Lec 16: Anti-Malware

CS492E: Introduction to Software Security

Sang Kil Cha
Anti-Malware or Anti-Virus (AV)

We will interchangeably use the terms.
Terminology

- Virus
- Worm
- Trojan
- Rootkit
- Spyware
- Bots
- Backdoor
- Adware
- Ransomware
- Etc.
AV (Anti-Virus)
Cohen’s Question

Given an arbitrary program, can we design a Turing machine that determines whether the program is malicious or not?

No, this is an undecidable problem!
Informal Proof

Define a function \texttt{isVirus} that takes a program as input, and outputs true if the program is a virus or false otherwise. Let’s assume that this function exists:

\begin{verbatim}
def isVirus(prog):
    ... # somehow test prog and returns true or false
\end{verbatim}
Informal Proof (cont’d)

Define a function \textit{myVirus}:

\begin{verbatim}
def myVirus(): # consider myVirus as a program
    if isVirus(myVirus):
        return # do nothing
    else:
        infectOtherPrograms()
        destroyUserData()
        return
\end{verbatim}

Self contradictory
Cohen’s Conclusion

• Precise virus detection is not decidable.

• Virus removal (AV) is not always guaranteed because it is dependent on virus detection.
Simplest Malware Detection

• Compute hashes of malware samples
• Compute hashes for target files and find ones that match with one of the malware hashes (a.k.a. signatures)

What’s wrong?
Easy to Bypass

• Add a dummy (dead) code

• Reorder instructions

• Replace instructions with semantically equivalent ones

Hash-based detection is still used in AV, why?
Pattern Matching (RegExp)

closeDoc{-35}setTimeout{-30}addAnnot...

A ClamAV signature for CVE-2016-0931
(Adobe Acrobat PDF exploit)
Defeating Pattern-based Detection

Signature: (\xb0\x0b)(.*)(\xcd\x80)

```assembly
mov al, 0xb
int 0x80

mov al, 0xa
inc al
int 0x80
```

Bypassing signature-based detection is so easy!
Polymorphism

Change the form of malware when it propagates in order to bypass pattern matching
Changing the Form?

• Malicious use:
  Bypass malicious code detection ($\approx$ Intrusion detection)

• Benign use:
  Software protection (make reverse engineering difficult)
Polymorphism Example

Decryption Routine

Jmp to XYZ

Encrypted Code

XYZ
Polymorphism Example

XYZ is often called **OEP** (Original Entry Point)

We can produce millions of distinct binaries (with the same semantics) by just changing the encryption key.
Self-Modifying Code

• Code that alters its own instructions while it is running

• $W \text{ xor } X$ (Write xor eXecute) policy of modern OS?
Polymorphism Example

Decryption Routine

Jmp to XYZ

Encrypted Code 1

Decryption Routine

Jmp to XYZ

Encrypted Code 2

Decryption Routine

Jmp to XYZ

Encrypted Code 3
Checking Decryption Routine

Decryption Routine

Encrypted Code 1

Jmp to XYZ

Decryption Routine

Encrypted Code 2

Jmp to XYZ

Decryption Routine

Encrypted Code 3

Jmp to XYZ
Possible to Create Signatures

Next Question:
Can we also make the decryption routine polymorphic?

See https://github.com/Yara-Rules/rules/tree/master/Packers
Polymorphic Encryption

Make the encryption/decryption routine unique!
Polymorphic Encryption (cont’d)

Original Code → Polymorphic Encryption Engine

Decryption Routine$_1$
Encrypted code$_1$
Decryption Routine$_2$
Encrypted code$_2$

...
Polymorphic Encryption Example

```c
for ( int i = 0; i < codeLen / 4; i++ ) {
    v = in[i]; // for every 4-byte value of the orig code
    key[i] = random_int(); // random 4-byte int
    op[i] = random_op(); // random operation
    switch ( op[i] ) {
        case ADD: v += key[i]; break;
        case SUB: v -= key[i]; break;
        case XOR: v ^= key[i]; break;
        ... // omitted
    }
    out[i] = v; // store the encrypted code
}
```
Polymorphic Decryption Example

for ( int i = 0; i < codeLen / 4; i++ ) {
    v = in[i]; // for every 4-byte of the encrypted code
    k = key[i];
    switch ( op[i] ) {
        case ADD: v -= k; break;
        case SUB: v += k; break;
        case XOR: v ^= k; break;
        ...
    }
    out[i] = v; // store the decrypted code
}

// The encrypted code can be located here (self-modifying)
Can We Still Write Signatures?

- Signature database will easily blow up
- Simple static pattern matching does not help anymore

Any issues in polymorphic encryption?
In-Memory Detection

• The same original code will be eventually unpacked to memory at some point

• Memory-based scanning still works! (no more static detection)

• Generic unpacking technique exists
Performance vs. Security

• Performance really matters

• **Signature-based detection** is still largely popular
Fun Fact

• Signature-based detection is fast
• But it gets slower as we add more signatures
More # of malicious apps
⇒ More # of signatures
⇒ More memory
⇒ Poor cache performance
⇒ Slow!
Motivation

Can we make signature-based scanning fast and more scalable?

SplitScreen: Enabling Efficient, Distributed Malware Detection, *NSDI 2010*
Opportunity: Fewer Signatures Matched

4 month study of CMU email malware

< 1% of signatures used by ClamAV for all malware
Traditional Signature-based AV

- All Malware Sigs
- Files
- Poor Locality
- Exact Signature Matching
- Many Bits
- Malware or not?
SplitScreen Architecture

All Malware Sigs

Good Locality

Identified Sigs

Good Locality

Files

FFBF

Exact Signature Matching

Suspect Files

Few Bits

Malware or not?
FFBF: Feed-Forward Bloom Filter

• A modified Bloom filter
• Quick matching with one-sided error
  − False positives possible
  − False negatives not possible
Traditional Bloom Filter

Patterns (signatures)

90ac0c9aff
940f047591

Bit Vector

1 0 0 1 0 0 0 1 0 1 1 0

Hash Functions
Traditional Bloom Filter

Patterns (signatures)

| 90ac0c9aff | 940f047591 |

...d9d0940f047591c320...

Target File

1 0 0 1 0 0 0 1 0 1 1 0

Bit Vector

Found Suspect

Rolling Hash
Feed-Forward Bloom Filter

Patterns (signatures)
- 90ac0c9aff
- 940f047591

Bit Vector
- 1 0 0 1 0 0 0 1 0 1 1 0

Hash Functions

...
Feed-Forward Bloom Filter

Patterns (signatures)

| 90ac0c9aff |
| 940f047591 |

...d9d0940f047591c320...

Bit Vector

1 0 0 1 0 0 0 1 0 1 1 0

Found Suspect

Rolling Hash
Feed-Forward Bloom Filter

Patterns (signatures)

<table>
<thead>
<tr>
<th>90ac0c9aff</th>
</tr>
</thead>
<tbody>
<tr>
<td>940f047591</td>
</tr>
<tr>
<td>...</td>
</tr>
</tbody>
</table>

Target File

...d9d0940f047591c320...

Bit Vector

1 0 0 1 0 0 0 1 0 1 1 0

0 0 0 1 0 0 0 1 0 0 0 0

Rolling Hash

All-patterns bitvector

Matched-patterns bitvector
Pattern Filtering

Patterns (signatures)

90ac0c9aff
940f047591

...d9d0940f047591c320...

Matched-patterns bitvector

Rolling Hash
SplitScreen Recap

All Malware Sigs

Good Locality

Identified Sigs

Suspect Files

Files

FFBF

Identified Sigs

Suspect Files

Few Bits

Exact Signature Matching

Malware or not?
Frequency-based Signature Fragment Selection

• Choose signature fragment based-upon frequency when initialize FFBF. (Choose a fragment of the least frequency)

• Example

\[
\text{2e6868636c69636b28293b\{-5\}73657474696d656f7574282242\text{7474696d656f75742822}}
\]

Frequency 0 (Choose this)

Frequency 20

Frequency 10
Throughput (1.6 GB Clean Files)
Better Cache Performance
Less Memory

![Graph showing memory usage comparison between ClamAV and SplitScreen]
Signature Distribution Cost?

As the signature database gets larger, distributing it also becomes expensive!

SplitScreen allows on-demand signature distribution
On-Demand Signature Distribution

All Malware Sigs

FFBF

Identified Sigs

Suspect Files

Exact Signature Matching

Malware or not?

Files
On-Demand Signature Distribution

All Malware Sigs

All-patterns Bit Vector

Server

Client

Files

Matched-patterns Bit Vector

Identified Sigs

Exact Signature Matching

Suspect Files

Malware or not?
Lower Signature Distribution Cost

10x less server bandwidth
Conclusion

• Perfect AV is not feasible

• Infinite war between malware authors and defenders
  – Hash-based detection
  – Signature-based detection
  – Polymorphic malware
  – Polymorphic encryption

• Signature-based detection is still critical, and SplitScreen enables efficient and distributed malware detection
Questions?