam nunc, id ultrices lorem aliquet eget. Nulla vel sem at sapien condimentum porttitor sed mattis dolor. Maecenas id faucibus risus, ut aliquet felis. Nulla efficitur nisi tellus, ut luctus massa volutpat quis. Pellentesque ultrices pretium imperdiet. Nulla laoreet lorem turpis, id aliquam nisl volutpat sit amet. Nulla eget odio ut nibh fringilla consequat eget quis quam. Duis in lacinia cras amet.

12:08tK4FBFcKdQRsgYmK43MyranjI
WUqQxozy:0jUFcKWjK431+UWVD+

Proin sapien dolor, pellentesque tincidunt nulla id, aliquet porttitor ligula. Phasellus euismod quam nunc, id ultrices lorem aliquet eget. Nulla vel sem at sapien condimentum porttitor sed mattis dolor. Maecenas id faucibus risus, ut aliquet felis. Nulla efficitur nisi tellus, ut luctus massa volutpat quis. Pellentesque ultrices pretium imperdiet. Nulla laoreet ipsum turpis, id aliquam nisl volutpat sit amet. Nulla eget odio ut nibh fringilla consequat eget quis quam. Duis in lacinia cras amet.

12:08tK4FBFcKdQRsgYmK43MyGanjI
WUqQxozy:0jUFcKWjK431dUWVD+

Proin sapien dolor, pellentesque tincidunt nulla id, aliquet porttitor ligula. Phasellus euismod quam nunc, id ultrices lorem aliquet eget. Ut porttitor finibus massa sed commodo sed mattis dolor. Maecenas id faucibus risus, ut aliquet felis. Nulla efficitur nisi tellus, ut luctus massa volutpat quis. Pellentesque ultrices pretium imperdiet. Nulla laoreet ipsum turpis, id aliquam nisl volutpat sit amet. Nulla eget odio ut nibh fringilla consequat eget quis quam. Duis in lacinia cras amet.

12:08tK4FBFcKOb5V9RsgYmK43MyG
anjIWUqQxoz7v:0jUFcKOb5V9jK431d
UWVDXv

The Malware Detection Cat-and-Mouse Game

Heuristic matching is also common in AV engines

- Instead of "fingerprinting" malware using some unique signature, check for general signs of suspiciousness
  - If enough suspiciousness indicators are found, sample may be flagged as potential malware
  - Heuristic matching can have high FPR – often used as a pre-filter to determine if a file should be subjected to more expensive (and precise) analysis
- Heuristic engines are typically *expert systems* that approximate the decision-making process of a human analyst – similar to a manually-crafted decision trees

Example of suspiciousness indicators:

- Certain combinations of API imports
- Malformed headers in executables
- Use of obfuscation

The Malware Detection Cat-and-Mouse Game

Malware countermeasure: Packing

- Wrap a compressed and/or encrypted copy of malicious executable inside another executable
  - File signature now looks completely different!
  - Wrapper binary decrypts malicious code into memory and transfers execution to it (*Note that this is different from a dropper that writes a malware executable to the file system*)
  - Easy to create malware with different signature by changing crypto key
  - Unpacking code can further be changed between samples by employing polymorphism/metamorphism
- Tools that create such "packed" executables are referred to as *packers*
  - Many free and commercial (for DRM purposes) packers
  - Also special-purpose malware packers sold on black market

Packed malware binary

Side Note: *Polymorphism and Metamorphism*

Umbrella terms for methods that transform an executable to "look" different while preserving its original semantics (i.e. functionality)

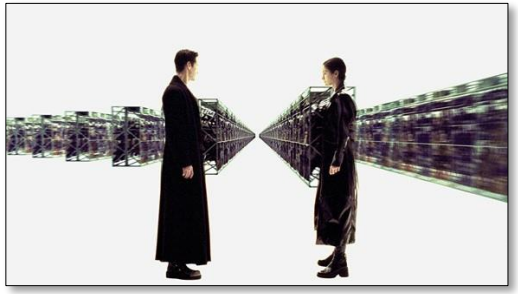
Somewhat ill-defined terms…

- Many different exact definitions, but commonly defined roughly as:
  - **Polymorphism:** Transformations that *doesn't actually change the code* of an executable. For example:
    - Appending or prepending data
    - Packing
    - Encrypting resources (strings, etc.)
  - **Metamorphism:** Transformations that create *syntactic* changes to make code look different, while *retaining semantics*. For example:
    - Different register allocations
    - Superficial changes to control-flow (e.g. swapping order of code in executable while retaining old flow relationships)

The Malware Detection Cat-and-Mouse Game

AV evolution: *Static unpacking* and *emulation*

- For simple packers with known functionality, possible to statically unpack payload executable
  - Requires that crypto keys are stored at a known position in wrapper executable
  - More advanced packers compute keys dynamically, or use other tricks
- Therefore, most AV products today use emulation to run suspicious binaries in a simulated environment
  - Can apply signature matching after unpacking code has run – works also on advanced or unknown packers
  - Dynamic heuristics can be applied to check for suspicious *behavior* (sequences of API calls, API call arguments, etc.)



- Emulation is resource-intensive
  - Typically applied only if heuristics indicate suspiciousness
  - For example, a large high-entropy section in a binary indicates use of packing

The Malware Detection Cat-and-Mouse Game

Malware countermeasure 1: *Emulator fingerprinting*

- Perfect emulation is not possible in practice
  - Emulator only handles subset of machine instructions
  - Only a subset of system APIs are emulated – typically in a highly simplified fashion

Possible for malware to detect that it runs in a simulated environment and refrain from exposing malicious functionality!

- AV companies need to constantly update their software to keep track with emulation bypass techniques used in malware!

The Malware Detection Cat-and-Mouse Game

Malware countermeasure 2: *Malware creation kits* allow easy creation of new malware variants – "overwhelm" AV companies with new samples

- User-friendly modular tools sold on black market for malware creation
- Allows less tech-savvy cybercriminals to carry out e.g. ransomware attacks

AV evolution: Cloud-based detection

- Send samples for analysis in cloud instead of on local client
  - Select candidates for cloud-based detection using e.g. heuristic suspiciousness score
  - Threat signatures can be updated in real time instead of periodic updates of AV client software
- Allows more "expensive" analysis
  - Advanced dynamic analysis (more accurate emulation, etc.)
  - **Machine learning based detection – *topic of next lecture***

Evading Antivirus Software

Possible to evade AV products with moderate effort

- Signature-based detection is still the most commonly used way to detect malware
  - ⇨ Systematically try modifying different parts of a malware binary until no AV detects it

Heuristic detection of entire families of malware is becoming more common due to the ease of creating new malware variants.

- Also possible to evade:
  - Black-box testing like above
  - Manual reverse-engineering of AV to understand heuristic rules

Malware Detection Conclusions

AV good for protecting against (variants of) *known* malicious programs

- Protects mostly against *non-targeted* opportunistic attacks
- Newly created malware, or new variants of malware designed for evasion, often slip through the net

Traditional AV mostly useless for detecting targeted APTs created by e.g. nation states or advanced cybercrime groups, instead:

- (Semi-) manual auditing of computers and networks traffic
- Intrusion detection systems

Intrusion Detection

Use either **Network Intrusion Detection System** (NIDS)

- Detect e.g. anomalous traffic to C&C servers

or use **Host Based Intrusion Detection System** (HIDS) to detect malicious activity on host computer

- Detect unexpected changes to filesystem
- Monitor program behavior
  - Many AV products implement this kind of HIDS to detect e.g. software exploits used to plant malware on machine
  - Monitor system APIs to detect anomalies indicative of e.g. a drive-by-download attack
    - Processes unexpectedly starting threads, opening new processes, loading new library binaries, opening sockets, etc.
    - AV can suspend execution of suspect process and scan its memory for e.g. signs of software exploits, etc.
  - Still possible to evade by determined attacker…

Hindering Manual Reverse-Engineering

Main objective of using obfuscation in malware is to evade automatic detection (dodging signatures or heuristics)

Also quite common to add obfuscation that slows down manual analysis

- Delay creation of signatures for new malware variants

Common approaches:

- *Anti dynamic analysis tricks* – make dynamic analysis with e.g. debugger harder
  - Detect if program is being debugged and, if so, terminate
    - APIs for checking if a debugger is attached
    - Scanning/checksumming own memory space to check for e.g. added breakpoints in code sections
- *Control-flow obfuscation* – prevent reconstruction of control-flow graph from a binary by using various code transformations
- *Disassembly desynchronization* – makes static analysis of executable code harder by fooling disassemblers to output incorrect assembly

Summary

- Typically many variants of each malware family
- Different types of malicious goals
  - Ransomware, spyware, botnets, etc.
- Different infection strategies – viruses, worms, trojans
  - Most malware today are some form of *trojan* – either relies on social engineering or software exploits (drive-by-downloads, malicious email attachments, etc.)
- AV use signatures and heuristics for detection
- Malware often employ obfuscation to evade AV detection
  - Polymorphism/metamorphism, packing
- AV constantly evolve to handle new evasion methods
  - Emulation, static unpacking, behavioral monitoring (HIDS)

Summary (cont.)

- AV mostly effective against variants of known malware used in *non-targeted* opportunistic attacks
  - Determined attackers can craft custom malware for targeted attacks that evades known AV – since malware is not spread *en masse*, malware is never picked up by AV companies and no signatures are generated
  - Intrusion detection systems and network monitoring is necessary to spot such advanced malware
- Main goal of obfuscation in malware is to evade automatic detection
  - Many malwares also employ some obfuscation that deters manual analysis
  - Slows down signature generation ⇨ malware can generate revenue for attackers for a longer time before new variant need to be created