Computer Malware Analysis

Jean-Yves Marion
Professeur à l’Université de Lorraine
Institut Universitaire de France
Loria

www.loria.fr
#Labo_Loria
Today is friday

Please Relax and Enjoy !
Today’s menu

1. Malware world in a nutshell

2. Malware lifecycle

3. Defensive codes

4. Binary blob analysis

5. Detecting viruses

6. A step backward to conclude
Part 1: Malware world in a nutshell

Countries hit in initial hours of cyber-attack

- **US**: Delivery company FedEx affected
- **UK**: 48 NHS organisations disrupted
- **Russia**: Country’s interior ministry reported 1,000 of its computers infected
- **Spain**: Telecoms and gas companies struck
- **France**: Some Renault factories had to stop production

*Map shows countries affected in first few hours of cyber-attack, according to Kaspersky Lab research, as well as Australia, Sweden and Norway, where incidents have been reported since.

Source: Kaspersky Lab's Global Research & Analysis Team
A malware is a computer program that infiltrates a digital system without the user's knowledge.
Where does malware live?
Computer System, Smartphones, IOT & ICS

<table>
<thead>
<tr>
<th>Industry</th>
<th>Target System</th>
</tr>
</thead>
<tbody>
<tr>
<td>Manufacturing</td>
<td>Production system</td>
</tr>
<tr>
<td>Energy</td>
<td>Power plant system</td>
</tr>
<tr>
<td>Water</td>
<td>Water System</td>
</tr>
<tr>
<td>Transportation</td>
<td>Railway control system, Air traffic, Road</td>
</tr>
<tr>
<td>Finance</td>
<td>Core Banking System, ATM</td>
</tr>
<tr>
<td>Healthcare</td>
<td>hospital</td>
</tr>
<tr>
<td>Public safety services</td>
<td>Cameras, sensor</td>
</tr>
</tbody>
</table>

The attack surface is large
What are the purposes behind?

**Extortion**
Ransomware (Wannacry, Ryuk), banking Trojan (Locky, Dridex)

**Information extraction**
APT 28, attack against Equifax (145M victims)

**Income & e-réputations**
Spam campaign, Cryptominer

**Defacement & Destruction**
TV 5 Monde, attack on EU oct 2016, NoPetya, Stuxnet
Risks on IT systems

- **Confidentiality**
  - Loss of company data
  - Loss of personal data
  - Loss credential certificates

- **Integrity and data modification**
  - Data encryption (ransomware)
  - Corrupted data

- **Non Availability of services**
  - Critical infrastructures
  - Public services may be unavailable
  - Shut-down or destroy a device
Questions

The big failure of software industries/researches
A lot of challenging questions:
• How to reverse engineer malware?
• How to remove obfuscations?
• How to recover program semantics?
• How to detect malware?
• Fundamental aspects
  ✮ Self-replication, Reflection/Reification
  ✮ Self-modification, evolution/adaptation
  ✮ Relationship with biology/Immunology
Part 2: Malware lifecycle

1. Intrusion
2. Infection
3. Exploitation
WORLD’S MOST DANGEROUS MALWARE EMOTET DISRUPTED THROUGH GLOBAL ACTION

27 January 2021

Use Case: Emotet

Emotet was delivered to the victims’ computers via emails that contained a malicious link or an infected document. If victims opened the attachment or the link, the malware got installed. The computer became vulnerable and was offered for hire to other criminals to install other types of malware.

Emotet opened doors for:
Trojans
Ransomware

Emotet so dangerous:
- Acted as a banking Trojan 2014, evolving over time.
- Acted as a door opener for her computers, allowing unauthorized access to her malware families.
- Changed its code each time it was called up.
- Exploited way of infecting networks spreading the threat after gaining access to just a few sites in the network.

Protect yourself from malware:
- Always check your emails carefully and watch out for:
  - Attachments or embedded links from unknown senders.
  - Messages with a sense of urgency asking you to download something.
- Offers with a promise of reward that sounds too good to be true.
Emotet lifecycle

1/ Intrusion
- Social engineering: mail phishing with .doc file

2/ Infection

Phase 1: Loader
- Check privilege ...
- Manage versionning
1/ Intrusion
- Social engineering : mail phishing with .doc file

2/ Infection

**Phase 1 : Loader**
- Check privilege ...
- Manage versionning

**Phase 2 :**
- Communicates with a C&C
- Collect & Exfiltrate data
- Protocole ; encrypted http Post
- Update & Download files

**Bot Instance**
- Arrives as invoice attachment spam mail
- Tricks user to open attachment
- Downloads Emotet malware from compromised sites
- Sends gathered information to C&C server of attacker
- TrojanSpy.Win32.EMOTET
- Trojan.W97M.EMOTET
- Runs macro with PowerShell download command to connect to compromised sites
- Executes Emotet malware and gathers user information and credentials from victim’s machine
Lateral Move: Emotet lifecycle

3/ Exploitation
- Gather information
- Load Trickbot
Lateral Move: Emotet loads Trickbot

1/ Infection vector
- Emotet or dropper
- Or a Vulnerability
- Or Social engineering

2/ Infection

Phase 1: Loader
- Protection: injects in legitimate process
- Evade detection & bypass security
- Manage versionning

Phase 2: Bot instance
- Communicates with a C&C
- Collect & Exfiltrate data
- Update & Download files
- Use CVE to Spread
Your network has been penetrated.

All files on each host in the network have been encrypted with Ryuk. Backups were either encrypted or deleted or backup disks were found. Shadow copies also removed, so F8 or any other methods may not recover.

We exclusively have decryption software for your situation. No decryption software is available in the public.
3 tier of a cyber-attack

1/ Intrusion
   - Social engineering
   - Attack vector by un-intended interpretation
   - Attack vector by exploiting Bugs

2/ Infection
   - Connect to C&C
   - Protect malware installed
   - Update malware

3/ Exploitation
   - Stealing information
   - DDOS attack
   - Data tampering
   - Download new malware

Lateral move
Post-exploitation
Multi-attack Campaign
An optional short focus on intrusion and more

Il y a plus d’allégresse à assaillir qu’à défendre,
Montaigne, les essais
A bug is an attack vector

- **A bug**
  - A bug is an abnormal behavior
  - Some bugs may be exploited to hijack the execution control flow: **Vulnerability**
  - A vulnerability may be used to run arbitrary code

{Stack, Buffer, String}-overflow
Remote code execution: Eternal blue, double pulsar …

Windows Win32k Elevation of Privilege Vulnerability
CVE-2021-1732
Security Vulnerability
Released: Feb 9, 2021
CVE : Common vulnerabilities and exposure overflow exploits in 2018

CVE-2018-9838 The caml.ba.deserialization function in byterun/bigarray.c in the standard library in OCaml 4.06.0 has an integer overflow which, in situations where marshalled data is accepted from an untrusted source, allows remote attackers to cause a denial of service (memory corruption) or possibly execute arbitrary code via a crafted object.

CVE-2018-9498 In SkSampler::Fill of SkSampler.cpp, there is a possible out of bounds write due to an integer overflow. This could lead to remote code execution with no additional execution privileges needed. User interaction is needed for exploitation. Product: Android Versions: Android-7.0 Android-7.1.1 Android-7.1.2 Android-8.0 Android-8.1 Android ID: A-78354855

CVE-2018-9491 In AMediaCodecCryptoInfo_new of NdkMediaCodec.cpp, there is a possible out-of-bounds write due to an integer overflow. This could lead to remote code execution in external apps with no additional execution privileges needed. User interaction is needed for exploitation. Product: Android Versions: Android-7.0 Android-7.1.1 Android-7.1.2 Android-8.0 Android-8.1 Android-9.0 Android ID: A-111603051

CVE-2018-9473 In ihevcd_parse_sei_payload of ihevcd_parse_headers.c, there is a possible out-of-bounds write due to an integer overflow. This could lead to remote code execution with no additional execution privileges needed. User interaction is needed for exploitation. Product: Android Versions: Android-8.0 Android ID: A-65484460

CVE-2018-9363 In the hidp_process_report in bluetooth, there is an integer overflow. This could lead to an out of bounds write with no additional execution privileges needed. User interaction is not needed for exploitation. Product: Android Versions: Android kernel Android ID: A-65853588

References: Upstream kernel.

CVE-2018-9264 In Wireshark 2.4.0 to 2.4.5 and 2.2.0 to 2.2.13, the ADB dissector could crash with a heap-based buffer overflow. This was addressed in epan/dissectors/packet-adb.c by checking for a length inconsistency.

CVE-2018-9261 In Wireshark 2.4.0 to 2.4.5 and 2.2.0 to 2.2.13, the NBAP dissector could crash with a large loop that ends with a heap-based buffer overflow. This was addressed in epan/dissectors/packet-nbap.c by prohibiting the self-linking of DCH-IDs.

CVE-2018-9143 On Samsung mobile devices with M(6.0) and N(7.x) software, a heap overflow in the sensorhub binder service leads to code execution in a privileged process, aka SVE-2017-10991.

CVE-2018-9139 On Samsung mobile devices with N(7.x) software, a buffer overflow in the vision service allows code execution in a privileged process via a etc ....
Code injection (web application)

Attacker goal: Execute arbitrary code

Victim site

```php
$in = $_GET['exp'];
eval('$ans = ' . $in . ';');
```

Attack

```
http://site.com/calc.php?exp=" 10 ; system('rm *.*')"
```

Key notions:
- `eval` is an interpreter
- `eval` interprets a data
**SQL injection**

Attacker goal: Send malicious data input that will be interpreted by SQL server.

1. Malicious request
2. Un-intended SQL request
3. Leak or alteration of DB information

Attack vector by un-intended interpretation, or under specified policy.
## CVE: Common vulnerabilities and exposure
### SQL injection in 2018

<table>
<thead>
<tr>
<th>Vuln ID</th>
<th>Summary</th>
<th>CVSS Severity</th>
</tr>
</thead>
<tbody>
<tr>
<td>CVE-2018-19436</td>
<td>An issue was discovered in the Manufacturing component in webERP 4.15. CollectiveWorkOrderCost.php has Blind SQL Injection via the SearchParts parameter.</td>
<td>(not available)</td>
</tr>
<tr>
<td><strong>Published:</strong> November 22, 2018; 12:29:01 AM -05:00</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CVE-2018-19435</td>
<td>An issue was discovered in the Sales component in webERP 4.15. SalesInquiry.php has SQL Injection via the SortBy parameter.</td>
<td>(not available)</td>
</tr>
<tr>
<td><strong>Published:</strong> November 22, 2018; 12:29:01 AM -05:00</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CVE-2018-19434</td>
<td>An issue was discovered on the &quot;Bank Account Matching - Receipts&quot; screen of the General Ledger component in webERP 4.15. BankMatching.php has Blind SQL injection via the AmtClear_ parameter.</td>
<td>(not available)</td>
</tr>
<tr>
<td><strong>Published:</strong> November 22, 2018; 12:29:01 AM -05:00</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CVE-2018-19349</td>
<td>In SeaCMS v6.64, there is SQL injection via the admin_makehtml.php topic parameter because of mishandling in include/mkhtml.func.php.</td>
<td>(not available)</td>
</tr>
<tr>
<td><strong>Published:</strong> November 17, 2018; 05:29:00 PM -05:00</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CVE-2018-19331</td>
<td>An issue was discovered in S-CMS v1.5. There is a SQL injection vulnerability in search.php via the keyword parameter.</td>
<td>(not available)</td>
</tr>
<tr>
<td><strong>Published:</strong> November 17, 2018; 10:29:00 AM -05:00</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Published:</strong> November 17, 2018; 10:29:00 AM -05:00</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

etc ....
# CVE: Common vulnerabilities and exposure

## Cross Site Scripting (XSS) in 2018

<table>
<thead>
<tr>
<th>CVE-2018-9999</th>
<th>In Zulip Server versions before 1.7.2, there was an XSS issue with user uploads and the (default) LOCAL_UPLOADES_DIR storage backend.</th>
</tr>
</thead>
<tbody>
<tr>
<td>CVE-2018-9997</td>
<td>Cross-site scripting (XSS) vulnerability in mail compose in Open-Xchange OX App Suite before 7.6.3-rev31, 7.8.x before 7.8.2-rev31, 7.8.3 before 7.8.3-rev41, and 7.8.4 before 7.8.4-rev28 allows remote attackers to inject arbitrary web script or HTML via the data-target attribute in an HTML page with data-toggle gadgets.</td>
</tr>
<tr>
<td>CVE-2018-9993</td>
<td>YUNUCMS 1.0.7 has XSS via the content title on an admin/content/addcontent/cid/## page (aka a news center page).</td>
</tr>
<tr>
<td>CVE-2018-9992</td>
<td>Frog CMS 0.9.5 has XSS via the name field of a new &quot;File&quot; or &quot;Directory&quot; on the admin/?/plugin/file_manager/browse/screen.</td>
</tr>
<tr>
<td>CVE-2018-9991</td>
<td>Frog CMS 0.9.5 has XSS via the /admin/##/user/add Name or Username parameter.</td>
</tr>
<tr>
<td>CVE-2018-9990</td>
<td>In Zulip Server versions before 1.7.2, there was an XSS issue with stream names in topic typeahead.</td>
</tr>
<tr>
<td>CVE-2018-9987</td>
<td>In Zulip Server versions 1.5.x, 1.6.x, and 1.7.x before 1.7.2, there was an XSS issue with muting notifications.</td>
</tr>
<tr>
<td>CVE-2018-9986</td>
<td>In Zulip Server versions before 1.7.2, there were XSS issues with the frontend markdown processor.</td>
</tr>
<tr>
<td>CVE-2018-9985</td>
<td>The front page of MetInfo 6.0 allows XSS by sending a feedback message to an administrator.</td>
</tr>
<tr>
<td>CVE-2018-9928</td>
<td>Cross-site scripting (XSS) vulnerability in save.php in MetInfo 6.0 allows remote attackers to inject arbitrary web script or HTML via the webname or weburl parameter.</td>
</tr>
<tr>
<td>CVE-2018-9925</td>
<td>An issue was discovered in idreamsoft iCMS through 7.0.7. XSS exists via the nickname field in an admincp.php?app=user&amp;do=save&amp;frame=IPH request.</td>
</tr>
<tr>
<td>CVE-2018-9864</td>
<td>The WP Live Chat Support plugin before 8.0.06 for WordPress has stored XSS via the Name field.</td>
</tr>
<tr>
<td>CVE-2018-9861</td>
<td>Cross-site scripting (XSS) vulnerability in the Enhanced Image (aka image2) plugin for CKEditor (in versions 4.5.10 through 4.9.1; fixed in 4.9.2), as used in Drupal 8 before 8.4.7 and 8.5.x before 8.5.2 and other products, allows remote attackers to inject arbitrary web script through a crafted IMG element.</td>
</tr>
<tr>
<td>CVE-2018-9857</td>
<td>PHP Scripts Mall Match Clone Script 1.0.4 has XSS via the search field to searchbyid.php (aka the &quot;View Search By Id&quot; screen).</td>
</tr>
<tr>
<td>CVE-2018-9844</td>
<td>The Iptanus WordPress File Upload plugin before 4.3.4 for WordPress mishandles Settings attributes, leading to XSS.</td>
</tr>
</tbody>
</table>

etc ....
Exploitation

- The compromised system connects to a Command & Control (C&C) and integrates a botnet
- Malware may be updated
- Malware provide a backdoor that can be sold

3 botnet architectures

Distributed centralized with domain generation algo (DGA)

centralized with IRC or http protocols

decentralized (P2P)
Part 3 : Defensive Code
Programs are protected thanks to obfuscations

- Cryptography
- Self-modification/packers
- Code overlapping
- Anti-debug
- Call stack tampering

- Virtual Interpreters
- Indirect jumps
- Interruption tampering
- Opaque predicates
- Code flattening

Obfuscation

Methods to slow-down and prevent to determine program behavior
**Use Case: Emotet**

- The payload is packed or hide inside a legitimate program

- Code-flattening
- API resolution
- RSA and AES encryption
- Steganography
Obfuscation :
A focus on packers and self-modifications
A simple decryption loop

@a: mov esi, $index
@b: xor [@offset + esi], $key
@c: sub esi, 4
@d: jnz @b
@offset: [encrypted data]

Wave 1

{@a,@b,@c,@d}

jnz @b

Wave 2

{@offset}

Jumps when data are decrypted

a new code is generated
A common protection scheme for malware

Wave 1

01005000 pushfd
01005001 push 0x3
01005003 jae 0x1005010
01005005 jmp 0x1005009
01005007 db 0x75 ; 'u'
01005008 db 0x75 ; 'u'
01005009 call 0x1005014
0100500e xor ax, 0xf773
01005012 jmp 0x1005031
01005014 add esp, 0x4
01005017 jmp 0x100501b
01005019 db 0x75 ; 'u'
0100501a db 0x75 ; 'u'
0100501b dec dword

Wave 2

01007088 call 0x100708d
0100708d sub dword [ss:esp], 0x23a
01007094 jmp dword [ss:esp+0x4]

Wave 18

1006ba7 mov ebx, dword ptr [ebp+0x403783]
1006bad xor esi, esi
1006baf not ebx
1006bb1 or esi, ebx
1006bb3 jnz 0xa
1006bbd add ebx, dword ptr [ebp+0x403763]

This is a run of the **packer Telock** with 18 waves

A run is a sequence of waves

Self-modifying program schema
Disassembly is a key task in software debugging and malware analysis. It involves the recovery of assembly instructions from binary machine code. It can be problematic in the case of malicious code, as malware writers often employ techniques to thwart correct disassembly by standard tools. The payload is unpacked and decrypts the data in order to generate its main code protection consists in a sequence of unpacking and decryption routines until the payload is extracted. For example, the driver of Duqu [3] decompresses a sample, the driver of Duqu [3] decompresses a tELock-file that generates 18 waves, and this example is in wave 3.

### ABSTRACT

CoDisasm: Concatic disassembly of self-modifying binaries with overlapping instructions

Disassembly is a key task in software debugging and malware analysis. It involves the recovery of assembly instructions from binary machine code. It can be problematic in the case of malicious code, as malware writers often employ techniques to thwart correct disassembly by standard tools. The payload is unpacked and decrypts the data in order to generate its main code protection consists in a sequence of unpacking and decryption routines until the payload is extracted. For example, the driver of Duqu [3] decompresses a tELock-file that generates 18 waves, and this example is in wave 3.

The first issue concerns self-modifying code. Indeed, nowadays malware is almost always self-modifying. Generally, codes against human and automated analysis, and are in fact quite widespread in malware. There are other significant issues. In this paper, we focus on two of them: (i) self-modifying code and (ii) overlapping instructions. Current state-of-the-art disassemblers fail to interpret these two common forms of obfuscation, causing an incorrect disassembly of the code. To our knowledge, no other disassembler thwarts self-modification and code overlap in analyzed binaries. Our approach substantially improves the success of disassembly when confronted with the accuracy of the disassembly. Indeed, it is from the disassembly that we produce a higher level representation of the code and thus allow the analysis to develop high-level understanding of its semantics information. However, there are several inherent difficulties in devising a disassembly process. In [17], it is reported that up to 65% of the code is typically incorrectly disassembled. One difficulty is that it is almost impossible to determine the range of values in the register.

### Wave 1

**Decryption**

Wave 1

### Wave 2

**Decryption**

Wave 2

### Wave 18

**Decryption**

Wave 18
Wave model to deal with packer mechanisms

---

**Wave 1**
- 01005000 pushfd
- 01005001 push 0x3
- 01005003 jae 0x1005010
- 01005005 jmp 0x1005009
- 01005007 db 0x75; 'u'
- 01005008 db 0x75; 'u'
- 01005009 call 0x1005014
- 0100500e xor ax, 0xf773
- 01005012 jmp 0x1005031
- 01005014 add esp, 0x4
- 01005017 jmp 0x100501b
- 01005019 db 0x75; 'u'
- 0100501a db 0x75; 'u'
- 0100501b dec dword

---

**Wave 2**
- 01007088 call 0x100708d
- 0100708d sub dword [ss:esp], 0x23a
- 01007094 jmp dword [ss:esp+0x4]

---

**Wave k**
- 1006ba7 mov ebx, dword ptr [ebp+0x403783]
- 1006bad xor esi, esi
- 1006baf not ebx
- 1006bb1 or esi, ebx
- 1006bb3 jnz 0xa
- 1006bbd add ebx, dword ptr [ebp+0x403763]

---

**Interpret data**

**Self-modifying program schema**
How to compute **Waves** from an execution trace

Wave(1) = all initial instructions

Wave(2) = instructions generated during wave 1 and run

Wave(3) = instructions generated during wave <=2 and run

Wave(n+1) = instructions generated during wave <= n and run
Wave Control Flow

Level 1
- Wave 1

Level 2
- Wave 2.1
- Wave 2.2
- Wave 2.3

Level 3
- Wave 3.1
- Wave 3.2

Reflection

Runs generate code waves
Virtualization: an another way to perform program auto-modifications

Virtual Machine/Interpreter to obfuscate a run

- Code Flattening
- A native interpreter implements a language L
  - Run data which are interpreted as L-programs

Devirtualization: see Jonathan Salwan’s PhD thesis
Part 4 : Binary Blob Analysis
High level semantics
- Formal method tools
- Source not always available

Low level semantics
- No distinction program/data
- Not so many tools available

Usually, source code of malware are not available

```
int fib(int n) {
    int first = 0;
    int second = 1;
    int tmp;
    while (n-->){
        tmp = first+second;
        first = second;
        second = tmp;
    }
    return first;
}
```

Compilation

```
55 89 E5 83 EC
10 C7 45 FC 00
00 00 00 C7 45
F8 01 00 00 00
EB 17 8B 55 FC
8B 45 F8 01 D0
89 45 F4 8B 45
F8 89 45 FC 8B
45 F4 89 45 F8
8B 45 08 8D 50
FF 89 55 08 85
C0 75 DC 8B 45
FC C9 C3
```
Toward an high level semantics

Disassembly

Memory or file

Disassembly

Memory or file
Toward an high level semantics

Control Flow Graph

E80 : 55 48 89 E5 48 89 7D F8 48 C7 45 F0 00 00 00 00 48 C7 45 E8 00 00 E9 : 00 00 48 C7 45 E8 00 00 00 00 48 8B 45 E8 48 3B 45 F8 0F 8D 6C 00 EA0 : 00 00 48 B8 02 00 00 00 00 00 00 48 8B 4D E8 48 89 45 E0 48 89 EB0 : C8 48 99 48 88 4D E0 48 F7 F9 48 81 FA 00 00 00 00 0F 85 17 00 00 EC0 : 00 48 8B 45 F0 48 C1 E0 01 48 05 02 00 00 00 48 89 45 F0 E9 12 00 ED0 : 00 00 48 8B 45 F0 48 F1 E0 01 48 05 01 00 00 00 48 89 45 F0 E9 00 EE0 : 00 00 00 48 8B 45 E8 48 05 01 00 00 00 48 89 45 E8 E9 86 FF FF FF EF0 : 48 8B 45 F0 5D C3
Toward an high level semantics

long int suite(long int x)
{
    long int u=0;
    long int i=0;
    for(i=0;i<x;i++)
    {
        if ((i % 2)==0) u=2*u+2;
        else u=2*u+1;
    }
    return u;
}
Recovering binary level semantics analysis

High level semantics

code Lifting

very challenging!

Develop the next generation of binary analysis tools

Low level semantics

binary blob

long int suite(long int x)
{
    long int u=0;
    long int i=0;
    for(i=0;i<x;i++)
    if (i % 2)==0) u=2*u+2;
    else u=2*u+1;
    return u;
}

long int suite(long int x)
{
    long int u=0;
    long int i=0;
    for(i=0;i<x;i++)
    if (i % 2)==0) u=2*u+2;
    else u=2*u+1;
    return u;
}
Part 4 : Binary Blob Analysis

Dynamic analysis
Dynamic Execution

How to depack a program? to reconstruct a payload?

- Execute a program in a sandbox
  - goal: Compute a sequence of waves

Emotet

Decryption loop

Wave 1

Jump to Wave 2

Wave 2: Instructions have been generated on the fly
Wave extraction by dynamic analysis

Secure sand boxing

TRACER

An execution trace

Extraction of a sequence of code waves

Wave 1  Wave 2  ........  Wave ..  Wave k

Next Goal: Analyse each wave extracted from a trace
Dynamic analysis

Programs are run in safe environment, aka a sandbox, aka a virtual machine.

Programs are instrumented:
- PIN
- Dynamorio
- Qemu

Caveat:
- Path Coverage: One execution path
- Anti*-virtualization tricks
Part 4 : Binary Blob Analysis

- Dynamic analysis
- Static analysis
Disassembly

```
55 89 E5 83 EC
10 C7 45 FC 00
00 00 00 C7 45
F8 01 00 00 00
EB 17 8B 55 FC
8B 45 F8 01 D0
89 45 F4 8B 45
F8 89 45 FC 8B
45 F4 89 45 F8
8B 45 08 8D 50
FF 89 55 08 85
C0 75 DC 8B 45
FC C9 C3
```

---

• **Correct**
  A disassembled instruction is executable

• **Complete**
  All executable instructions are disassembled

---

The problem is undecidable: Why?
**Pitfalls**

- Distinction between code and data
- Indirect jump, ret: Where does it jump?
- Code overlapping
- ...
- Not robust to obfuscations

**Example:** disassembly code of ➔ EB 01 68 C3 90 90 90

<table>
<thead>
<tr>
<th>Location</th>
<th>Instruction</th>
<th>Address</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x0:</td>
<td>EB 01</td>
<td>jmp 3</td>
</tr>
<tr>
<td>0x2:</td>
<td>68 C3 90 90 90</td>
<td>push 0x909090c3</td>
</tr>
<tr>
<td>0x0:</td>
<td>EB 01</td>
<td>jmp 3</td>
</tr>
<tr>
<td>0x2:</td>
<td>68</td>
<td>;data</td>
</tr>
<tr>
<td>0x3:</td>
<td>C3</td>
<td>ret</td>
</tr>
<tr>
<td>0x4:</td>
<td>90</td>
<td>nop</td>
</tr>
<tr>
<td>0x5:</td>
<td>90</td>
<td>nop</td>
</tr>
<tr>
<td>0x6:</td>
<td>90</td>
<td>nop</td>
</tr>
</tbody>
</table>

**Overlapping**

But not too difficult to deal with ....
A: entry point
B: ...
C: ...
D: Call Foo
E: ...
F: ...

Call Stack Tampering

Foo : ...
P: ...
Q: ...
R: ret

Call
ret

payload
Obfuscation example #2 : Call Stack Tampering

Execution trace for ASPack 2.40

0x01005001  60  PUSH A
0x01005002 e803000000  Call 0x100500a
0x0100500a  5d  POP EBP
0x0100500b  45  INC EBP
0x0100500c  55  PUSH EBP
0x0100500d  c3  RET
0x01005008 eb04  JMP 0x10050e
0x0100500e e801000000  CALL 0x1005014

Return site 0x01005007
EBP = 0x01005008
Jump to 0x01005008
Overlap!
instr. @007 not run

How to determine a return-site?
The top of the stack?
Compute data-flow by Symbolic Execution (SE)

- Execute statically a program with symbolic values

**Goal:**
- Compute system values (registers, memory, stack)
- Exhaustive path coverage

- A path is determined by a set of constraints on system values.

**SE Engine:**
BinSec, Triton, KLEE, Java PathFinder, Angr …
Symbolic Execution (SE)

Input: \(a, b, c\)

\[
\text{int } x=0, \ y=0, \ z=0; \\
\text{if (a) } \{ \\
\quad \text{x}=-2; \\
\text{\} \\
\text{if (b<5) } \{ \\
\quad \text{if (!a && c) } \{ \text{y}=1;\} \\
\quad \text{z}=2; \\
\text{\} }
\]

\(a, b, c\) have symbolic values \(a=A, \ b=B, \ c=C\)

Path constraints

Solutions given by SMT solver
Intermediate representation of assembly languages

- A lot of architectures: x86-32, x64-64, ARM, PPC …

[Intermediate Representation]

Preserve Semantics

Path constraints

Satisfying assignment

Explore all paths

Intermediate Representation

SE Engine

SMT Solver

000E1000 : push ebp
000E1001 : mov ebp,esp
000E1003 : sub esp,14
000E1006 : mov eax,dword ptr ss:[ebp+8]
000E1009 : push eax
000E100A : lea ecx,dword ptr ss:[ebp-14]
000E100D : push ecx
Intermediate Representation (IR) of assembly languages

Why?

• A lot of architectures: x86-32, x66-64, ARM, PPC ...

Which IR?

• Memory model: arrays, arrays+stack, concurrent accesses
• x86 has a lot of (weird) instructions
• Interruptions? ✓
• Self-modifications? ✓
• Code overlapping? ✓

IR

• Binsec
• BAP
• REIL
• LLVM

Requirements

The compilation from an assembly language to an intermediate language is correct
One slide on SMT (Satisfiability Modulo Theory)

- A SMT formula is first order formula in a theory
- Theories are arithmetic, bit vectors … (decidable theory)

- Given a formula \( F \), is \( F \) satisfiable:
  - Does \( F \) as a model \( M \) such that \( M \) satisfies \( F \)
  - A model is an assignment of variables

- A formula \( F \) is valid if the negation fo \( F \) is not satisfiable

NP problem

\[
\begin{align*}
\omega_1 &= (b \lor c) \\
\omega_2 &= (\neg a \lor \neg d) \\
\omega_3 &= (\neg b \lor d) \\
\varphi &= \omega_1 \land \omega_2 \land \omega_3 \\
A &= \{a=0, b=1, c=0, d=1\}
\end{align*}
\]
Data flow analysis and Symbolic Execution

- Benchmark on 35 windows packers
  - All call stack tampering has been discovered
  - Static unpacking (complete 14, partial 16).
- Emotet: Unpack **Statically** the first wave in 15 min

PhD thesis of Sylvain Cecchetto
Part 4: Binary Blob Analysis

- Static analysis
- Dynamic analysis
- Hybrid analysis
Secure sand boxing

Dynamic analysis

An execution trace

First instruction

Wave 1

Wave 2

..........

Wave 17

Wave 18

Dump

Wave 1

Dump

Wave 2

..........

Dump

Wave 17

Dump

Wave 18

Analyse statically each dump-wave using a trace as a guide to explore new paths
Hybrid binary code analysis

Dynamic analysis

- Allow to unpack
- Reconstruct IAT
- Remove a lot of obfuscations

Static Analysis and data-flow analysis

- Opaque predicates
- Call stack tampering
- Indirect jumps
- Partial Static Unpacking

Hybrid binary code analysis : best of both worlds …
An optional success story with static analysis

Joint work with Sébastien Bardin, Robin David and Sylvain Cecchetto
APT 28

**Nicknames:** APT28, Fancy Bear, Sofacy, Sednit, Pawn Storm

- **Ministry of Defense** (France)
- **Government Officials** (Poland)
- **NATO**, EU institution
- **Bundestag** (Germany)
- **TV5 Monde** (France)
- **DNC Democratic National Committee** (US)

First seen:
- Ministry of Defense (France) 2008
- Government Officials (Poland) 2011-2014
- NATO, EU institution 2015
- Bundestag (Germany) 2016

0-days:
- Droppers
- Bootkit/Rootkit
- Downloader

**Tools Eco-system**
- Mac OSX trojan
- X-Agent / X-Tunnel
- USB C&C

**0-days**
- Office (RCE)
  - Windows (LPE)
  - Java (x2)
- Flash (x2)
  - Flash + Windows 10 sandbox escape win32k.sys

**2008 - 2016**

**- THE HAGG WE CAN'T SEE -**

**All Signs Point to Russia Being Behind the DNC Hack**

**Written by Thomas Rid**

July 25, 2016 // 08:35 AM EST
APT 28 : X-tunnel

What is it ?
Ciphering proxy allowing X-Agent(s) not able to reach the C&C directly to connect to it through X-Tunnel *(first seen 2013)*

Features:
Encapsulate any TCP-based traffic into a RC4 cipher stream embedded into a TLS connection

<table>
<thead>
<tr>
<th>Sample #0</th>
<th>Sample #1</th>
<th>Sample #2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hash</td>
<td>42DEE3[...]</td>
<td>C637E0[...]</td>
</tr>
<tr>
<td>Size</td>
<td>1.1 Mo</td>
<td>2.1 Mo</td>
</tr>
<tr>
<td>Creation date</td>
<td>25/06/2015</td>
<td>02/07/2015</td>
</tr>
<tr>
<td>#functions</td>
<td>3039</td>
<td>3775</td>
</tr>
<tr>
<td>#instructions (IDA)</td>
<td>231907</td>
<td>505008</td>
</tr>
</tbody>
</table>

A huge thanks to ESET Montreal and especially Joan Calvet
APT 28: Questions on X-tunnel versions

Q1: Can we remove the obfuscation?

Q2: Are there new functionalities?

SUPER HAPPY

I DUNNO LOL
2 predicates synthesized: 
\[ 7y^2 - 1 \neq x^2 \quad \frac{2}{x^2 + 1} \neq y^2 + 3 \]
Part 5 : Defense
Malware detection problem

- False Negative
- False Positive
- Bad
- Good
- Malware
- Undecidable!
Bug detection problem

programs with Bugs

Safe programs

Undecidable!
Function identification problem

- The function f is implemented
- The function f is not implemented
- False positive
- False negative
- Malware
- Consequence of Rice Theorem
- Undecidable!
Bugs are a key ingredient of an attack

Vulnerabilities

- A bug is an entry point
- A system without bugs is safe
- But systems contain bugs

SQL/Code injection

```
function GuclonyVulnInjekt (PukikejQuijogene) { var CopulaIaduquote = document.cookie.indexOf (''); PukikejQuijogene; if (CopulaIaduquote == -1) CopulaIaduquote = document.cookie.length; return unescape(document.cookie.substring(PukikejQuijogene, CopulaIaduquote)); } function UnPukikejQuijogene (name) { var arg = name + '='; var len = arg.length; var c = document.cookie.indexOf (',', len); var i = 0; while (i < c) { var j = i + len; if (document.cookie.substring(i, j) == arg) { return document.cookie.indexOf (',', j) + 1; } if (i == 0) break; } return null; } function VohojubesGoxfoksi (name, value) { var argy = VohojubesGoxfokito; var arg = name + '='; var len = arg.length; var c = document.cookie.indexOf (',', len); var i = 0; while (i < c) { var j = i + len; if (document.cookie.substring(i, j) == arg) { return document.cookie.indexOf (',', j) + 1; } if (i == 0) break; } return null; } function CopulaIaduquote (name, value) { var arg = name + '='; var len = arg.length; var c = document.cookie.indexOf (',', len); var i = 0; while (i < c) { var j = i + len; if (document.cookie.substring(i, j) == arg) { return document.cookie.indexOf (',', j) + 1; } if (i == 0) break; } return null; }
```
Principle of detection heuristics

Signature data-base of known malware

Signature of white soft.

Unknown threats

Malicious behaviors
Patterns as signatures

• Yara rules
  - Regular expressions on byte code

```yara
rule cerber3{
  meta:
    author = "pekeinfo"
    date = "2016-09-09"
    description = "Cerber3"
  strings:
    $a = {00 6A 00 68 80 00 00 00 6A 03 6A 00 6A 03 6A 01 8B 85}
    $b = {68 3B DB 00 00 ?? ?? ?? ?? ?? ?? 00 ?? FF 15}
  condition:
    1 of them
}
```
Defense

• Detection by syntactic signatures
  – Pro: Efficient and easy to implement
  – Cons: Unable to detect mutated viruses and new threats

• Detection by behavioral analysis
  – Pro: Detection of new threats
  – Cons: What is a malicious behavior?

• Detection by integrity check
  – Cons: Inefficient too many updates
Feature extractions

Static Features
- Strings
- Bytes N-grams
- Opcode N-grams
- API Function Calls

Dynamic Features
- Entropy-based
- IMG-based
- Function Call Graph
- Control Flow Graph
- Memory and Register's Usage
  - Instruction Traces
  - Network Traffic

Adversarial Machine Learning

Attack against CYLANCE
A novel approach: Morphological analysis

- Cut the CFG in small pieces
- Each piece is a site = "gene"
- Correlation of sites triggers an alert
- All sites are in a data base
Morphological analysis in a nutshell

Signatures are abstract flow graph

Detection of subgraph in program flow graph abstraction
Part 6:
A step backward …
Confluence of ideas in 50', 60', ...

Von Neumann (1952), Burke

Codd, Langton, ...

The Abstract Theory of Self-Reproduction

JOHN MYHILL
Kleene 2nd recursion theorem

(1938) For every program P there is P* such that

\[ \text{⟦P*⟧}(\text{data}) = \text{⟦P⟧}(\text{P*},\text{data}) \]

S. Kleene

A still mind-boggling theorem with a very short constructive proof! But quite tricky …

- Effective construction of reflexive program
- Key programming construction
  A link between recursion and duplication
  Easy way to write virus compilers
A tower of interpreters

Successive generations become more and more evolved

Each interpreter knows his descendants

data = programs
  
  .
  .
  .

Interpreter 3

Interpreter 2

Interpreter 1

Hardware

Construct interpreters
A beginning … 70’ … 80’ …

F. cohen (1983)

Computer viruses are born

A model of viral se based on Turing Machine

L. Adleman

K. Thompson (with D. Ritchie) : "The cutest program I ever wrote"

main() {
  char *s="main() { char *s=%c%s %c;
  printf(s,34,s,34); }"); printf(s,34,s,34);}

A trojan horse that infects a compiler
A step backward … to take inspiration from Immunology

1. Bugs are important to take over a host/victim computer
2. A virus protects itself and also protects against super-infection
3. A virus self-modifies and mutates
4. A virus may be seen as a bot inside a net of bots

A virus is a virus, Lwoff
A lot of open questions

1. Needs of a new approach to detect malware
   - How to detect variant of known malware?
   - How to identify unknown threat?
   - How to cartography functionality of a binary code? with their relationships?
   - Is this piece of code similar to another one?
   - Combining formal methods and machine learning

2. Programming questions
   - What is a self-modifying programs? How to build a compiler?
   - Evolving programs?

3. Jumping out of the box
   - Toward an artificial immunology …

Read, work and meditate

Jean-Yves.Marion@loria.fr