Malware (2)

System Security (CM0625, CM0631) 2024-25
Università Ca’ Foscari Venezia

Riccardo Focardi
www.unive.it/data/persone/5590470
secgroup.dais.unive.it

Università
Ca'Foscari
Venezia

Brief history of worm attacks (2)

Sobig.F (2003): exploited proxy servers to turn them into **spam engines**

- **> 1M hosts** of in **24 hours**

Mydoom (2004): mass-mailing e-mail worm

- replicated **~1000 times/minute**
- **100M** infected messages in **36h**
- **exploited IE** to install a backdoor

Samy (2005): the first Web worm, onto MySpace (details here)

Conficker (2008): one of the largest worm infection ever

- exploited vulnerabilities in Windows systems
- **millions of computers** including government, business and home computers **>190 countries**

Brief history of worm attacks (3)

Stuxnet (2010): targeting Industrial Control Systems (ICS)

- exploiting **0-day** vulnerabilities
- first **Cyberwarfare** weapon ever
- targeting the Iranian nuclear program

⇒ Worm induced **stealthy failures** on the centrifuges for uranium enrichment

Flame (2012): Cyber-espionage on Middle-Eastern countries exploiting advanced vulnerabilities

- **MD5 collisions** using a new attack! (see the paper)

WannaCry (2017): vulnerability in the SMB file sharing of Windows

- encrypting files and asking for a **ransom**

Worm “technologies”

Multiplatform: OSs, Web, …

Multi-exploit: use different exploits to spread

Ultrafast spreading: try to spread fast, thanks to multi-exploitation and 0-days

Polymorphic: as viruses, various forms to evade detection

Metamorphic: as viruses, change form and behaviour

Transport vehicles: used to transport other malware

0-day: use unknown vulnerabilities, which makes it hard to stop/detect them

Client-side vulnerabilities (1)

Bugs in user applications that allow malware to install

Drive-by download: user visits a page that downloads and install malware without user knowledge

- Typically due to **browser** and **plugin vulnerabilities**
- **Examples**: Flash and Java plugin vulnerabilities

Watering-hole attack: is a variant of drive-by download. The attacker:

- targets a **specific** victim
- discovers websites **commonly visited** by the victim and look for vulnerabilities
- **exploit website vulnerabilities** so to install the drive-by download payload

Client-side vulnerabilities (2)

Malvertising: attacker pays for advertisements that incorporate malware

- users visiting pages with malvertising would **get infected** (e.g. through drive-by download)

Clickjacking: hijack user clicks

- User clicks on a button but the click goes to a **different page**

Example: transparent layers that hide what the user is really clicking on

- Click would go to the transparent page, possibly performing **unwanted actions** (user might be logged in a session)

Propagation mechanisms

(malware classification)

1. Infection
2. Exploitation
3. **Social engineering**

Social engineering

Definition: “tricking” users to **assist** in the **compromise** of their own systems or personal information

Examples:

- a user views and responds to a **spam** e-mail
- a user permits the **installation** and execution of a **Trojan horse** program

Spam and phishing

Spam emails can carry malware:

- attached document, which, **if opened**, may exploit a software vulnerability to install malware

Phishing attacks

- **a fake website** that attempts to capture user's credentials
- **forms** with personal details to allow user impersonation

Phishing over HTTPS: fake websites have valid HTTPS certificates, thanks to free CAs such as Let's Encrypt

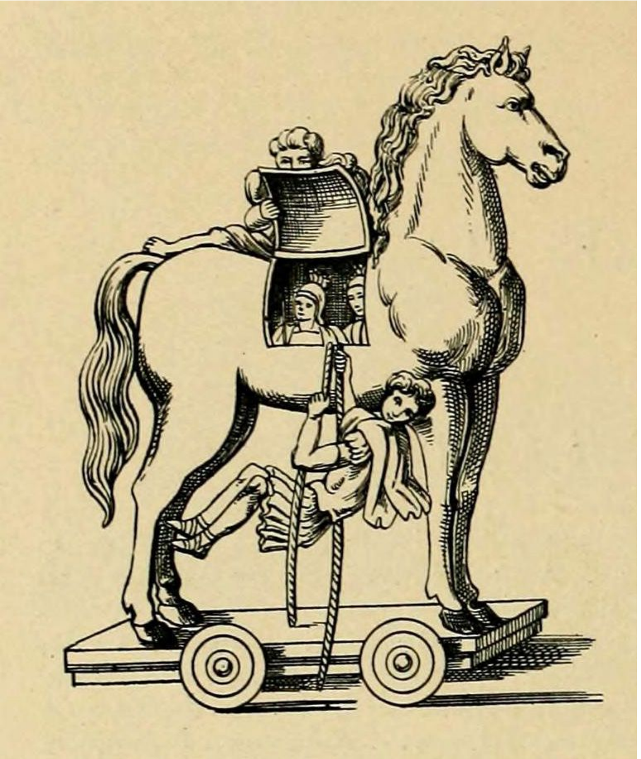
- HTTPS may create a **false sense of security**

Phishing over social networks: spam email phenomenon is reducing thanks to filters, but social media offer a **new vehicle** for social engineering attacks

Trojan horses

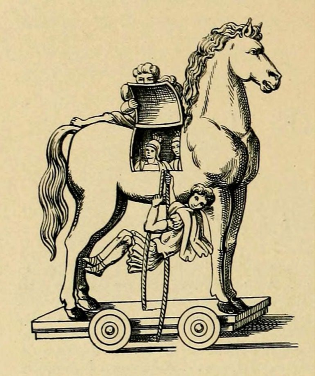
Trojan horse: a useful, or apparently useful, program containing hidden code that, when invoked, performs some **unwanted or harmful function**

- **Example**: incorporate malicious code into a game and making it available via a known app store



Categories of Trojans

1. Continuing to perform the original function and **additionally** performing a separate malicious activity
2. Continuing to perform the original function but **modifying** it so to perform malicious activity or to **disguise** other malicious activity. For example:
   a. a **Trojan horse version of a login program** collecting passwords
   b. a **Trojan horse version of `ls`** not displaying malicious programs
3. Performing a malicious function that completely **replaces** the original one



Note: some Trojans exploit vulnerabilities to install but, unlike worms, they **do not replicate**

Payload action

(malware classification)

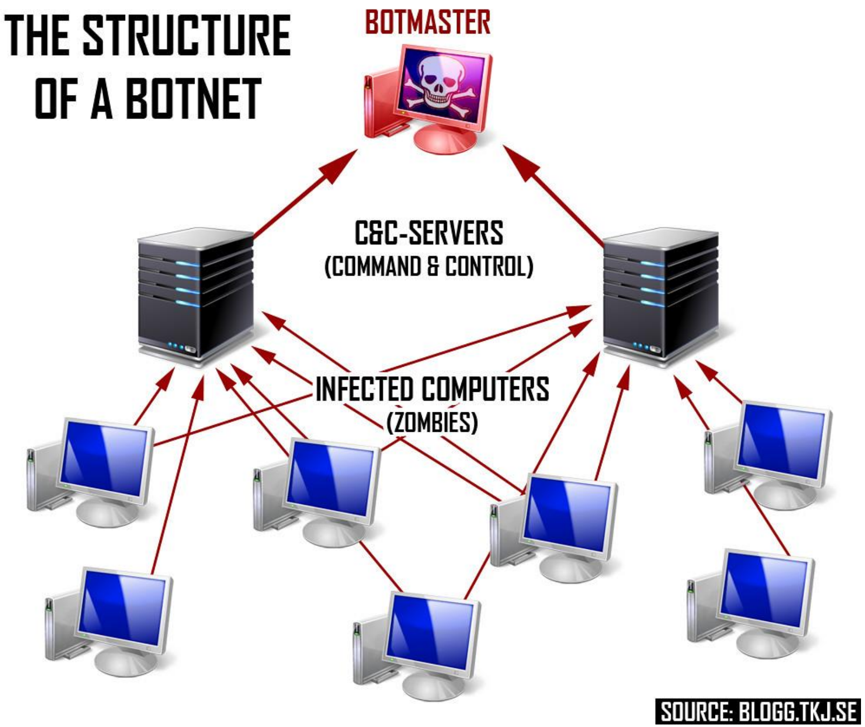
1. corruption of system / data
2. <u>**theft of a service**</u>
3. theft of information
4. stealthing

Botnets

Bot (zombie): device whose computational and network resources have been subverted for **use by the attacker**

Botnet**:** a collection of bots that can act in a coordinate manner

- thousands of computers, servers, embedded devices (**IoT**), ...



SOURCE: BLOGG.TKJ.SE

Botnet activities

Distributed DoS (DDos): **flooding** the target

Spamming: massive amount of **bulk emails**

Sniffing traffic (infected hosts): retrieving **sensitive** information

Keylogging (infected hosts): useful when traffic is encrypted

Spreading malware: botnet as the start base for **viruses** or **worms**

Automated tasks: get **financial** advantage (e.g. clicking on ads)

Manipulating polls and on-line games: votes and activities from thousand of different IPs will appear as from distinct users

Botnet Command & Control (C&C)

C&C control servers are contacted by zombies in the botnet

Fixed address: easy to take over by law enforcement agencies

Pool of addresses generated automatically: if server is down bot contacts the next address

⇒ Much harder to **detect**

C&C servers:

- issue **commands** to bots
- send **updates**
- **gather** sensitive information collected by bots

Note: A significant number of C&C have been taken over and shut down in the recent years

Payload action

(malware classification)

1. corruption of system / data
2. theft of a service
3. theft of information
4. **stealthing**

Rootkits

Rootkit: a set of programs installed on a system to maintain **covert access** to that system with **administrator** privileges, while hiding evidence of its presence

- **Persistent**: easier to detect as it needs to be stored, or
- **Memory based**: harder to detect but does not survive reboots

User mode: Intercepts APIs and modifies results. Example: hide rootkit file in `ls`

Kernel mode: privileged mode, hides processes, modifies kernel memory

Virtual machine based: runs the OS in a lightweight virtual machine

External mode: direct access to hardware (BIOS, UEFI, Intel SMM, ...)

Kernel mode rootkits

Change syscalls:

1. **Modify the system call table**: The attacker modifies entries so to point to the rootkit's functions
2. **Modify system call table targets**: The attacker overwrites selected legitimate system call routines
3. **Redirect the system call table**: The attacker redirects references to a new table in kernel memory

The idea is to exploit a "layer-below" form of attack:

- Any "anti-virus" program would now be subject to the **same "low- level" modifications** that the rootkit uses to hide its presence

⇒ Detecting the rootkit becomes really hard!

Countermeasures

Prevention:

- **Appropriate access control** (possibly MAC) so to limit virus propagation and damage
- Keep systems **up-to-date**: reduce vulnerabilities limiting worm propagation
- Improve **user awareness** so to limit social engineering attacks

Mitigation, when prevention fails:

- **Detection**: malware should be promptly detected and located
- **Identification**: once detected, identify the specific malware
- **Removal**: once identified, remove all traces of malware

Note: when identification or removal are not possible it is necessary to restore a **backup** or **reinstall** system

Sandbox analysis

Run malware in an ***emulated sandbox*** so to study its behaviour and develop adequate mitigation strategies

Problem 1: How **long** should the analysis run?

- modern malware extensively **sleep** to evade sandbox analysis

Problem 2: Is it possible to make sandbox **indistinguishable** from real setting?

- modern malware tries to detect if it is running in a sandbox and, in such a case, it **deactivates**

Example: network connections are *emulated* to prevent that malware easily notices isolation.
Read how this killed WannaCry!