On detection methods and analysis of malware

Jean-Yves Marion
LORIA

On detection methods and analysis of malware

1. A quick tour of Malware detection methods

2. Behavioral analysis using model-checking

3. Cryptographic function identification

What is a malware ?

What is a malware ?

- A malware is a program which has malicious intentions

What is a malware ?

- A malware is a program which has malicious intentions
- A malware is a virus, a worm, a botnet ...

What is a malware ?

- A malware is a program which has malicious intentions
- A malware is a virus, a worm, a botnet ...
- Giving a mathematical definition is difficult

What is a malware ?

- A malware is a program which has malicious intentions
- A malware is a virus, a worm, a botnet ...
- Giving a mathematical definition is difficult
    - How to protect a system ?

What is a malware ?

- A malware is a program which has malicious intentions
- A malware is a virus, a worm, a botnet ...
- Giving a mathematical definition is difficult
  - How to protect a system ?
  - How to detect a malware ?

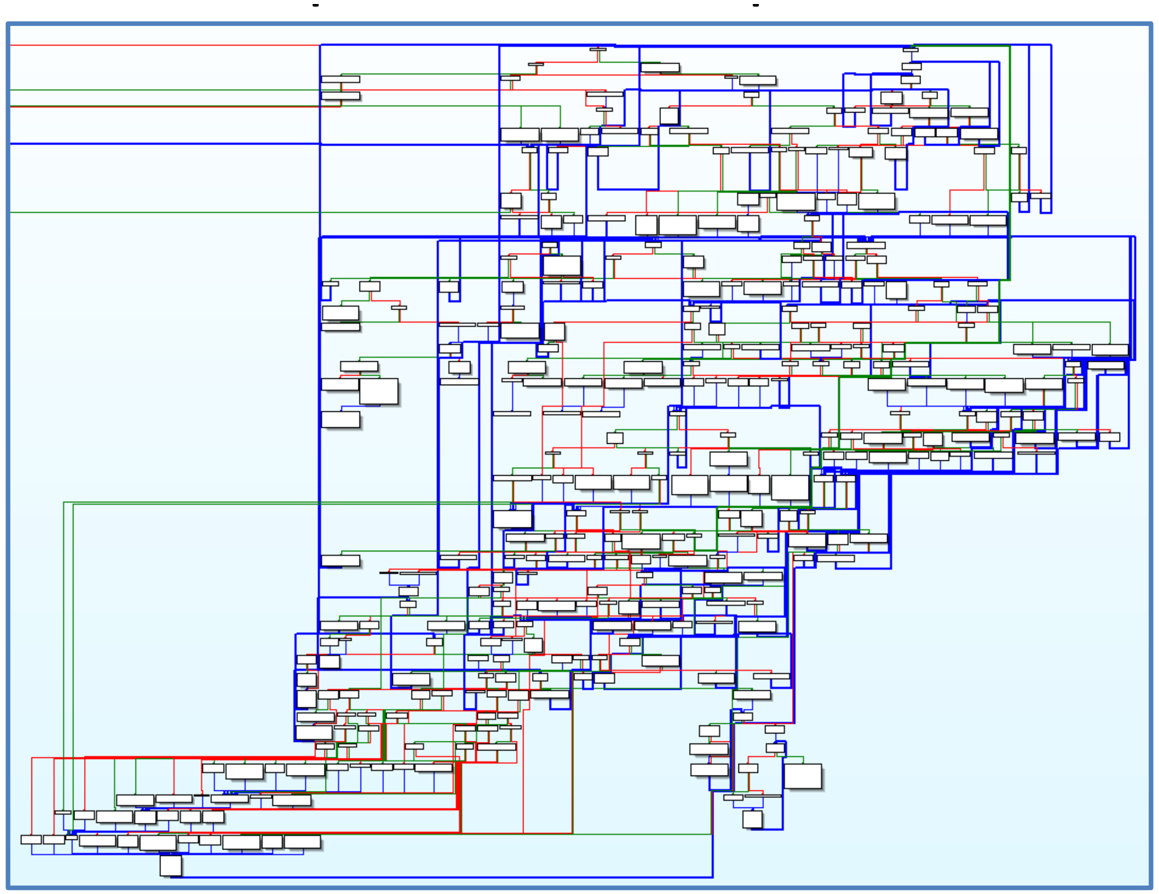
What is a malware ?

- A malware is a program which has malicious intentions
- A malware is a virus, a worm, a botnet ...
- Giving a mathematical definition is difficult
  - How to protect a system ?
  - How to detect a malware ?

Code protection

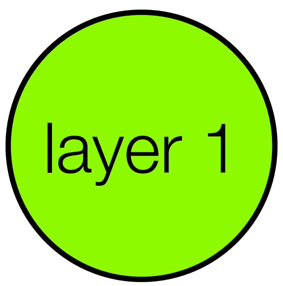
Detection is hard because malware are protected

1. Obfuscation
2. Cryptography
3. Self-modification
4. Anti-analysis tricks



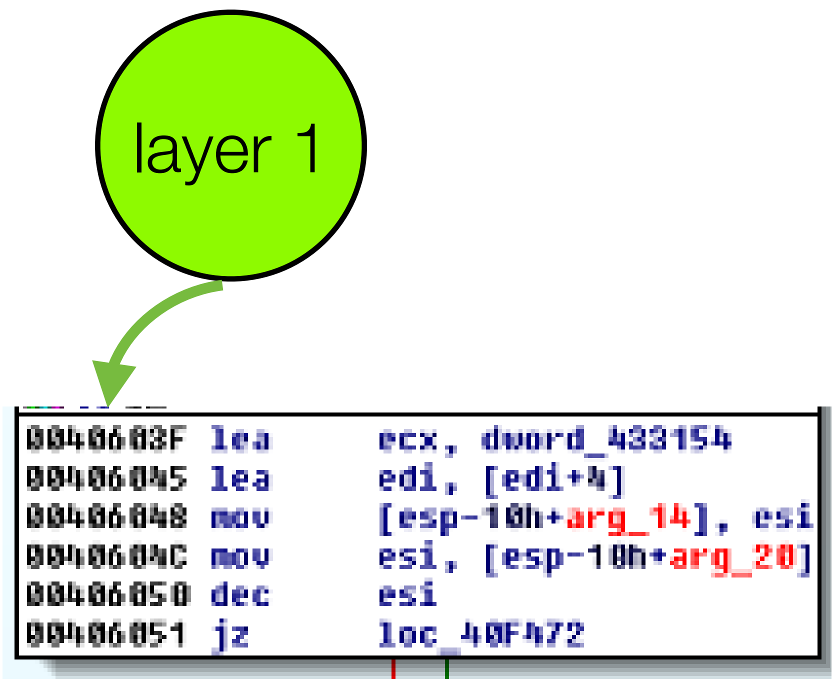
Win32.Swizzor Packer
displayed by IDA

A common protection scheme for malware



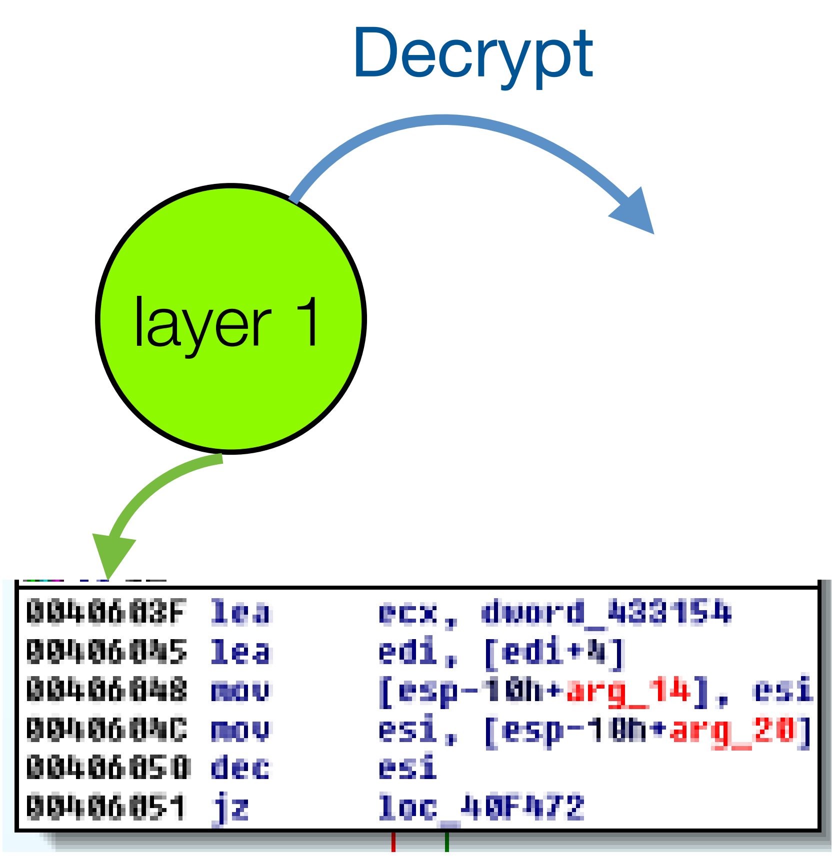
Slide à remplacer ?

A common protection scheme for malware



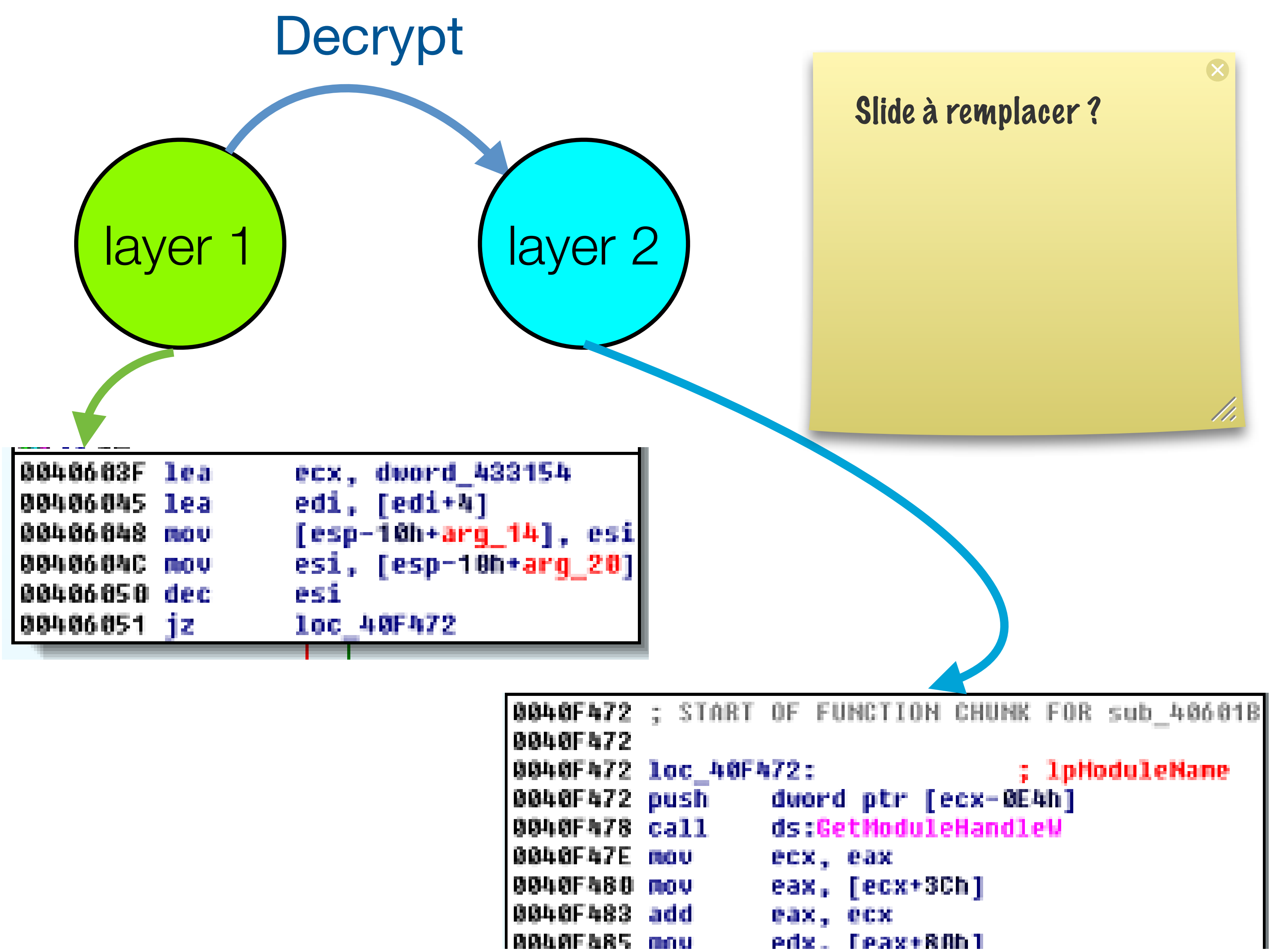
Slide à remplacer ?

A common protection scheme for malware

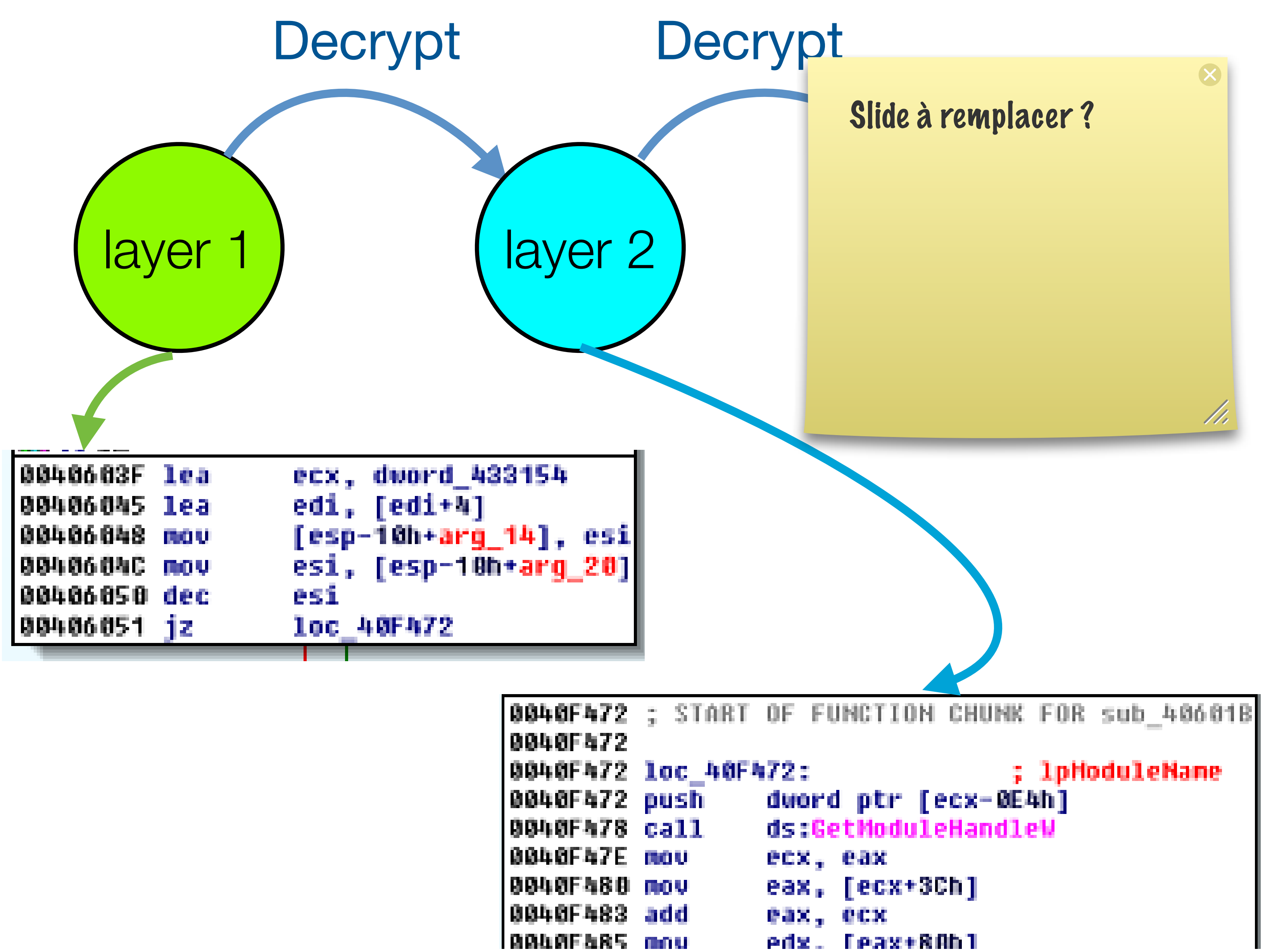


Slide à remplacer ?

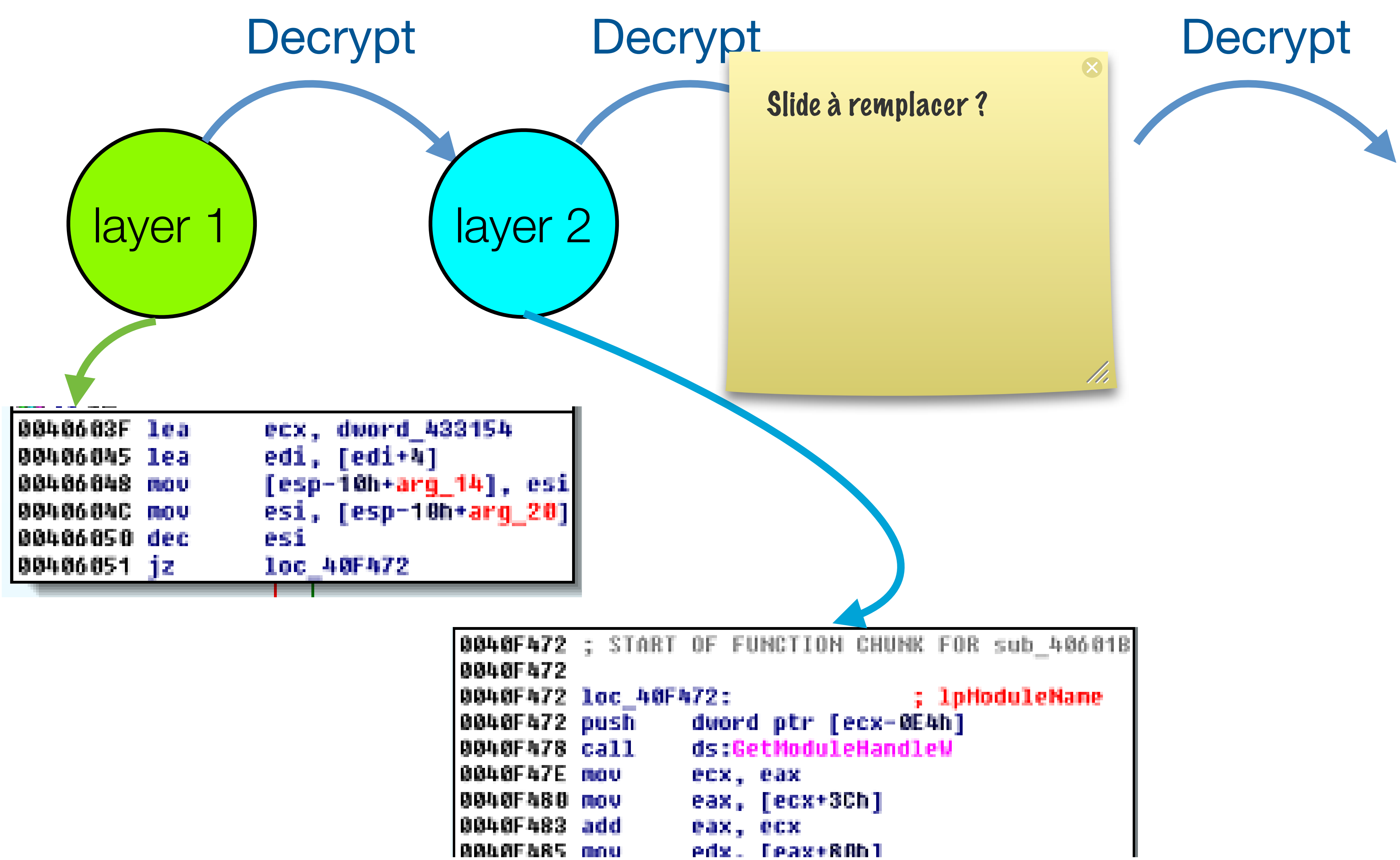
A common protection scheme for malware



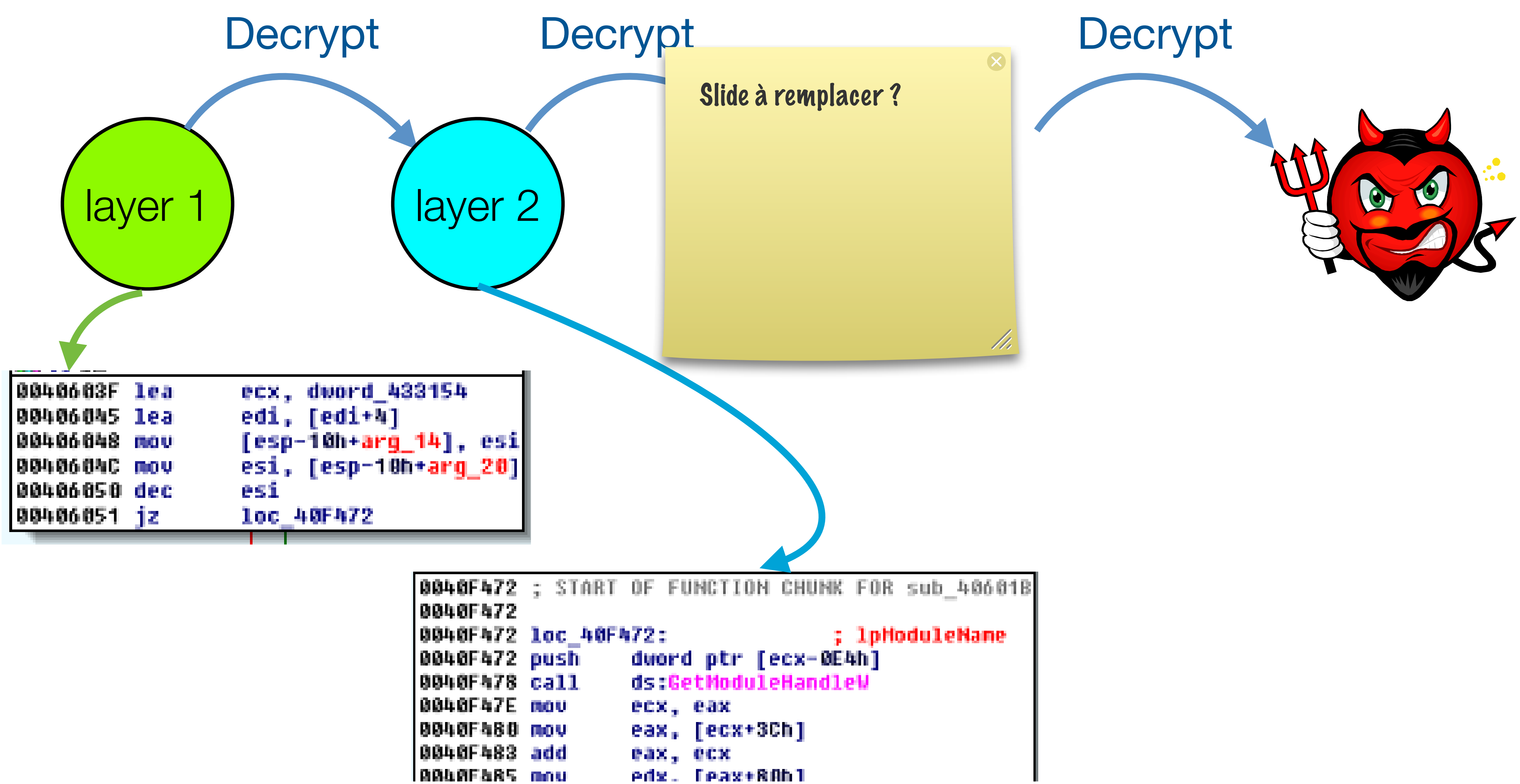
A common protection scheme for malware



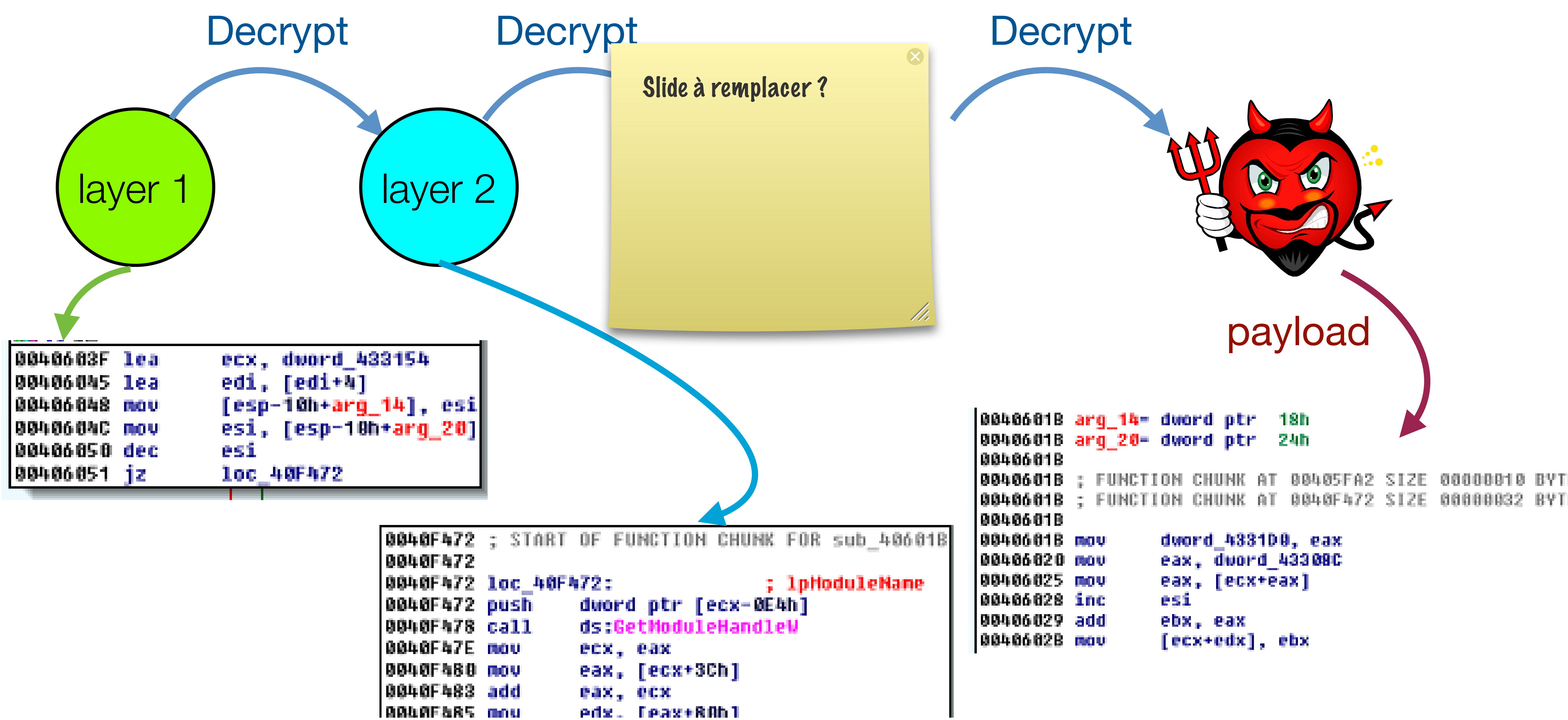
A common protection scheme for malware



A common protection scheme for malware

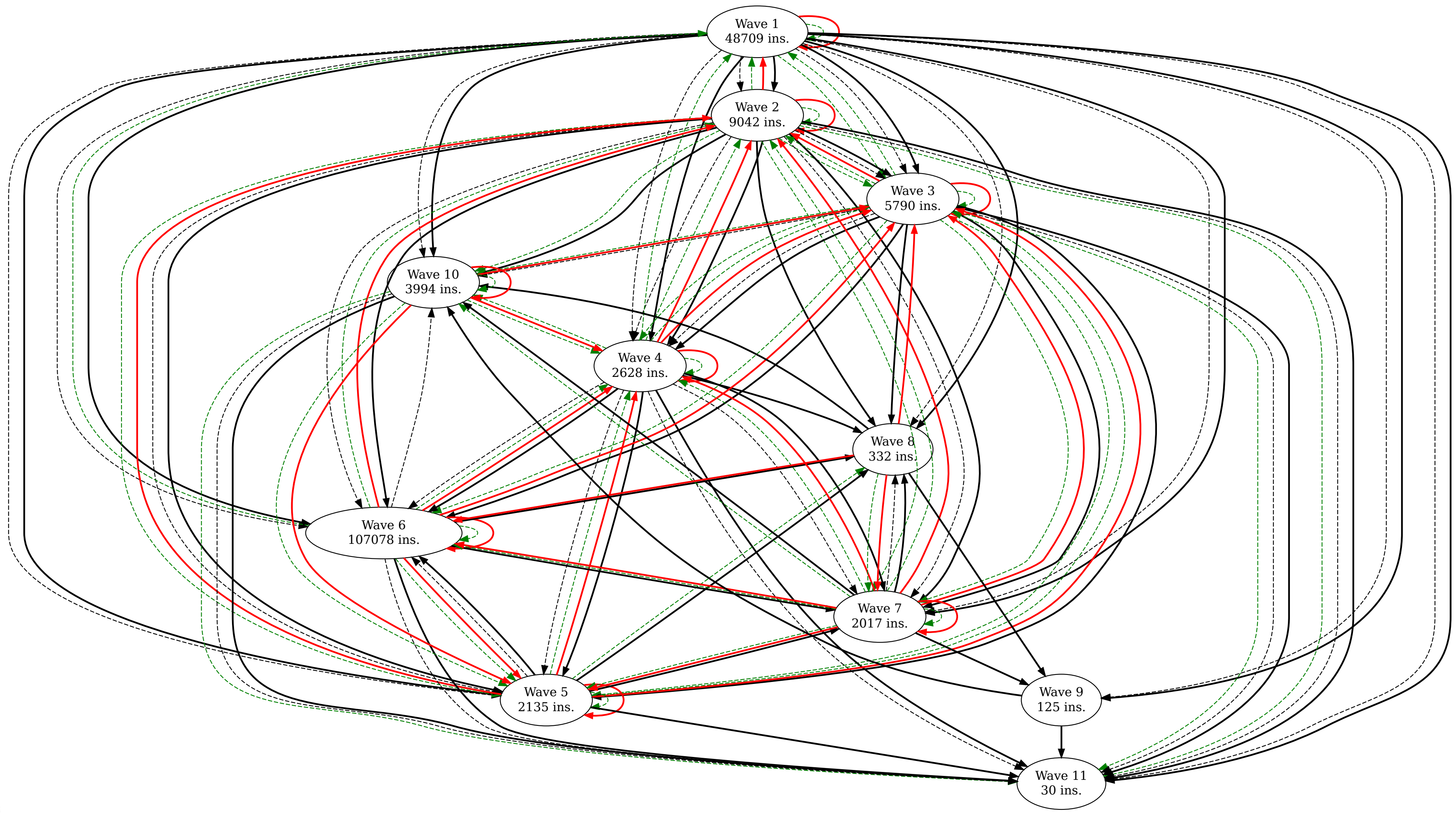


A common protection scheme for malware

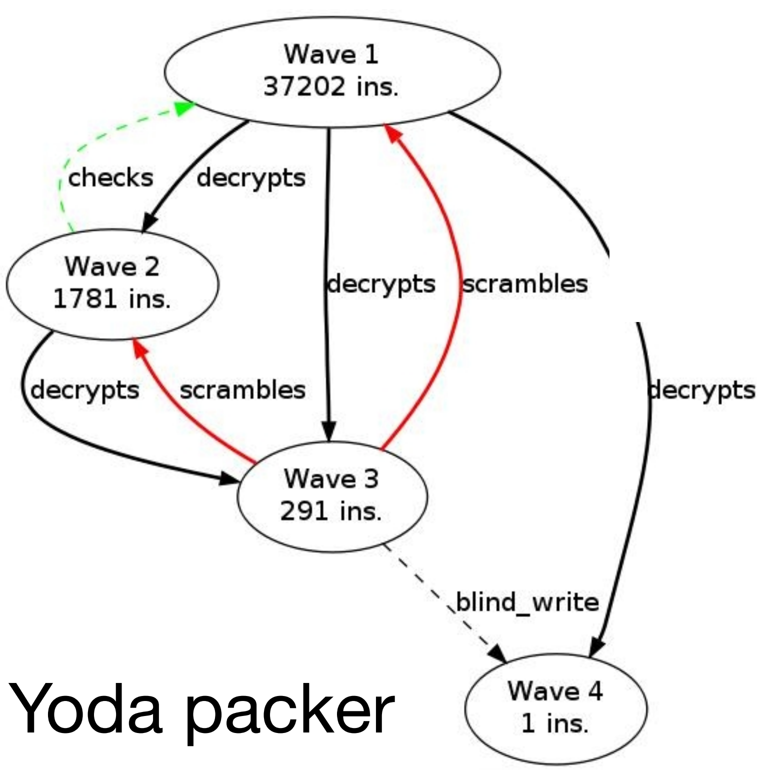


Packer protections

Different code waves with their relations



Themida packer



Yoda packer

Malware detection by string scanning

- Signature is a regular expression denoting a sequence of bytes

Pros :

- Accuracy: low rate of false positive

  ➡ programs which are not malware are not detected

- Efficient : Fast string matching algorithm

  ➡ Karp & Rabin, Knuth, Morris & Pratt, Boyer & Moore

Malware detection by string scanning

- Signature is a regular expression denoting a sequence of bytes

Pros :

- Accuracy: low rate of false positive

  ➡ programs which are not malware are not detected

- Efficient : Fast string matching algorithm

  ➡ Karp & Rabin, Knuth, Morris & Pratt, Boyer & Moore

Cons :

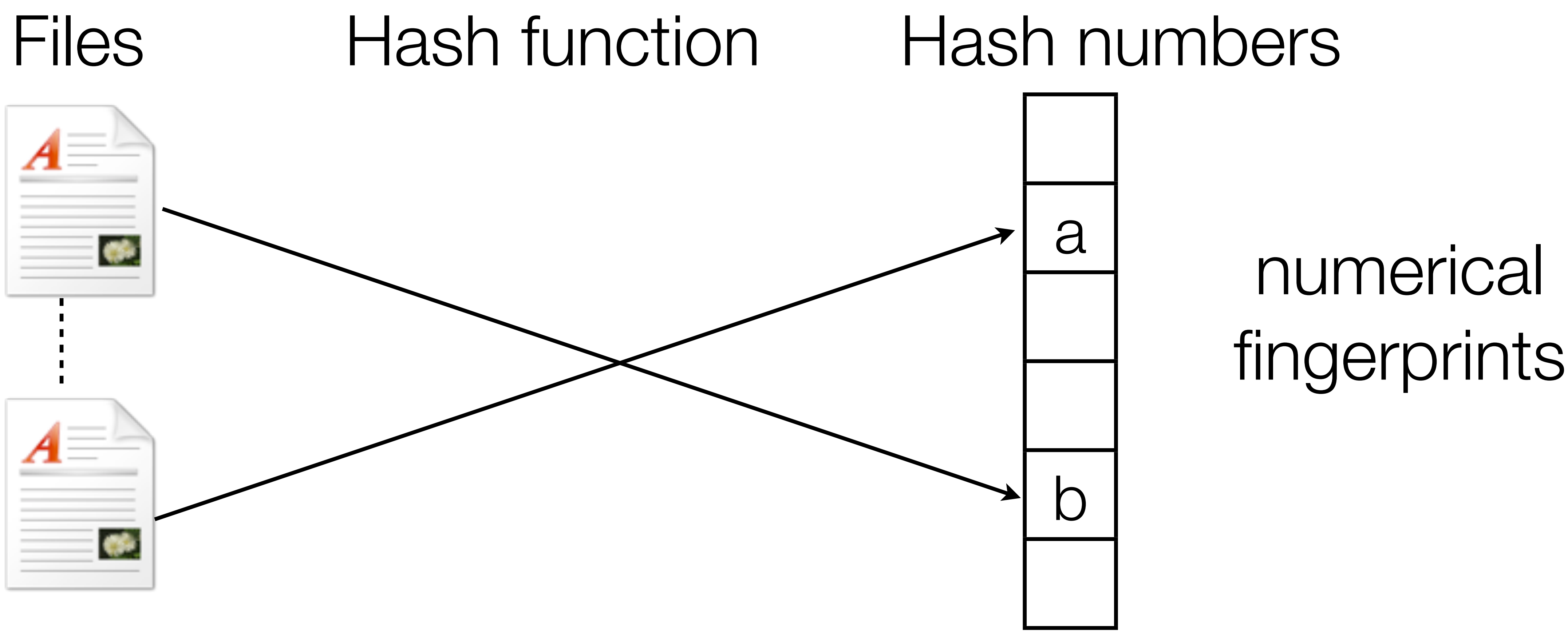
- Signature are quasi-manually constructed
- Signatures are not robust to malware protections

  ➡ Mutations, Code obfuscations, ...

- Static analysis of binary is very difficult

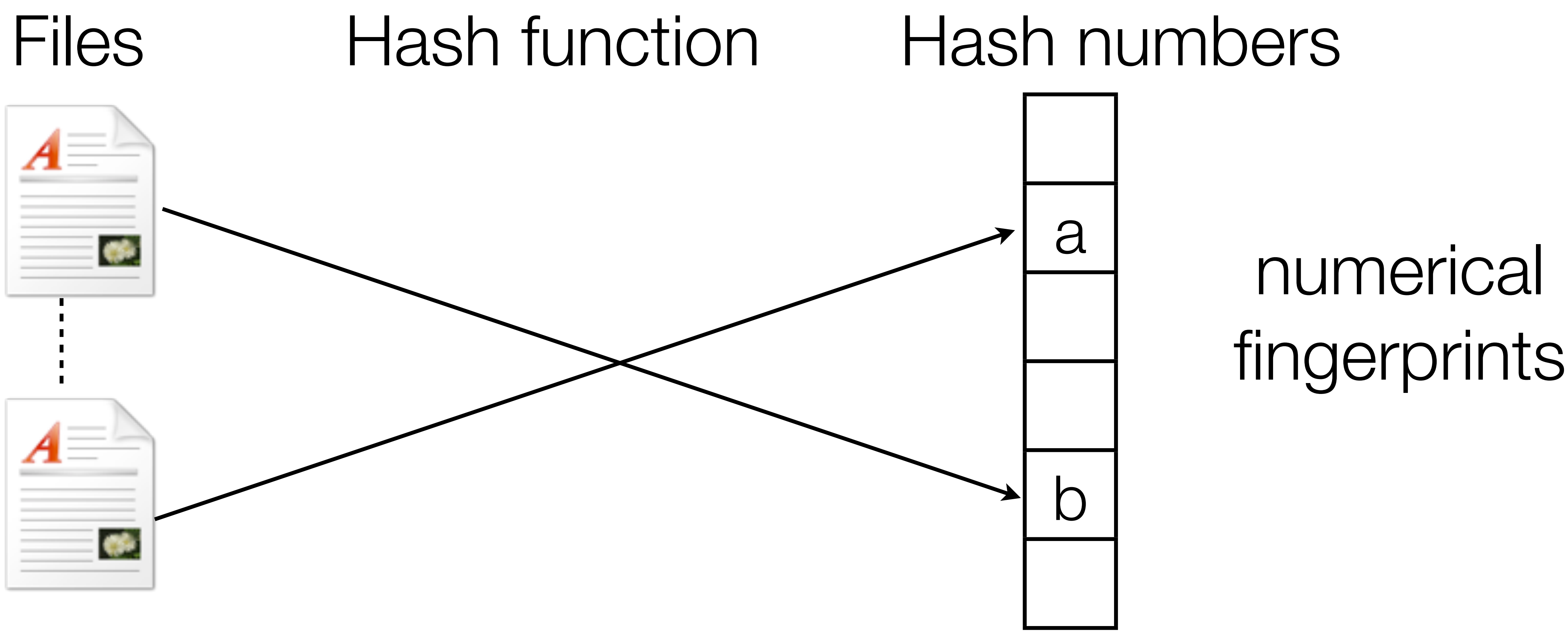
Detection by integrity check

- Identify a file using a hash function



Detection by integrity check

- Identify a file using a hash function



Cons :

- File systems are updated, so numerical fingerprints change
- Difficult to maintain in practice
- Files may change with the same numerical fingerprint (due to hash fct)

Behavioral detection

- Identification of a sequence of actions :
  - System calls or library calls, Network interactions, ...

Behavioral detection

- Identification of a sequence of actions :
  - System calls or library calls, Network interactions, ...
- Two approaches

Behavioral detection

- Identification of a sequence of actions :
  - System calls or library calls, Network interactions, ...
- Two approaches
  - Anomaly Detection from a set of normal behaviours

Behavioral detection

- Identification of a sequence of actions :
  - System calls or library calls, Network interactions, ...
- Two approaches
  - Anomaly Detection from a set of normal behaviours
  - Detection from a set of potential malicious behaviours

Behavioral detection

- Identification of a sequence of actions :
  - System calls or library calls, Network interactions, ...
- Two approaches
  - Anomaly Detection from a set of normal behaviours
  - Detection from a set of potential malicious behaviours

Cons :

- Difficult to have a set of normal or bad behaviours
- Difficult to maintain in practice
- Functional obfuscations :

Behavioral detection

- Identification of a sequence of actions :
  - System calls or library calls, Network interactions, ...
- Two approaches
  - Anomaly Detection from a set of normal behaviours
  - Detection from a set of potential malicious behaviours

Cons :

- Difficult to have a set of normal or bad behaviours
- Difficult to maintain in practice
- Functional obfuscations :

Two ways of writing into a file

```
h=fopen(C:\windows\sys.dll);fwrite(«test»,h)
```

```
h=createFile(C:\windows\sys.dll);writeFile(h,«test»)
```

✓ Several possible implementations of a high level action

Anti-virus tests against unknown threats

Source : A study of anti-virus response to unknown threats by C. Devine and N. Richaud (EICAR 2009)

Product name	**testA01**	**testA02**	**testA03**	**testA11**	**testA12**	**testA13**
avast!	No alert; keys logged.	No alert; keys logged.	No alert; keys logged.	No alert; keys logged.	No alert; keys logged.	No alert; keys logged.
AVG	No alert; keys logged.	No alert; keys logged.	No alert; keys logged.	No alert; keys logged.	No alert; keys logged.	No alert; keys logged.
Avira	No alert; keys logged.	No alert; keys logged.	No alert; keys logged.	No alert; keys logged.	No alert; keys logged.	No alert; keys logged.
BitDefender	No alert; keys logged.	No alert; keys logged.	No alert; keys logged.	No alert; keys logged.	No alert; keys logged.	No alert; keys logged.
ESET	No alert; keys logged.	No alert; keys logged.	No alert; keys logged.	No alert; keys logged.	No alert; keys logged.	No alert; keys logged.

[U] testA01: The GetRawInputData() API was introduced in Windows XP to access input devices at a low level, mainly for DirectX-enabled games. This function was documented in 2008 on the Firewall Leak Tester [6] web site.

[U] testA02 installs a WH_KEYBOARD_LL windows hook to capture all keyboard events (contrary to the WH_KEYBOARD hook, it does not inject a DLL into other processes).

[U] testA03: The GetAsyncKeyState() API allows querying the state of the keyboard asynchronously.

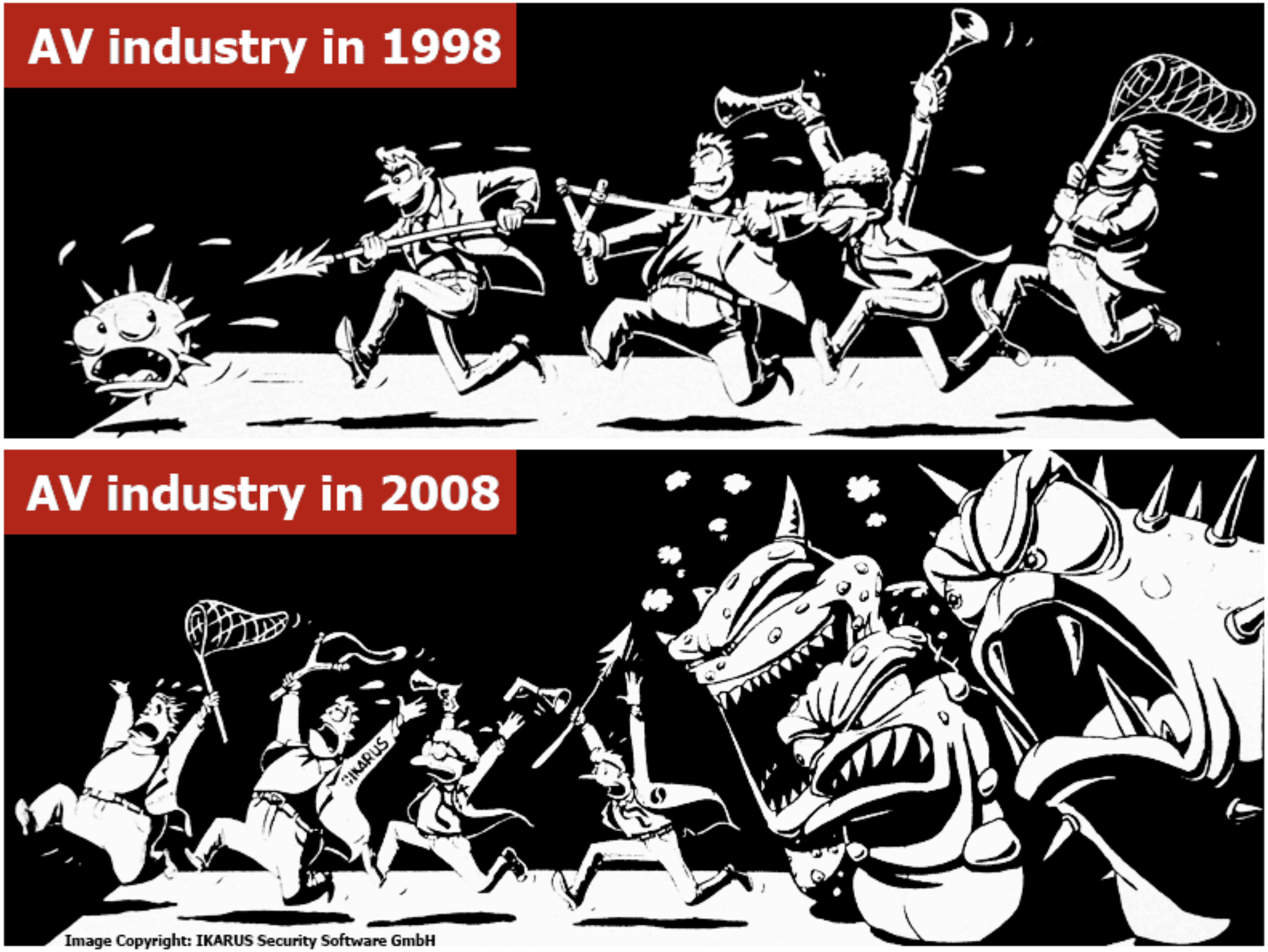
[A] testA11 hooks the keyboard driver's IRJ_MJ_READ function.

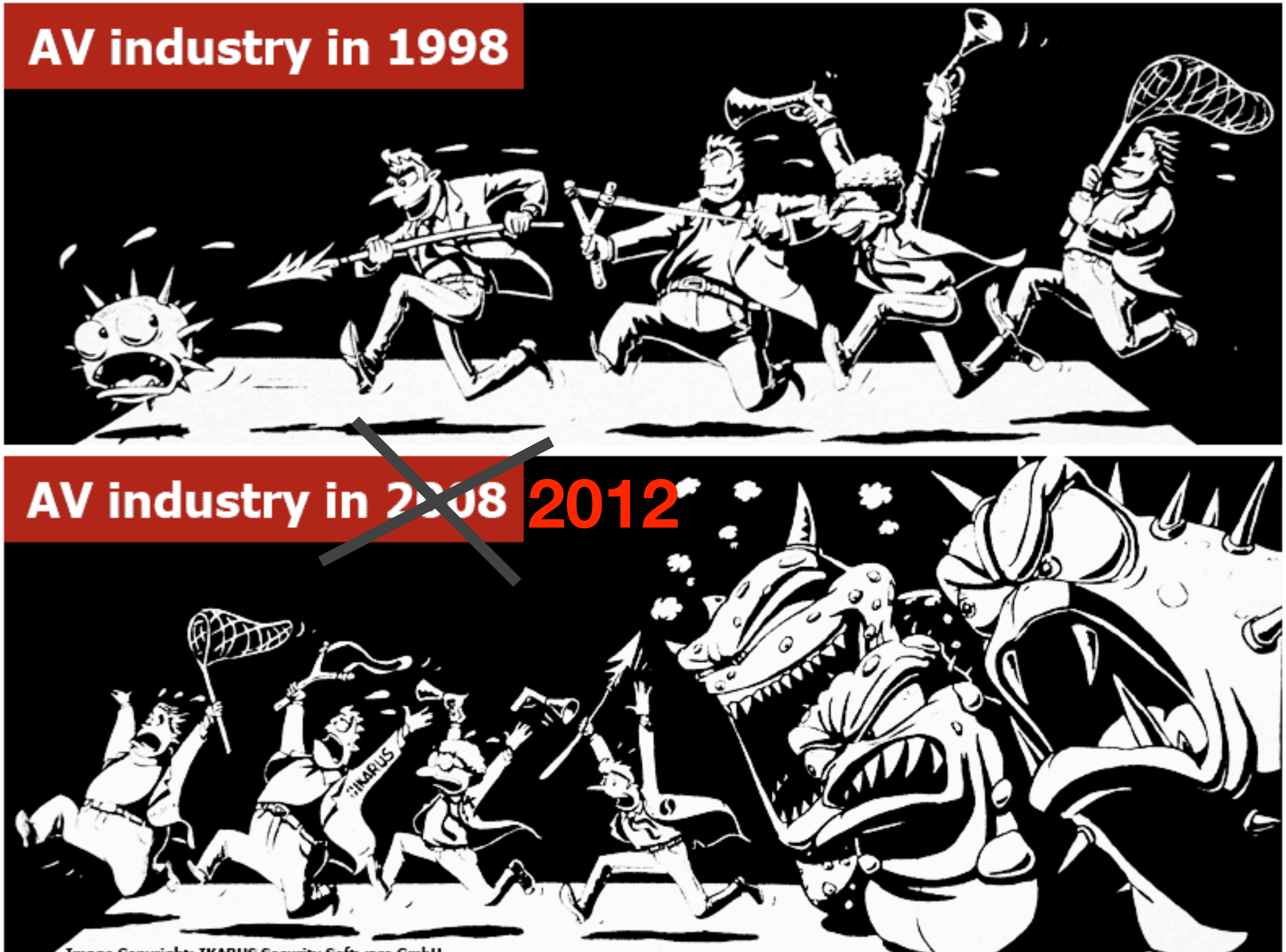
[A] testA12 hooks the keyboard driver's Interrupt Service Request.

[A] testA13 installs a "chained" device driver which places itself between the keyboard driver and upper level input device drivers.

Versions of anti-virus software

Product name	Version tested
avast! professional edition	4.8.1296
AVG Internet Security	8.0.200
Avira Premium Security Suite	8.2.0.252
BitDefender Total Security 2009	12.0.11.2
ESET Smart Security (NOD32)	3.0.672.0
F-Secure Internet Security 2009	9.00 build 149
Kaspersky Anti-Virus For Windows Workstations	6.0.3.837
McAfee Total Protection 2009	13.0.218
Norton 360 Version 2.0	2.5.0.5
Panda Internet Security 2009	14.00.00
Sophos Anti-Virus & Client Firewall	7.6.2
Trend Micro Internet Security Pro	17.0.1305


AV industry in 1998
AV industry in 2008
IKARUS
Image Copyright: IKARUS Security Software GmbH


AV industry in 1998
AV industry in ~~2008~~ 2012
IKARUS
Image Copyright: IKARUS Security Software GmbH

On detection methods and analysis of malware

1. A quick tour of Malware detection methods

2. Behavioral analysis using model-checking

3. Cryptographic function identification

Joint work with Philippe Beaucamps and Isabelle Gnaedig
Esorics 2012

Low level Traces

```
void scan_dir(const char* dir) {
  HANDLE hFind;
  char szFilename[2048];
  WIN32_FIND_DATA findData;

  sprintf(szFilename, "%s\\%s", dir, "*.*");
  hFind = FindFirstFile(szFilename, &findData);
  if (hFind == INVALID_HANDLE_VALUE) return;
  do {
    sprintf(szFilename, "%s\\%s", dir,
            findData.cFileName);
    if (findData.dwFileAttributes
        & FILE_ATTRIBUTE_DIRECTORY)
      scan_dir(szFilename);
    else { ... }
  } while (FindNextFile(hFind, &findData));
  FindClose(hFind);
}

void main(int argc, char** argv) {
  HANDLE hIcmp;
  const char* icmpData = "Babcdef...";
  char reply[128];
```

```
  /* Behavior pattern: ping of a remote host */
  hIcmp = IcmpCreateFile();
  for(int i = 0; i < 2; ++i)
    IcmpSendEcho(hIcmp, ipaddr, icmpData, 10,
                 NULL, reply, 128, 1000);
  IcmpCloseHandle(hIcmp);

  /* Behavior pattern: Netbios connection */
  SOCKET s = socket(AF_INET, SOCK_STREAM, 0);
  struct sockaddr_in sin =
      {AF_INET, ipaddr, htons(139)/* Netbios */};
  if (connect(s, (SOCKADDR*)&sin, sizeof(sin))
      != SOCKET_ERROR) {
    ...
  }

  /* Behavior pattern: scanning of local drives */
  char buffer[1024];
  GetLogicalDriveStrings(sizeof(buffer), buffer);
  const char* szDrive = buffer;
  while (*szDrive) {
    if (GetDriveType(szDrive) == DRIVE_FIXED)
      scan_dir(szDrive);
    szDrive += strlen(szDrive) + 1;
  }
}
```

Allaple.a excerpt

Low level Traces

```
void scan_dir(const char* dir) {
  HANDLE hFind;
  char szFilename[2048];
  WIN32_FIND_DATA findData;

  sprintf(szFilename, "%s\\%s", dir, "*.*");
  hFind = FindFirstFile(szFilename, &findData);
  if (hFind == INVALID_HANDLE_VALUE) return;
  do {
    sprintf(szFilename, "%s\\%s", dir,
            findData.cFileName);
    if (findData.dwFileAttributes
        & FILE_ATTRIBUTE_DIRECTORY)
      scan_dir(szFilename);
    else { ... }
  } while (FindNextFile(hFind, &findData));
  FindClose(hFind);
}

void main(int argc, char** argv) {
  HANDLE hIcmp;
  const char* icmpData = "Babcdef...";
  char reply[128];
```

```
  /* Behavior pattern: ping of a remote host */
  hIcmp = IcmpCreateFile();
  for(int i = 0; i < 2; ++i)
    IcmpSendEcho(hIcmp, ipaddr, icmpData, 10,
                 NULL, reply, 128, 1000);
  IcmpCloseHandle(hIcmp);

  /* Behavior pattern: Netbios connection */
  SOCKET s = socket(AF_INET, SOCK_STREAM, 0);
  struct sockaddr_in sin =
      {AF_INET, ipaddr, htons(139)/* Netbios */};
  if (connect(s, (SOCKADDR*)&sin, sizeof(sin))
      != SOCKET_ERROR) {
    ...
  }

  /* Behavior pattern: scanning of local drives */
  char buffer[1024];
  GetLogicalDriveStrings(sizeof(buffer), buffer);
  const char* szDrive = buffer;
  while (*szDrive) {
    if (GetDriveType(szDrive) == DRIVE_FIXED)
      scan_dir(szDrive);
    szDrive += strlen(szDrive) + 1;
  }
}
```

Allaple.a excerpt

Trace are finite terms:

FindfirstFile(x,y).FindNextFile(z,x). FindNextFile(z,x).FindClose(z).
IcmpSendEcho(u,...).IcmpSendEcho(u,...).IcmpCloseHandle(u)....

Program behaviour

- The program behaviour is given by sequences of system calls
    - represented by a set L of terms
- How to collect traces ?

Program behaviour

- The program behaviour is given by sequences of system calls
  - represented by a set L of terms
- How to collect traces ?

Static analysis

- A good approximation of a set of execution traces
- Good detection coverage
- But static analysis is difficult to perform

Program behaviour

- The program behaviour is given by sequences of system calls
    - represented by a set L of terms
- How to collect traces ?

Static analysis

- A good approximation of a set of execution traces
- Good detection coverage
- But static analysis is difficult to perform

Dynamic analysis

- Collect an execution trace (with use PIN)
- Monitor program interactions (sys calls, network calls, ...)
- What is the detection coverage ? partial behaviours ...

Trace abstraction

Trace abstraction

- Several ways to send a ping :

Trace abstraction

- Several ways to send a ping :

    1. Call the `socket` function with the parameter `IPPROTO_ICMP` and then call the `sendto` function with `ICMP_ECHOREQ`

Trace abstraction

- Several ways to send a ping :

  1. Call the `socket` function with the parameter `IPPROTO_ICMP` and then call the `sendto` function with `ICMP_ECHOREQ`

  2. Call the `IcmpSendEcho` function

Trace abstraction

- Several ways to send a ping :
    1. Call the `socket` function with the parameter `IPPROTO_ICMP` and then call the `sendto` function with `ICMP_ECHOREQ`
    2. Call the `IcmpSendEcho` function
- Abstract the ping behaviour by a predicate `PING(x)` to represent a ping on socket `x`

Trace abstraction

- Several ways to send a ping :

  1. Call the `socket` function with the parameter `IPPROTO_ICMP` and then call the `sendto` function with `ICMP_ECHOREQ`

  2. Call the `IcmpSendEcho` function

- Abstract the ping behaviour by a predicate `PING(x)` to represent a ping on socket `x`

-Define an abstraction relation R as a term rewrite system

socket(x,u).sendto(x,v,y) ⟶ socket(x,u).sendto(x,v,y).PING(x)
IcmpSendEcho(x) ⟶ IcmpSendEcho(x).PING(x)

- We abstract/rewrite a pattern on a trace only once

Trace abstraction

- Several ways to send a ping :

  1. Call the `socket` function with the parameter `IPPROTO_ICMP` and then call the `sendto` function with `ICMP_ECHOREQ`

  2. Call the `IcmpSendEcho` function

- Abstract the ping behaviour by a predicate `PING(x)` to represent a ping on socket `x`

-Define an abstraction relation R as a term rewrite system

`socket`(x,u).`sendto`(x,v,y) ⟶	`socket`(x,u).`sendto`(x,v,y).`PING`(x)
`IcmpSendEcho`(x) ⟶	`IcmpSendEcho`(x).`PING`(x)

- We abstract/rewrite a pattern on a trace only once

-As a result, we have a terminating and rational abstraction system

`IcmpSendEcho`(u,...).`IcmpSendEcho`(u,...).`IcmpCloseHandle`(u)

⟶ `IcmpSendEcho`(u,...).`PING`(u).`IcmpSendEcho`(u,...).`IcmpCloseHandle`(u)

⟶ `IcmpSendEcho`(u,...).`PING`(u).`IcmpSendEcho`(u,...). `PING`(u).`IcmpCloseHandle`(u)

- We keep the LHS to deal with complex patterns

Computation of Abstract Trace language

Abstract a trace language L by reducing it w.r.t. an abstraction relation R

$$L \to \ldots \to L^{\downarrow}$$

Theorem :

Let R be a rational abstraction relation and L be a trace language.

If L is regular then so is $L^{\downarrow}$

- Based on tree automata methods

Related work

- Martignoni et al. 2008: multi-layered abstraction on a single trace

Behaviour patterns

- A behavior pattern is a First-order LTL (Linear temporal logic) formula

$$\varphi_1 = \exists x, y.\, socket\,(x, \alpha) \wedge (\neg closesocket\,(x)\ \mathbf{U}\ sendto\,(x, \beta, y))$$

$$\varphi_2 = \exists x.\, IcmpSendEcho\,(x)$$

$$\varphi_{ping} = \varphi_1 \vee \varphi_2$$

Quantification domain is the finite set of paramerter names

Let L be the behaviour of the program P. If a trace t of L satisfies a behaviour pattern φ, then P has the behaviour described by φ

Behaviour patterns

- A behavior pattern is a First-order LTL (Linear temporal logic) formula

$$\varphi_1 = \exists x, y.\, socket\,(x, \alpha) \wedge (\neg closesocket\,(x)\ \mathbf{U}\ sendto\,(x, \beta, y))$$

$$\varphi_2 = \exists x.\, IcmpSendEcho\,(x)$$

$$\varphi_{ping} = \varphi_1 \vee \varphi_2$$

Quantification domain is the finite set of paramerter names

- Traces satisfying a FO-LTL formula are :

$$B = \{t \in T_{Trace}\,(\mathcal{F}_\Sigma)\ |\ t \models \varphi\}$$

Let L be the behaviour of the program P. If a trace t of L satisfies a behaviour pattern φ, then P has the behaviour described by φ

Malicious behavior detection

Theorem : Let L be a finite set of finite traces. Let $L^{\downarrow}$ be a trace correctly abstracted from a rational abstraction relation R. Let φ be a FOLTL formula. Deciding whether deciding $L^{\downarrow}$ is infected by φ is linear-time computable.

It works also when L is regular (and infinite), see the paper for details

Related work

- Jacob et al., 2009: low-level functionalities, exponential-time detection

A C Keylogger or a sms message leaking app

```c
LRESULT WndProc(HWND hwnd, UINT msg, WPARAM wParam, LPARAM lParam) {
  RAWINPUTDEVICE rid;
  RAWINPUT *buffer;
  UINT dwSize;
  USHORT uKey;

  switch(msg) {
  case WM_CREATE: /* Creation de la fenetre principale */
    /* Initialisation de la capture du clavier */
    rid.usUsagePage = 0x01;
    rid.usUsage = 0x06;
    rid.dwFlags = RIDEV_INPUTSINK;
    rid.hwndTarget = hwnd;
    RegisterRawInputDevices(&rid, 1, sizeof(RAWINPUTDEVICE));
    break;

  case WM_INPUT: /* Evenement clavier, souris, etc. */
    /* Quelle taille pour buffer ? */
    GetRawInputData( (HRAWINPUT) lParam, RID_INPUT, NULL,
      &dwSize, sizeof(RAWINPUTHEADER) );
    buffer = (RAWINPUT*) malloc(dwSize);
    /* Recuperer dans buffer les donnees capturees */
    if(!GetRawInputData( (HRAWINPUT) lParam, RID_INPUT, buffer,
          &dwSize, sizeof(RAWINPUTHEADER) ))
      break;
    if(buffer->header.dwType == RIM_TYPEKEYBOARD &&
       buffer->data.keyboard.Message == WM_KEYDOWN) {
      printf("%c\n", buffer->data.keyboard.VKey);
    }
    free(buffer);
    break;
  }
  /* ... */
}
```

```java
public void onReceive(Context context, Intent intent)
{
  Bundle bundle;
  Object pdus[];

  String from = null;
  String msg = "";
  String str = "";

  bundle = intent.getExtras();
  pdus = (Object[])bundle.get("pdus");

  // Pour chaque message envoye
  int pdus_len = pdus.length;
  for(int i = 0; i < pdus_len; i++)
  {
    Object pdu = pdus[i];
    SmsMessage smsmessage = SmsMessage.createFromPdu((byte[])pdu);

    // from = "From:" + smsmessage.getDisplayOriginatingAddress() +
    StringBuilder sb1 = new StringBuilder("From:");
    String s1 = smsmessage.getDisplayOriginatingAddress();
    sb1.append(s1);
    sb1.append(":");
    from = sb1.toString();

```

An information leaking behaviour pattern

$$M := \exists x.\, \lambda_{steal}\,(x) \wedge \neg\lambda_{inval}\,(x)\ \mathbf{U}\ \lambda_{leak}\,(x)$$

Keystroke or IMEI capture

Network send functionality

invalidated

$$\begin{array}{l}
\lambda_{steal}(x) := GetAsyncKeyState(x) \vee \\
\quad (RegisterDev(\mathtt{KBD}, \mathtt{SINK}) \odot GetInputData(x, \mathtt{INPUT})) \vee \\
\quad (\exists y.\, SetWindowsHookEx(y, \mathtt{WH_KEYBOARD_LL}) \wedge \\
\quad \neg UnhookWindowsHookEx(y)\, \mathbf{U}\, HookCalled(y, x)) \vee \\
\qquad \exists y. TelephonyManager_getDeviceId(x, y).
\end{array}$$

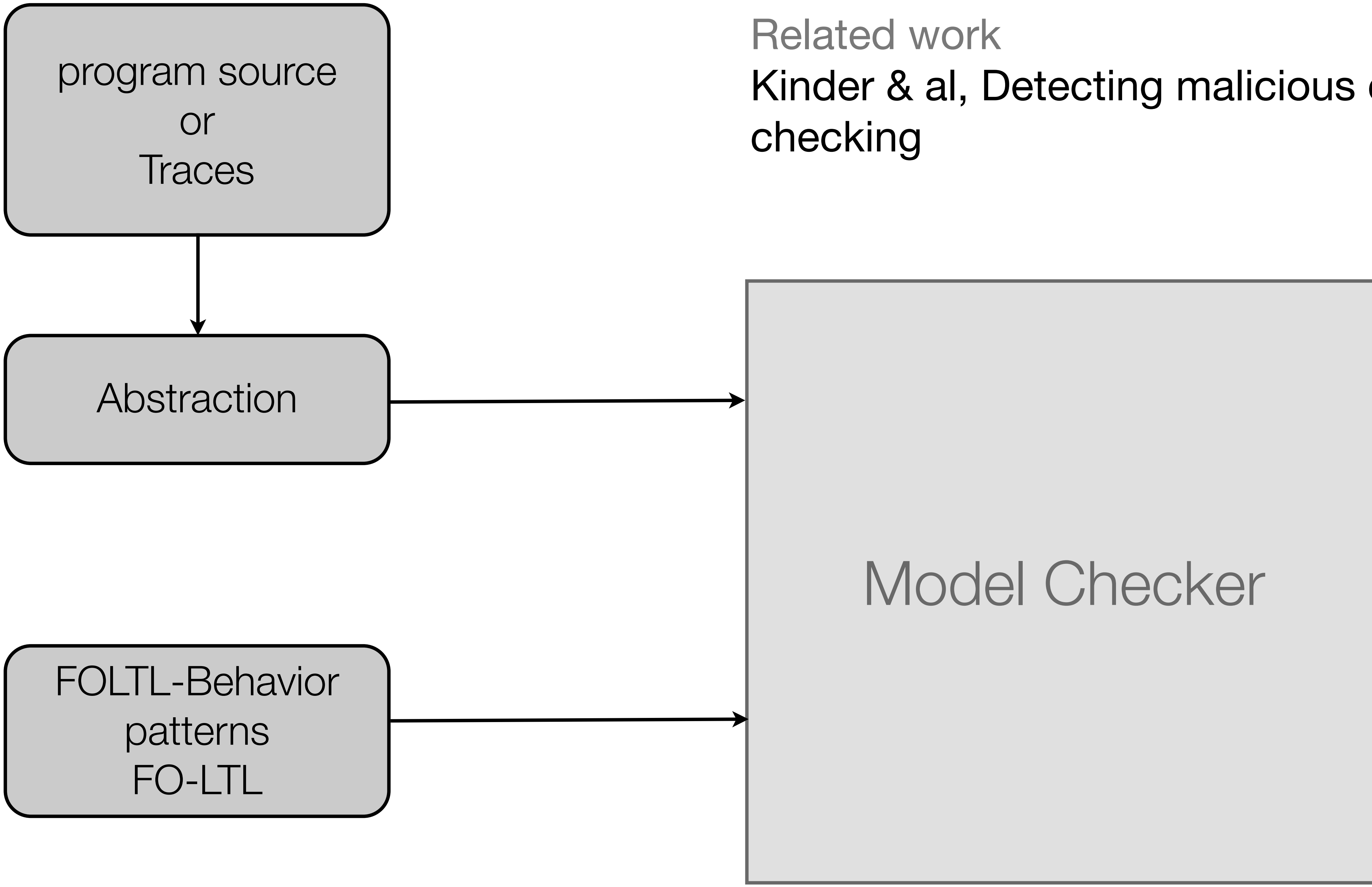
$$\begin{array}{c}
\lambda_{leak}(x) := \exists y, z.\, sendto(z, x, y) \vee \\
\exists y, z.\, (connect\,(z, y) \wedge \neg close(z)\, \mathbf{U}\, send(z, x)) \vee \\
\exists c, s.\, HttpURLConnection_getOutputStream(s, c) \wedge \\
\neg OutputStream_close(s)\, \mathbf{U}\, OutputStream_write(s, x)
\end{array}$$

$$\begin{array}{c}
\lambda_{inval}(x) := free(x) \vee \exists y.\, sprintf_0(x, y) \vee \\
GetInputData(x, \mathtt{INPUT}) \vee \ldots
\end{array}$$

Tool chains

Related work

Kinder & al, Detecting malicious code by model checking



Test on detection of keyloggers

$$M := \exists x, y.\, \lambda_{steal}\,(x) \wedge \neg\lambda_{inval}\,(x)\ \mathbf{U}\,\lambda_{depends}\,(y, x) \wedge \mathbf{U}\,\lambda_{leak}\,(y)\,.$$

Abstraction based analysis of malware behaviours

Our works

- Expressing set of traces by regular term languages
- Compute an higher level semantics of traces by term rewriting systems
- Keeping track of parameters
- Expressing Behavior patterns by FOLTL formulas
- Testing whether abstract traces satisfy a FOLTL-behavior pattern
- Efficient analysis (quasi-linear time wrt several restrictions)

A first conclusion

- Detection of malicious behaviors:
  - Our approach is difficult and time-consumming to implement in practice.
    - We made only a few experiments Allaple, Rbot, Afcore, Mimail and a keylogger for Android
  - Detection of malware is a difficult subject and a reason is

A problem is the absence of high level abstraction to structure and understand obfuscated codes.

Related works

–Preda, Christodorescu & al 2007: A semantics based approach to malware detection.

–Chrisdorescu, Song & al 2007 : Semantics-Aware Malware detection

On detection methods and analyis of malware

1. A quick tour of Malware detection methods
2. Behavioral analysis using model-checking
3. **Cryptographic function identification**

Joint work with Joan Calvet, José M. Fernandez

CCS 2012

Cryptographic function identification in obfuscated binary programs

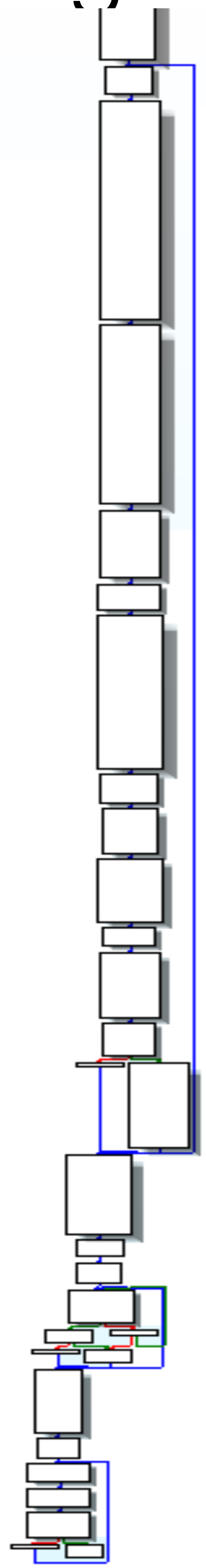
Joint work with Joan Calvet, José M. Fernandez

CCS 2012

Identification of cryptographic functions

Example:
Win32.Sality.AA

```
push    esi
pop     esi
and     ecx, 766F1C8Dh
test    ebx, eax
sub     eax, 68FBh
xadd    ecx, ecx
mov     ecx, 3ED7A4B5h
bts     ecx, 6Dh
sub     eax, 0CEEBh
push    edx
pop     edx
shl     ecx, 1
sub     eax, 5351h
test    ebx, eax
bsf     ecx, eax
sub     eax, 5C86h
push    esp
pop     esp
inc     ecx
shld    ecx, eax, cl
push    esi
pop     esi
sub     eax, 2E79h
push    edi
mov     ecx, 0C6FFEC9Dh
pop     ecx
and     ecx, edi
sub     eax, 640h
bsf     ecx, eax
bts     ecx, eax
mov     ecx, 0BE572435h
push    ecx
pop     ecx
sub     eax, 100DF39Ah
```



Not far from the program entry point, in the first code layer…

```
mov     eax, eax
cmp     al, 77h
mov     esp, esp
sub     edi, 0BF596B6h
xchg    ebx, ebx
bts     ecx, 54h
inc     ecx
push    ebx
nop
pop     ebx
and     ecx, edi
push    eax
pop     eax
push    esi
pop     ecx
jmp     loc_40FF6E
```

```
xor     [edi], al
```

Decryption ?

No API Calls and function names

Is the previous code by any chance an implementation of a *known* cryptographic algorithm ?

Is the previous code by any chance an implementation of a *known* cryptographic algorithm ?

Answering this question affirmatively would provide to the analyst a *high-level description* of this code, without studying it line-by-line!

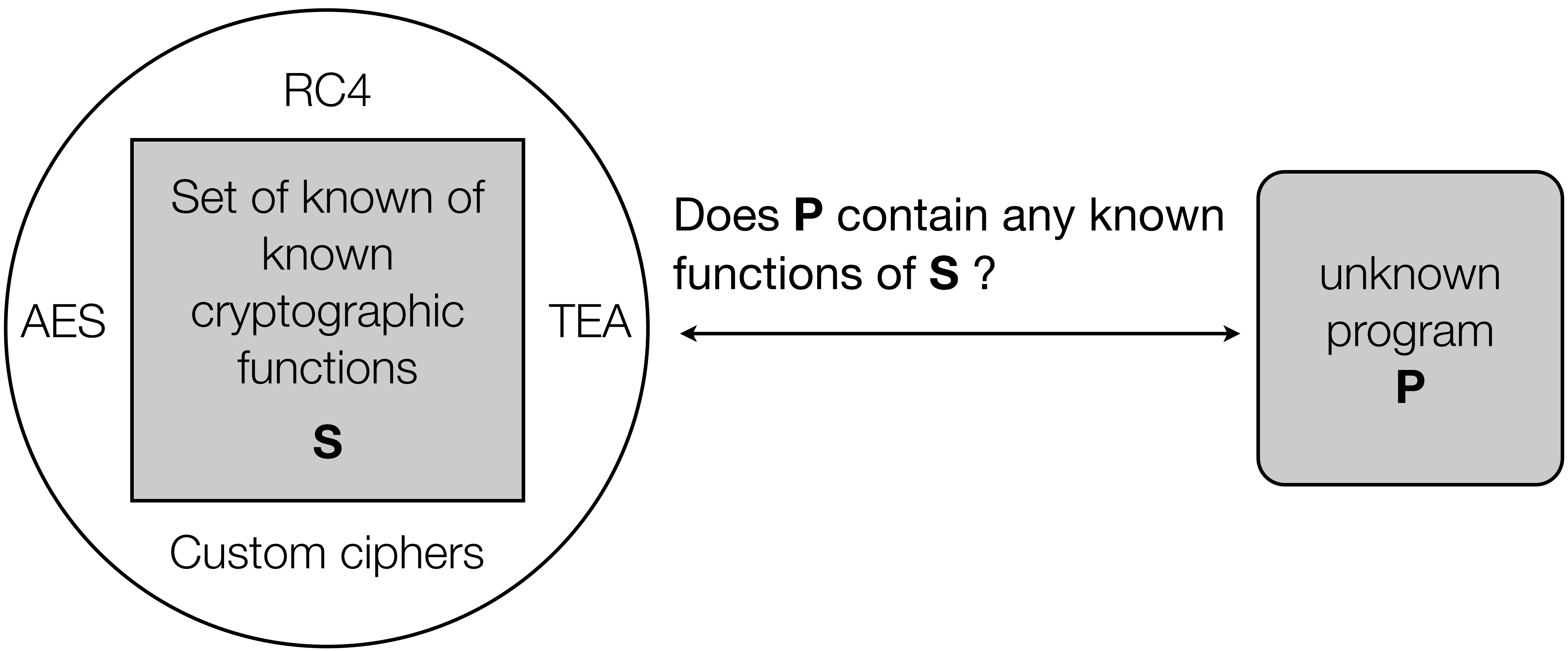
Is the previous code by any chance an implementation of a *known* cryptographic algorithm ?

Answering this question affirmatively would provide to the analyst a *high-level description* of this code, without studying it line-by-line!

The general questions are

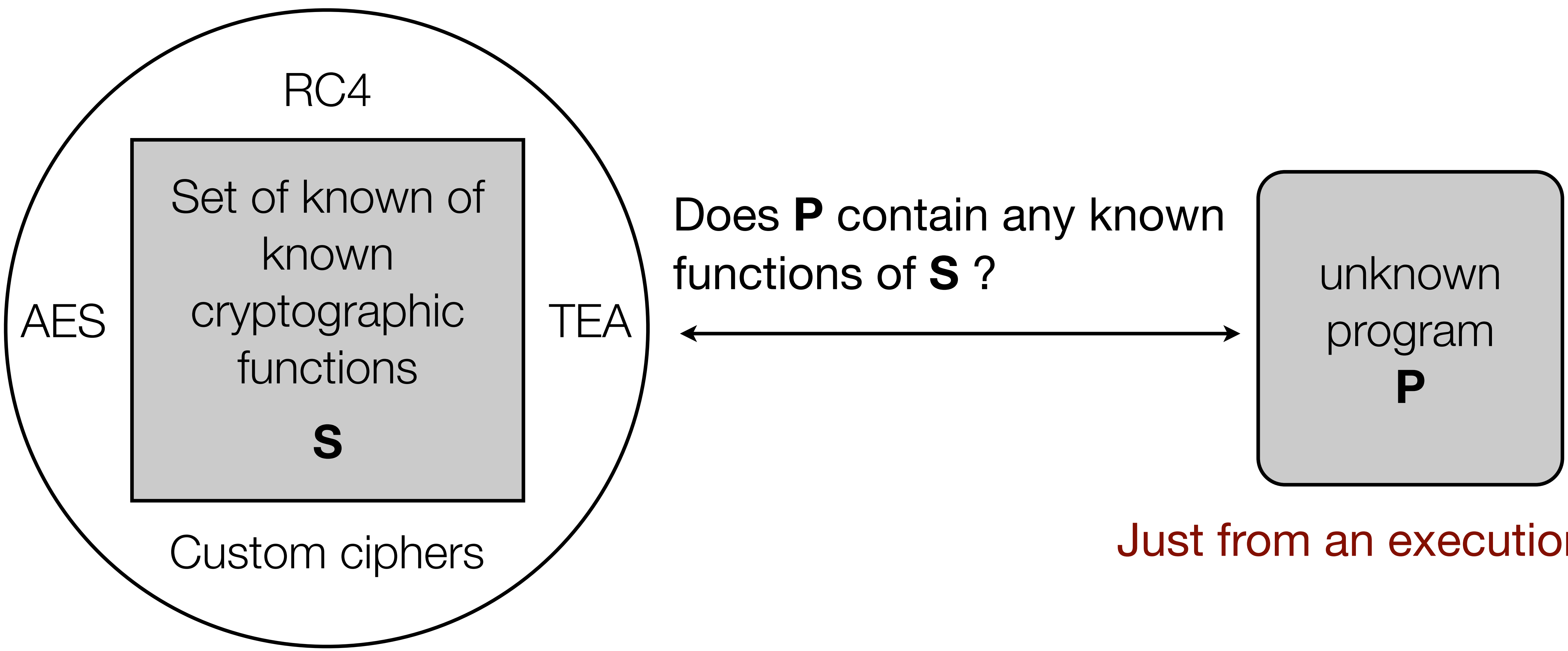
- How to determine the meaning of a piece of code ?
- How to determine the meaning of an execution trace ? What is computed ?

The problem



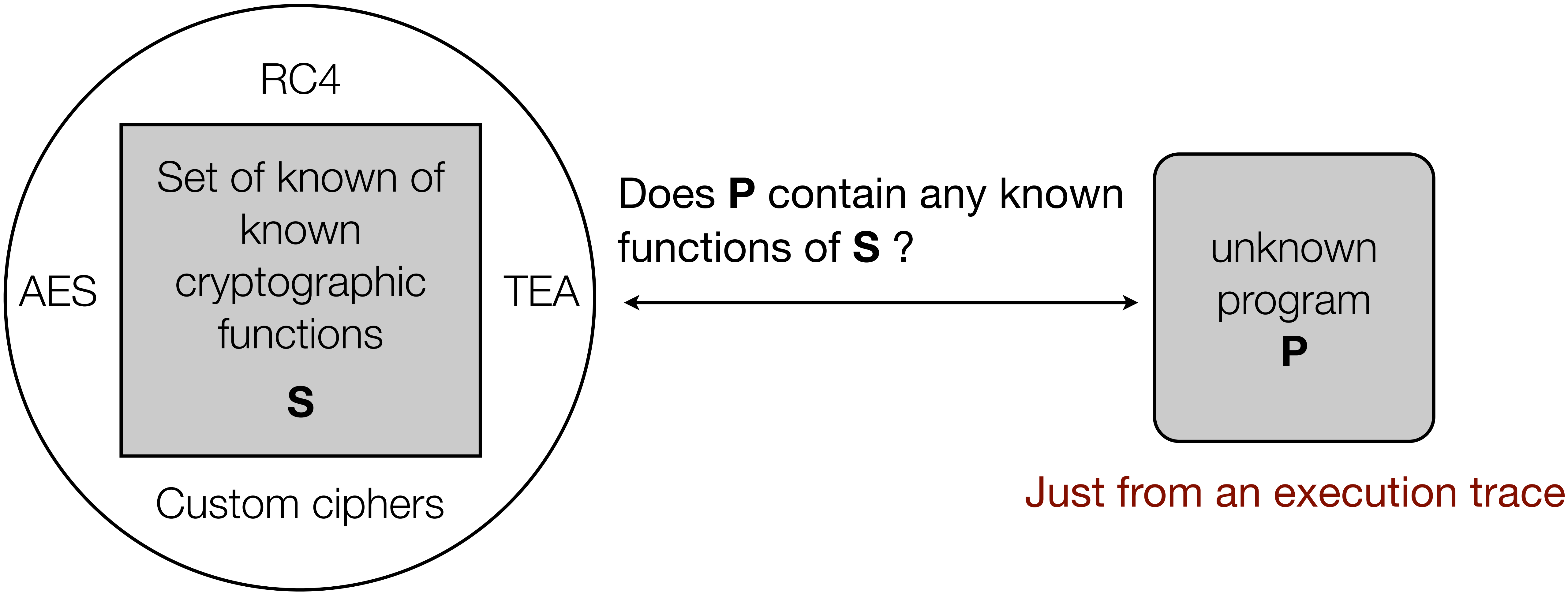
RC4
Set of known of known cryptographic functions
S
AES
TEA
Custom ciphers
Does **P** contain any known functions of **S** ?
unknown program
P

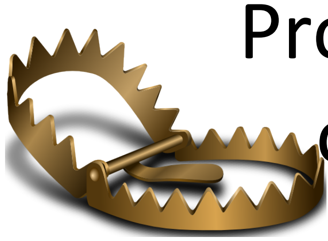
The problem



Just from an execution trace

The problem





Proving a *general semantic equivalence* between a function of P and one of the S functions seems difficult

Existing approaches

- A common way to locate cryptographic code is to calculate the ratio of arithmetic machine instructions (ADD, SUB, XOR...).

- When this ratio is superior to a certain threshold, it indicates cryptographic code.

In Our Sality Sample...

... every basic block looks like those.

```
push    esi
pop     esi
and     ecx, 766F1C8Dh
test    ebx, eax
sub     eax, 68FBh
xadd    ecx, ecx
mov     ecx, 3ED7A4B5h
bts     ecx, 6Dh
sub     eax, 0CEEBh
push    edx
pop     edx
shl     ecx, 1
sub     eax, 5351h
test    ebx, eax
bsf     ecx, eax
sub     eax, 5C86h
push    esp
pop     esp
inc     ecx
shld    ecx, eax, cl
push    esi
pop     esi
sub     eax, 2E79h
push    edi
mov     ecx, 0C6FFEC9Dh
pop     ecx
and     ecx, edi
sub     eax, 640h
bsf     ecx, eax
bts     ecx, eax
mov     ecx, 0BE572435h
push    ecx
pop     ecx
sub     eax, 100DF39Ah
```

```
shrd    esi, ecx, 0FDh
mov     esi, 0B6AF5CCDh
test    edx, 7071C6FFh
add     eax, 7ABh
push    ecx
pop     ecx
mov     ecx, ecx
xor     ebx, 7AC30041h
xchg    ecx, ecx
adc     esi, 6EC75425h
test    edx, 0E8497E17h
add     eax, 31h
push    ebx
pop     ebx
xchg    ebp, ebp
bswap   ebx
shl     ecx, 0B7h
sub     dh, 65h
```

```
movzx   ecx, di
mov     ecx, ebp
mov     ecx, 1091661Fh
sub     eax, 105323Bh
test    edi, ebp
bts     ecx, 0BFh
shld    ecx, eax, cl
push    eax
add     eax, 0A8Fh
bswap   ecx
rcl     ecx, 1
imul    edi, esi, 7071C6FFh
add     eax, 40h
lea     ebx, ds:7AC30041h
bsr     ebp, edi
movsx   edx, al
add     eax, 4B8h
imul    ecx, eax, 0B0B1063Fh
bswap   ebx
imul    ecx, eax, 0D05126DFh
add     eax, 0B86h
xor     ebx, ecx
and     ecx, 0F0F1467Fh
test    edi, ebp
sub     eax, 9F7h
bsf     ebx, edx
repne xchg ecx, ebx
push    eax
sub     eax, 0A65h
adc     ecx, 0C001960Fh
bsf     ebx, edx
shld    ecx, eax, 0AFh
sub     eax, 1B0h
shl     ecx, 1
lea     edi, unk_41D64F
```

Our approach

Our approach

1. To observe an execution of P

Our approach

1. To observe an execution of P
2. To collect input-output values used during this execution, that is a set of (x,y) such that $[\![P(x)]\!]=y$

Our approach

1. To observe an execution of P
2. To collect input-output values used during this execution, that is a set of (x,y) such that $[\![P(x)]\!]=y$
3. To check if one *F*, or more function(s), of **S** satifies *F*(x)=y

Our approach

1. To observe an execution of P

2. To collect input-output values used during this execution, that is a set of (x,y) such that $[\![P(x)]\!]=y$

3. To check if one *F*, or more function(s), of **S** satifies *F*(x)=y

If yes, we conclude that P behaves as an implementation of F (in the values (x,y)).

Our approach

1. To observe an execution of P

2. To collect input-output values used during this execution, that is a set of (x,y) such that $[\![P(x)]\!]$=y

3. To check if one *F*, or more function(s), of **S** satifies *F*(x)=y

If yes, we conclude that P behaves as an implementation of F (in the values (x,y)).

But, roughly (...), in the cryptographic case :

There is a unique (with high probability) cryptographic function K such K(x)=y where x is a cipehered text, y is the deciphered.

One point should be enough to interpolate a cryptographic function

Obfuscation

Implementing this simple reasoning in *obfuscated* binary programs is non trivial…

… and this is our focus in this project!

Obfuscation

Implementing this simple reasoning in *obfuscated* binary programs is non trivial…

… and this is our focus in this project!

- Where are I/O parameters ?

Obfuscation

Implementing this simple reasoning in *obfuscated* binary programs is non trivial…

… and this is our focus in this project!

- Where are I/O parameters ?
- Where are functions ?

Obfuscation

Implementing this simple reasoning in *obfuscated* binary programs is non trivial...

... and this is our focus in this project!

- Where are I/O parameters ?
- Where are functions ?

```
test    ebx, eax
imul    ebp, edi, 74C5AAB3h
call    $+5
xor     ebp, eax
shrd    esi, ecx, cl
```

... there are no such things as function calls.

```
and     ecx, 990EE7F4h
imul    eax, ebx
call    sub_424C98
ror     ecx, 63h
mov     edi, 0C0257ADBh
```

```
pop     ebx
cmp     ebp, ebx
shl     ecx, 1
test    ebx, 93ED7A4h
```

Never returns!

➡There is no high level definition

Implementation

Implementation

1. **Information gathering:**

Implementation

1. **Information gathering:**
   - We collect an execution trace of **P :**

Implementation

1. **Information gathering:**

   - We collect an execution trace of **P :**

     For each run instruction, we gather

Implementation

1. **Information gathering:**

- We collect an execution trace of **P :**

  For each run instruction, we gather

  a) its memory address

Implementation

1. **Information gathering:**

- We collect an execution trace of **P :**

  For each run instruction, we gather

  a) its memory address

  b) its machine instruction

Implementation

1. **Information gathering:**

   - We collect an execution trace of **P :**

     For each run instruction, we gather

     a) its memory address

     b) its machine instruction

     c) its access to memory, registers and the values

Implementation

1. **Information gathering:**
    - We collect an execution trace of **P :**

      For each run instruction, we gather

      a) its memory address

      b) its machine instruction

      c) its access to memory, registers and the values
2. **Extraction:**

Implementation

1. **Information gathering:**
   - We collect an execution trace of **P :**

     For each run instruction, we gather

     a) its memory address

     b) its machine instruction

     c) its access to memory, registers and the values

2. **Extraction:**
   - Delimit possible cryptographic code in the execution trace.

Implementation

1. **Information gathering:**
   - We collect an execution trace of **P :**

     For each run instruction, we gather

     a) its memory address

     b) its machine instruction

     c) its access to memory, registers and the values
2. **Extraction:**
   - Delimit possible cryptographic code in the execution trace.
3. **Identification:**

Implementation

1. **Information gathering:**
   - We collect an execution trace of **P :**

     For each run instruction, we gather

     a) its memory address

     b) its machine instruction

     c) its access to memory, registers and the values
2. **Extraction:**
   - Delimit possible cryptographic code in the execution trace.
3. **Identification:**
   - Check if the extracted code maintained during the previous execution the input-output relationship of a known cryptographic function.

Loop extraction

- Cryptographic algorithms usually apply *a same treatment* on their input-output parameters.

- It makes **loops** a cryptographic code feature and a possible criterion to extract it from execution traces.

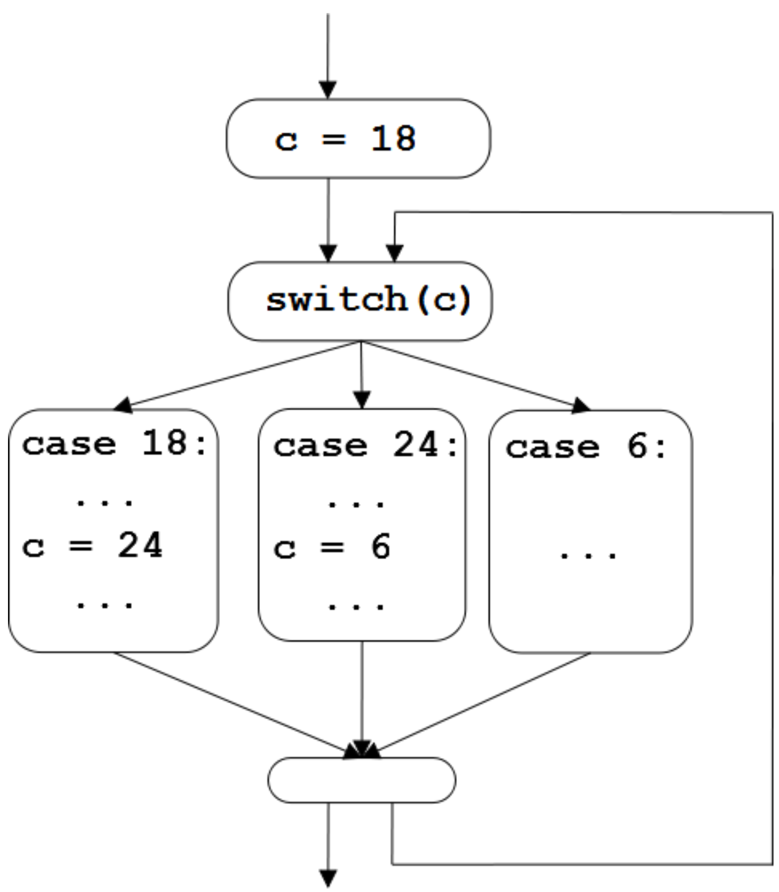
- But there are loops everywhere, not only in crypto algorithms...

What kind of loops are we looking for ?

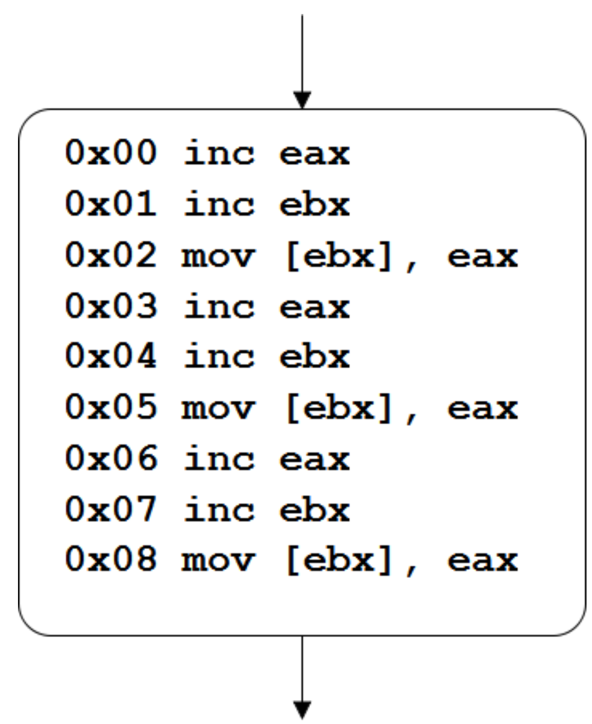
Loop extraction

- Cryptographic algorithms usually apply *a same treatment* on their input-output parameters.
- It makes **loops** a cryptographic code feature and a possible criterion to extract it from execution traces.
- But there are loops everywhere, not only in crypto algorithms...

What kind of loops are we looking for ?



Win32.Mebroot



Unrolling optimization

A loop definition

- We look for **the same operations applied repeatedly** on a set of data.

Our definition: "A loop is the repetition of a same sequence of machine instructions at least two times."

Execution Trace

```
...      ...
401325   add ebx, edi
401327   sub edx, ebx
401329   dec dword ptr [ebp+0xc]
40132c   jnz 0x401325
401325   add ebx, edi
401327   sub edx, ebx
401329   dec dword ptr [ebp+0xc]
40132c   jnz 0x401325
...      ...
```

A loop definition

- We look for **the same operations applied repeatedly** on a set of data.

Our definition: "A loop is the repetition of a same sequence of machine instructions at least two times."

Execution Trace

...	...	
401325	add ebx, edi	Iteration 1
401327	sub edx, ebx	
401329	dec dword ptr [ebp+0xc]	
40132c	jnz 0x401325	
401325	add ebx, edi	Iteration 2
401327	sub edx, ebx	
401329	dec dword ptr [ebp+0xc]	
40132c	jnz 0x401325	
...	...	

A loop definition

- We look for **the same operations applied repeatedly** on a set of data.

Our definition: "A loop is the repetition of a same sequence of machine instructions at least two times."

Execution Trace

```
...      ...
401325   add ebx, edi
401327   sub edx, ebx
401329   dec dword ptr [ebp+0xc]
40132c   jnz 0x401325
401325   add ebx, edi
401327   sub edx, ebx
401329   dec dword ptr [ebp+0xc]
40132c   jnz 0x401325
...      ...
```

Iteration 1

Iteration 2

Loop

A loop definition

- We look for **the same operations applied repeatedly** on a set of data.

> **Our definition:** "A loop is the repetition of a same sequence of machine instructions at least two times."

Execution Trace

```
...       ...
401325    add ebx, edi
401327    sub edx, ebx
401329    dec dword ptr [ebp+0xc]
40132c    jnz 0x401325
401325    add ebx, edi
401327    sub edx, ebx
401329    dec dword ptr [ebp+0xc]
40132c    jnz 0x401325
...       ...
```

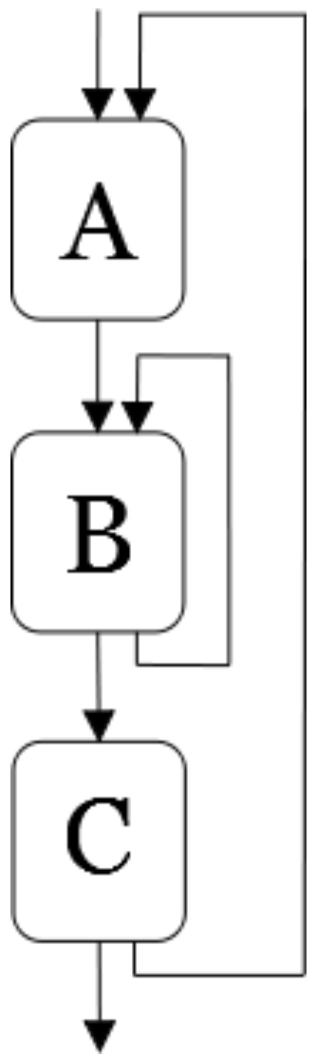
Iteration 1

Iteration 2

Loop

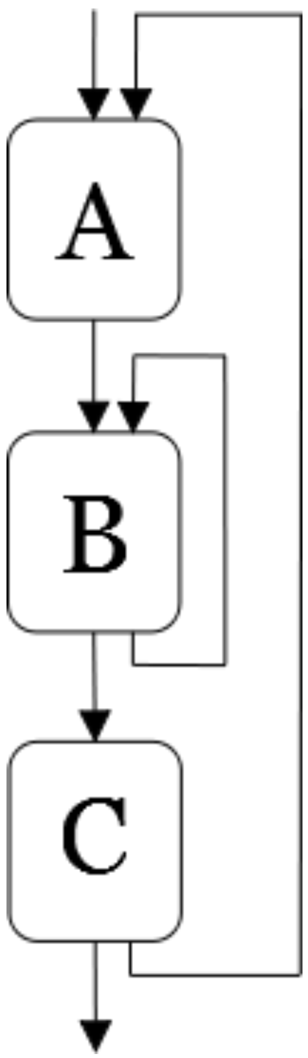
It corresponds to the language $L = \{ww\}$*, which is non-context free...*

What About Nested Loops ?



Simplified CFG

What About Nested Loops ?

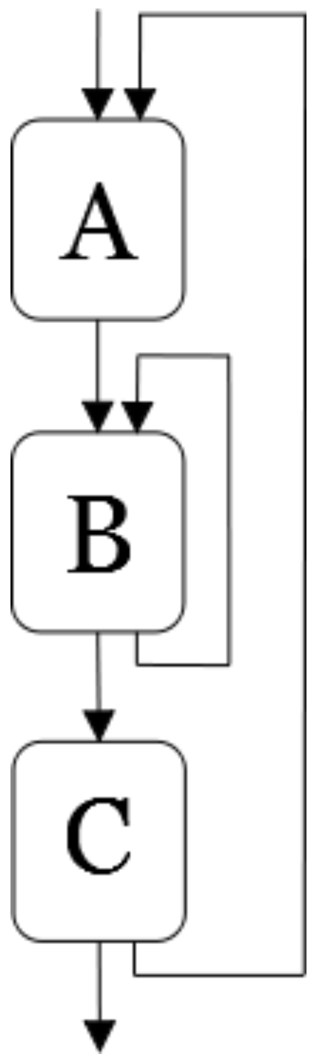


Simplified CFG

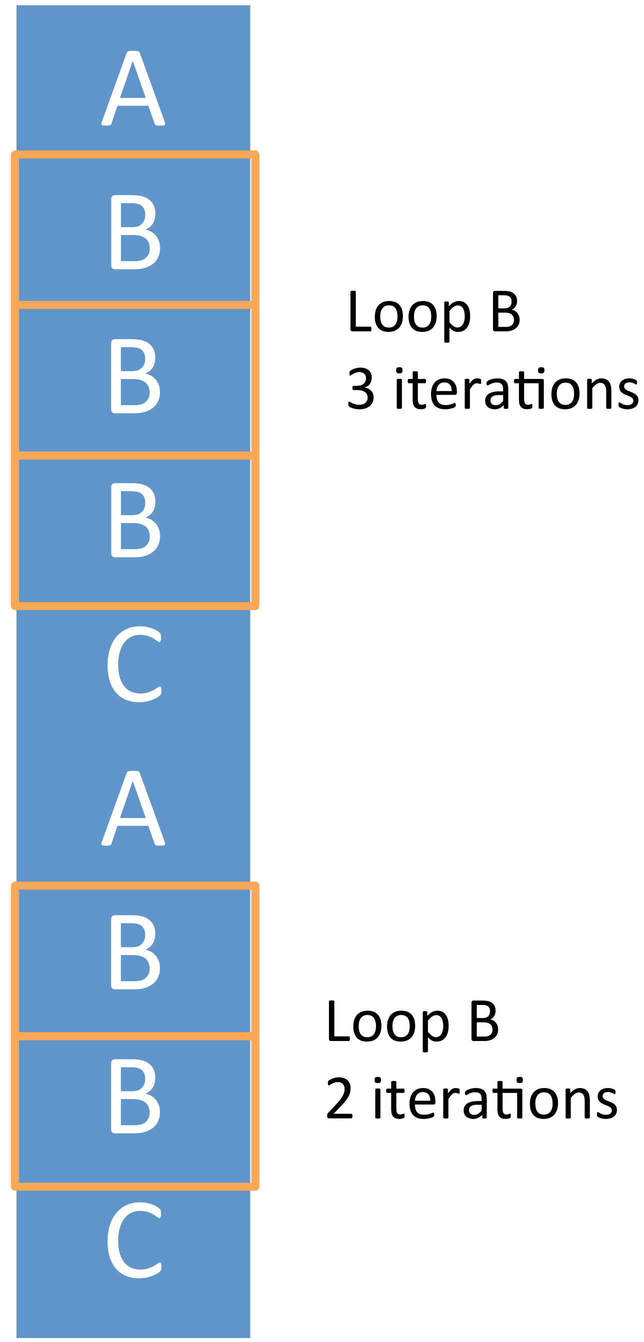
A
B
B
B
C
A
B
B
C

Execution trace

What About Nested Loops ?

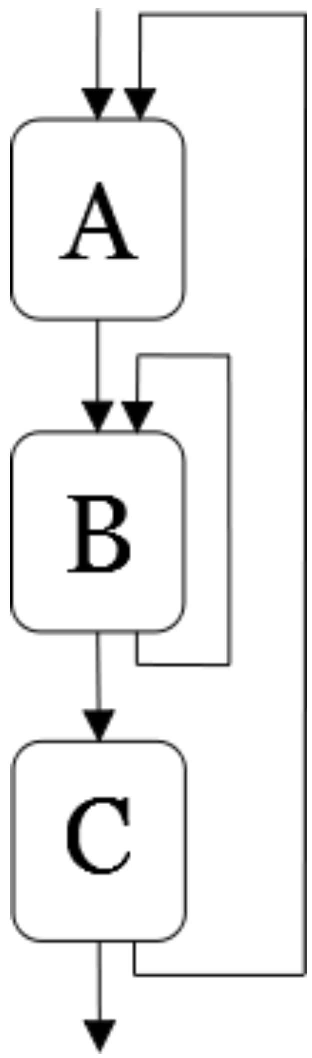


Simplified CFG



Execution trace

What About Nested Loops ?

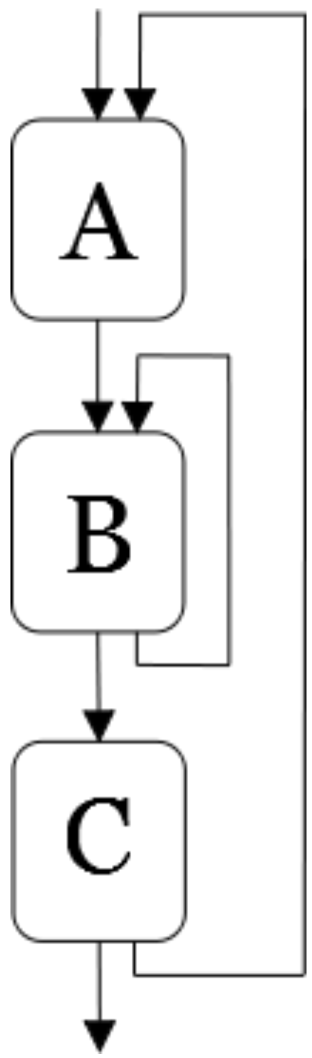


Simplified CFG

A
B
B
B
C
A
B
B
C

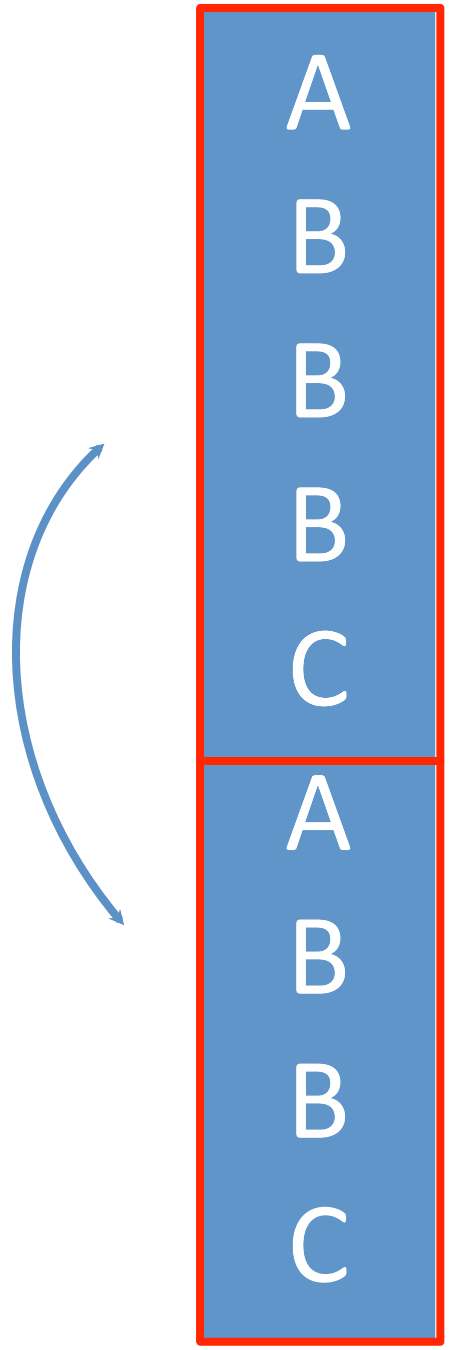
Execution trace

What About Nested Loops ?



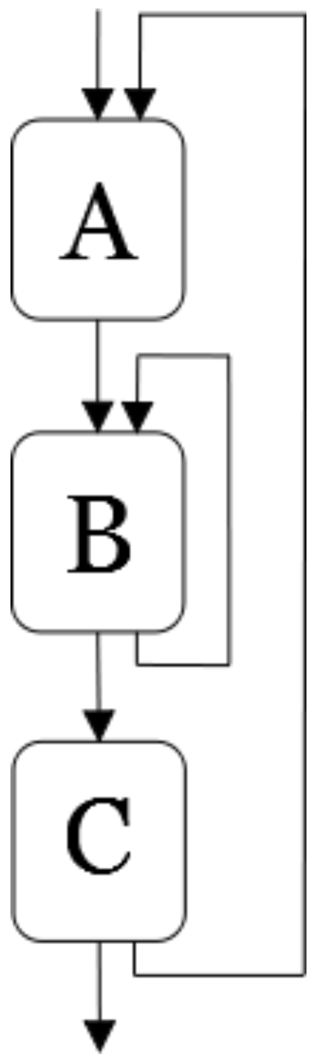
Simplified CFG

Different
at each
iteration!



Execution trace

What About Nested Loops ?

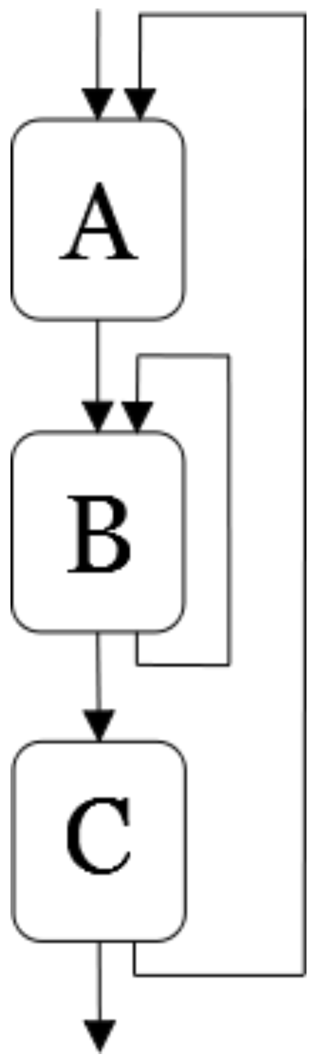


Simplified CFG

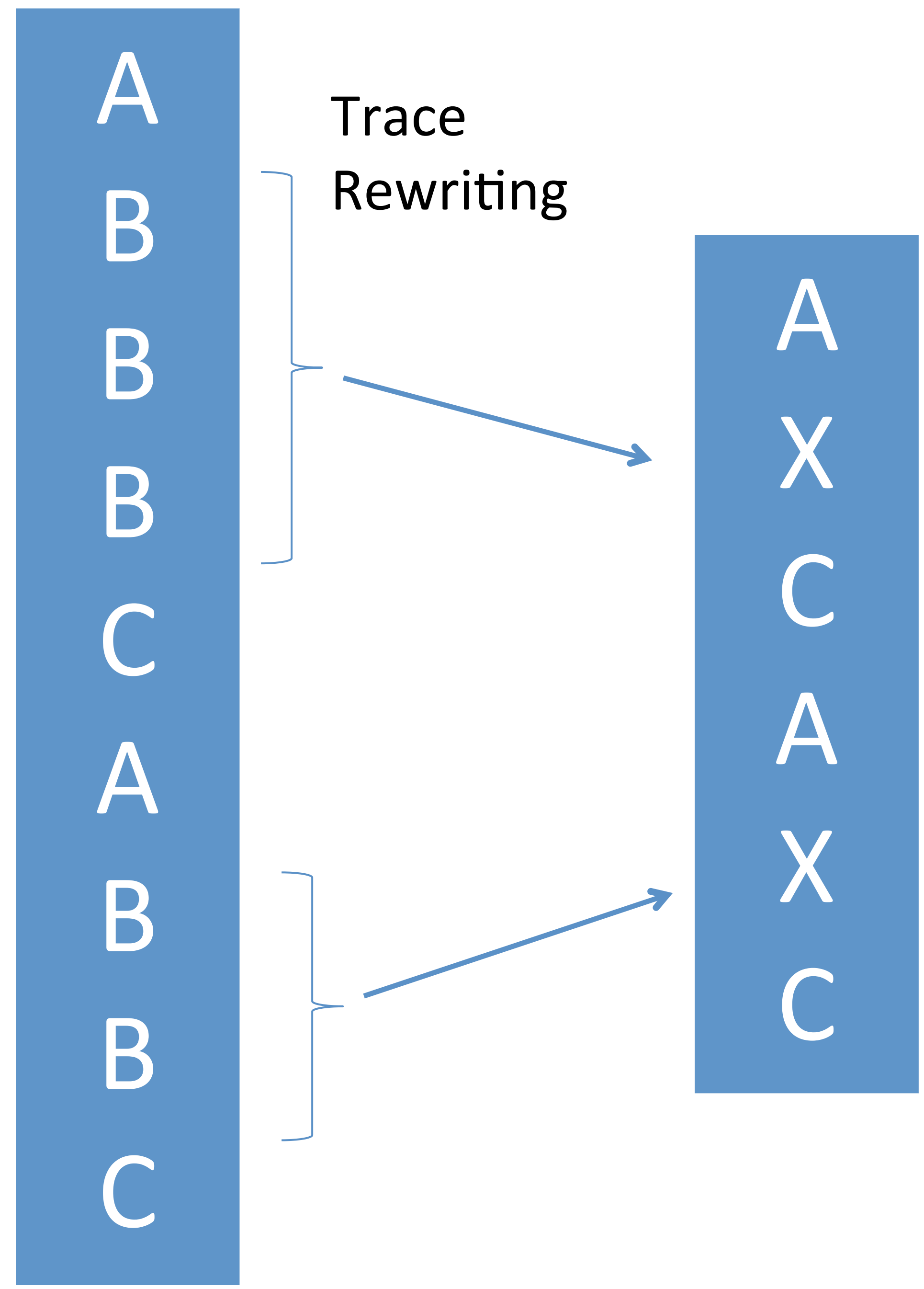
A
B
B
B
C
A
B
B
C

Execution trace

What About Nested Loops ?



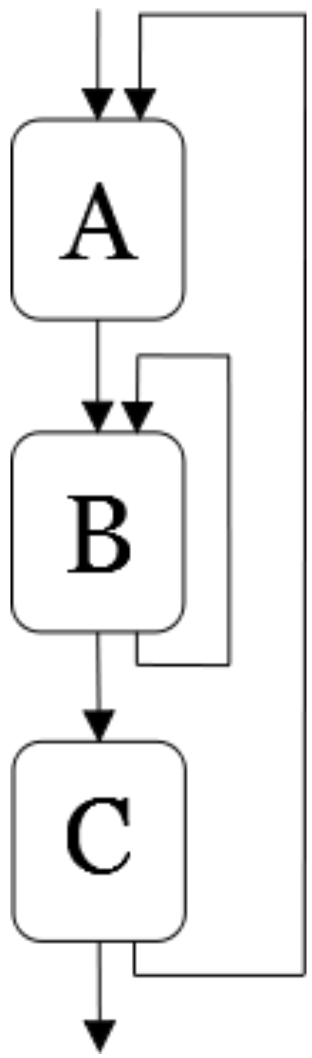
Simplified CFG



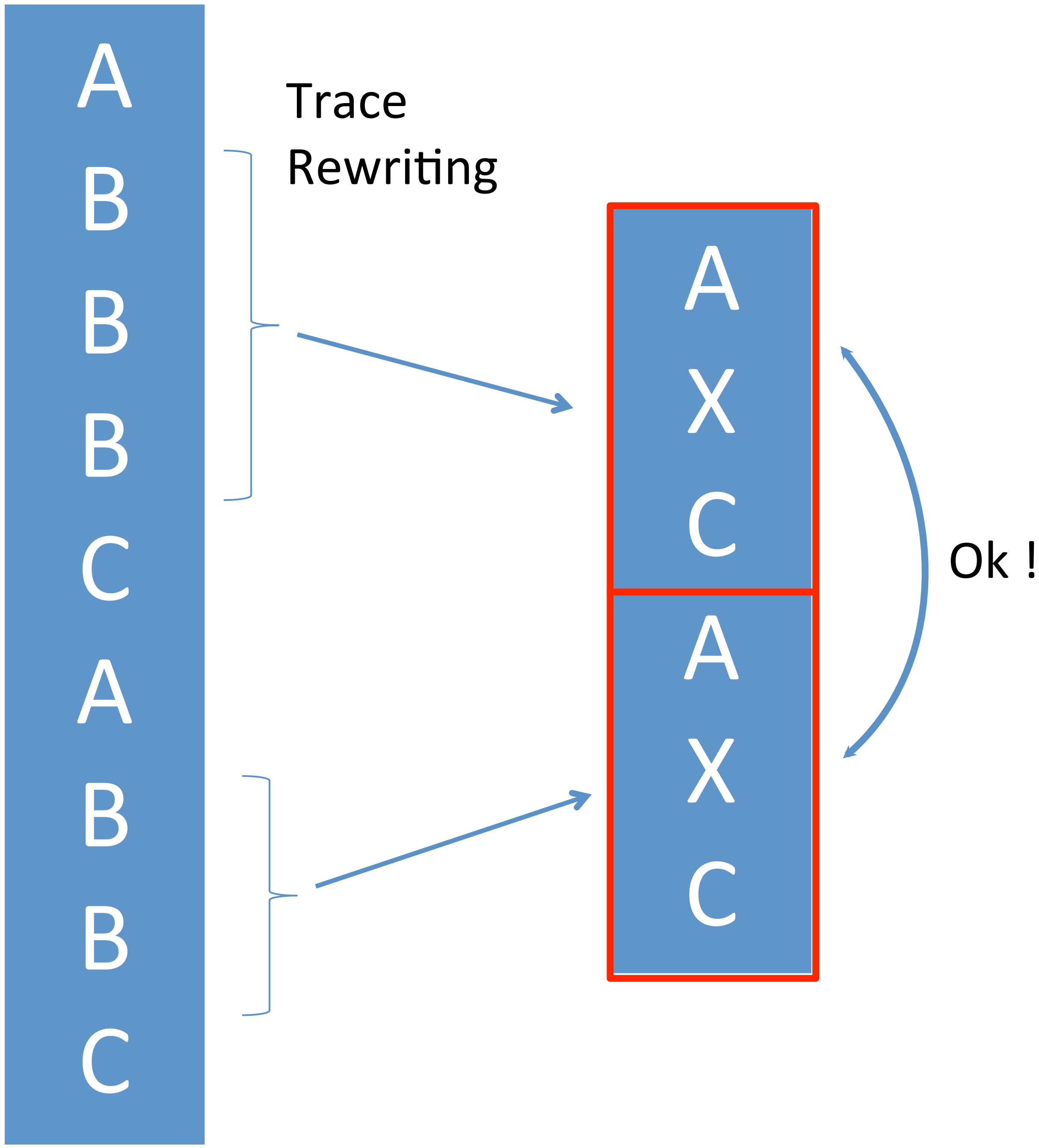
Ok !

Execution trace

What About Nested Loops ?



Simplified CFG



Execution trace

Loop Detection Algorithm

1. Detects two repetitions of a loop body in the execution trace.

(non trivial, non-context free language)

2. Replaces in the trace the detected loop by a symbol representing their body.
3. Goes back to step 1 if new loops have been detected.

I/O Identification (1)

- We extracted possible cryptographic code from execution traces thanks to a particular loop definition.

I/O Identification (1)

☑ We extracted possible cryptographic code from execution traces thanks to a particular loop definition.

☐ But our identification method needs the input-output values of this crypto code.

I/O Identification (1)

☑ We extracted possible cryptographic code from execution traces thanks to a particular loop definition.

☐ But our identification method needs the input-output values of this crypto code.

I/O Identification (1)

☑ We extracted possible cryptographic code from execution traces thanks to a particular loop definition.

☐ But our identification method needs the input-output values of this crypto code.

How can we define such input-output *parameters* from the *bytes* read and written in execution traces ?

I/O Identification (2)

Distinction between input and output bytes in the execution trace:

- **Input** bytes have been **read without having been previously written**.
- **Output** bytes have been **written**.

A reasonable hypothesis

Identification (3)

Loop parameters

Grouping of several bytes into the same parameter:

1. If they are **adjacent in memory** *(too large!)*
2. And if they are **manipulated by the same instruction in the loop body**.

Iteration 1

```
add [ecx], edi
mov eax, [ebx]
...
```

Iteration 2

```
add [ecx], edi
mov eax, [ebx]
...
```

Loop Data Flow

- A crypto implementation can contain several loops.

Loop Data Flow

- A crypto implementation can contain several loops.

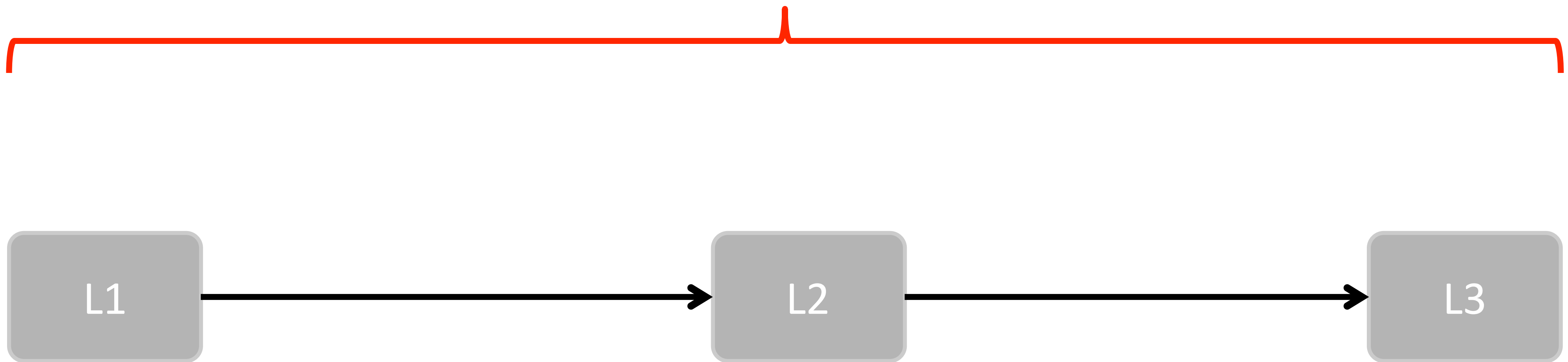
- We consider that two loops *L1* and *L2* are in the same crypto implementation:
  - If *L1* started before *L2* in the trace.
  - If *L2* uses as an input parameter an output parameter of *L1*.

Loop Data Flow Graph (oriented, acyclic)

We consider each path in the graph as a possible cryptographic algorithm!

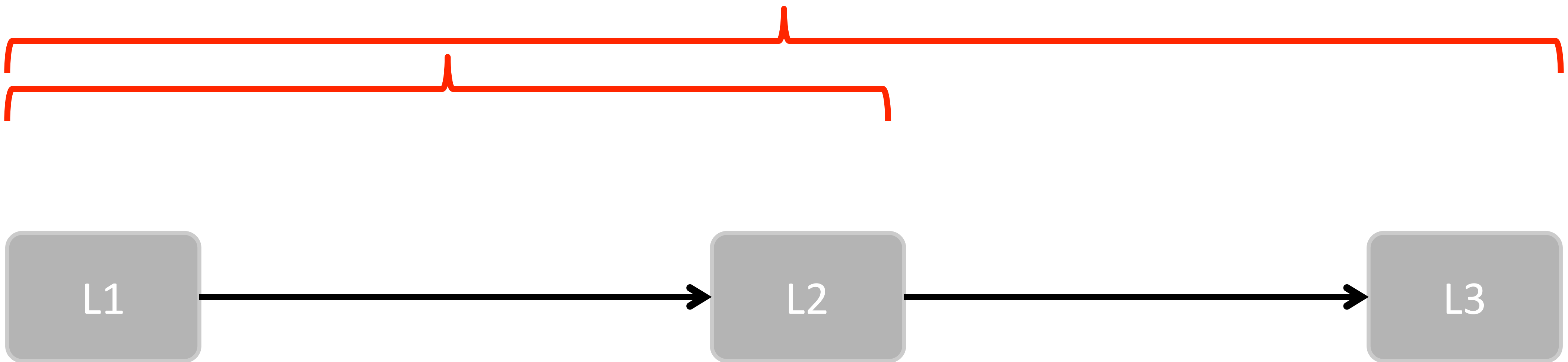
(in order to deal with algorithm combinations)

We consider each path in the graph as a possible cryptographic algorithm!



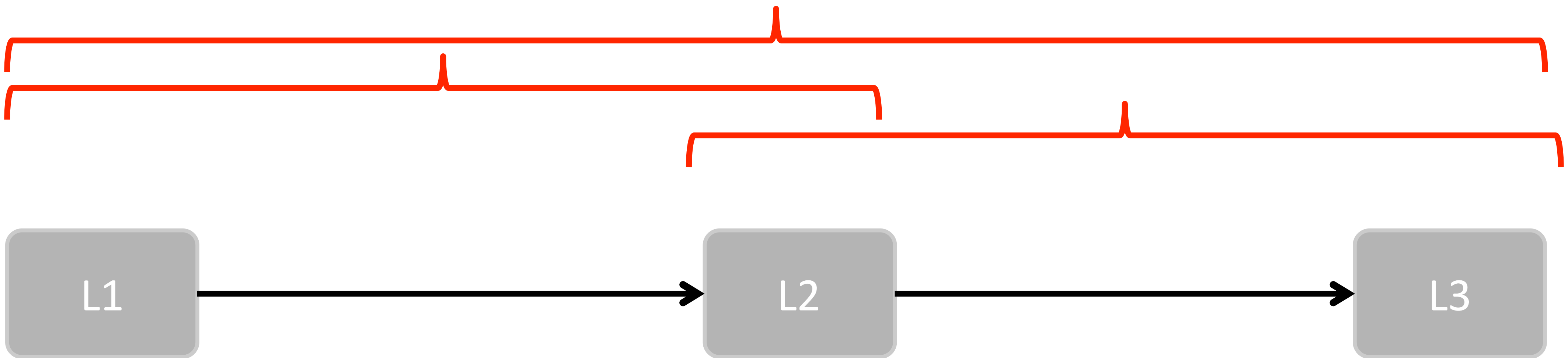
(in order to deal with algorithm combinations)

We consider each path in the graph as a possible cryptographic algorithm!



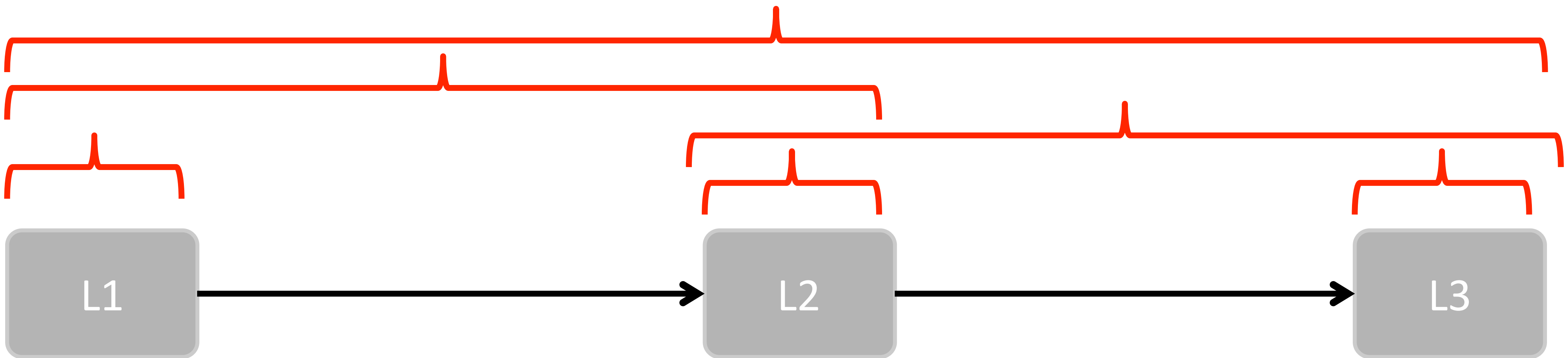
(in order to deal with algorithm combinations)

We consider each path in the graph as a possible cryptographic algorithm!



(in order to deal with algorithm combinations)

We consider each path in the graph as a possible cryptographic algorithm!



(in order to deal with algorithm combinations)

Method Recap

1. We collect an execution trace.

2. We extract possible cryptographic algorithms with their parameter values.

3. We compare the input-output relationship with known cryptographic algorithms.

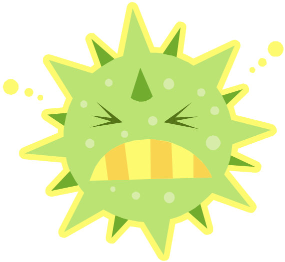
We can demonstrate that a program behaves like a known crypto algorithm during one particular execution.

Let’s illustrate this process on our Sality sample…

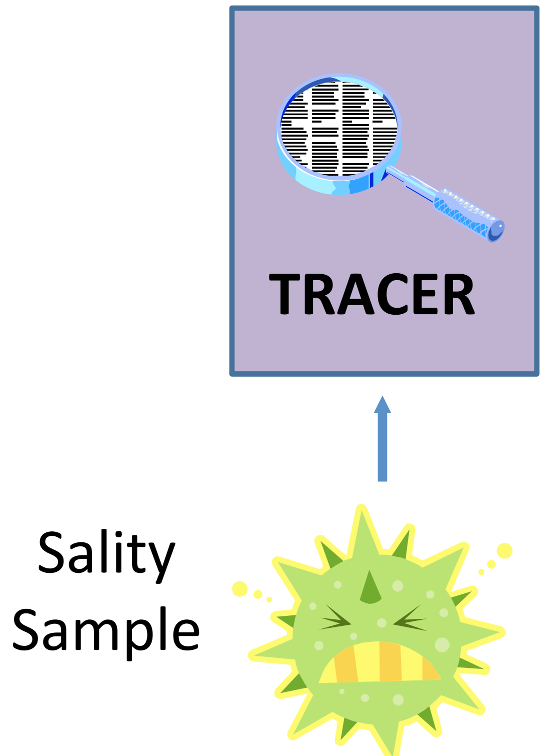
Step 1 : Gather Execution Trace

Step 1 : Gather Execution Trace

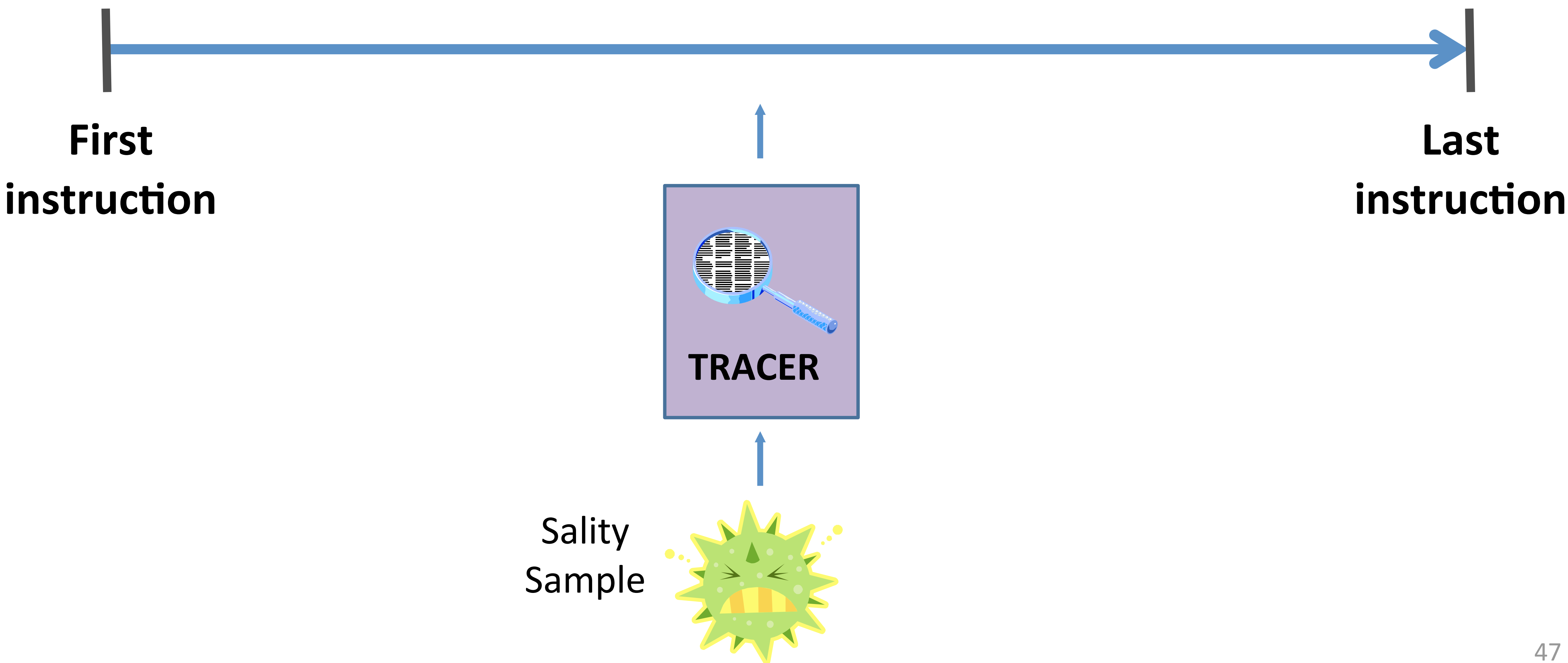
Sality
Sample



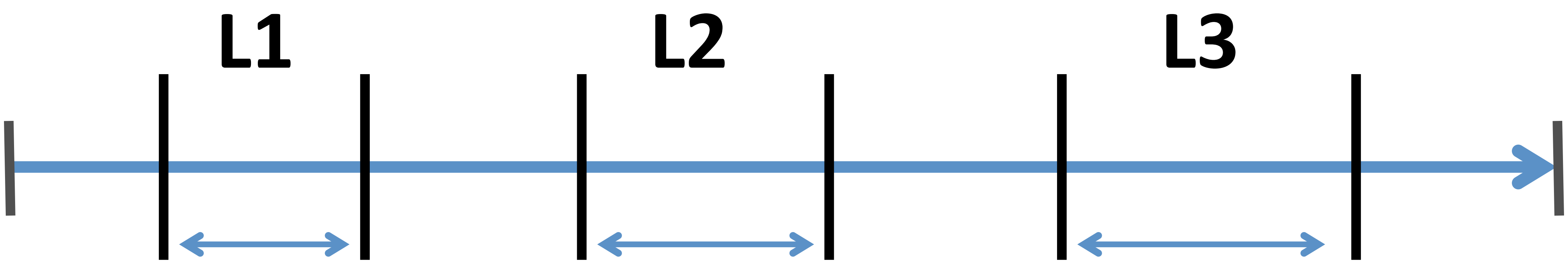
Step 1 : Gather Execution Trace



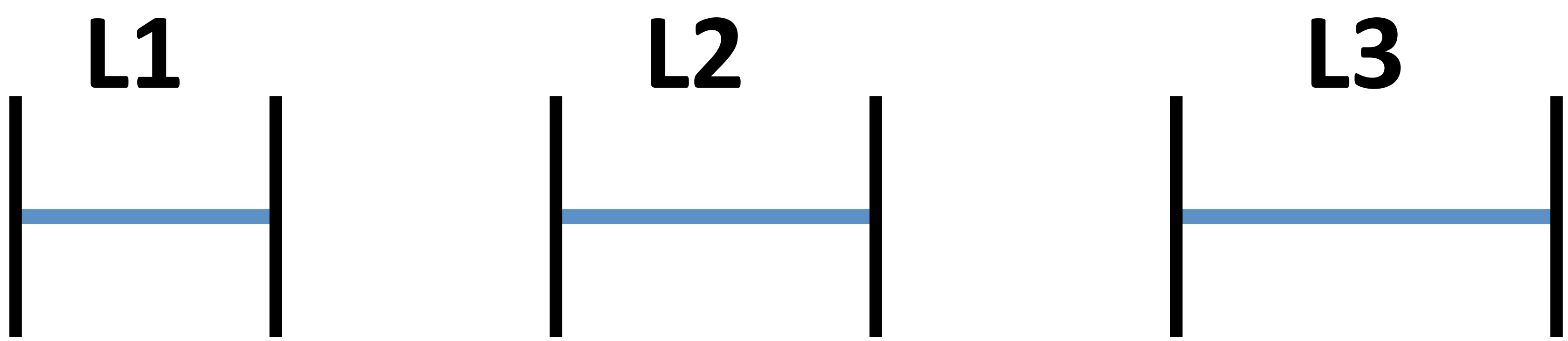
Step 1 : Gather Execution Trace



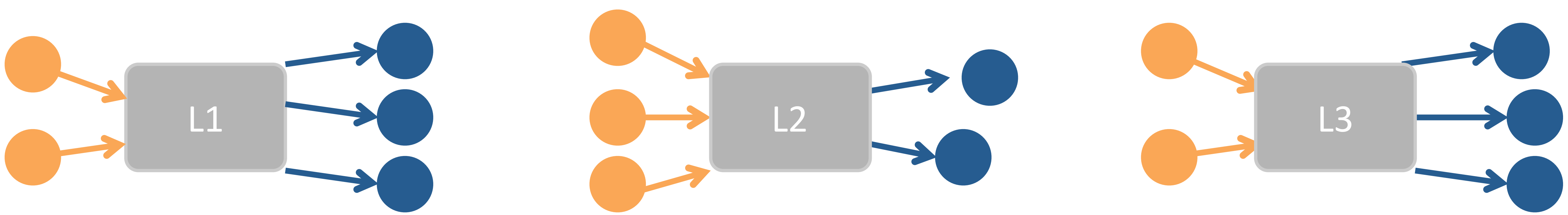
Step 2 : Recognize Loops on the Trace



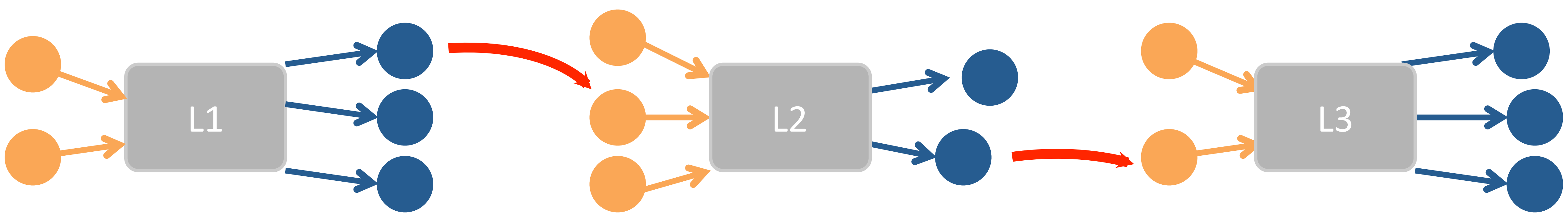
Step 2 : Recognize Loops on the Trace



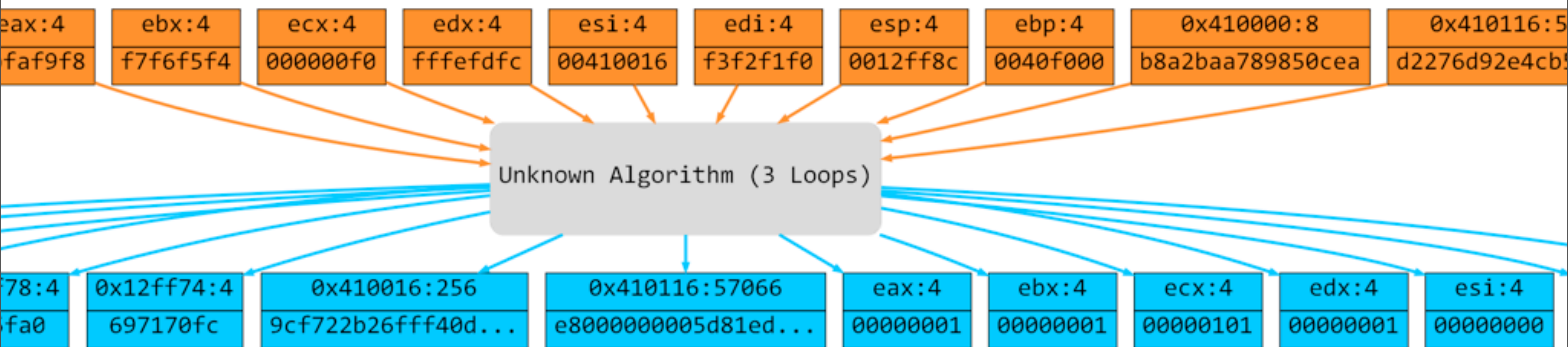
Step 3 : Define Loop I/O Parameters



Step 4 : Connect Loops With Data-Flow



Unknown Algorithm Extracted



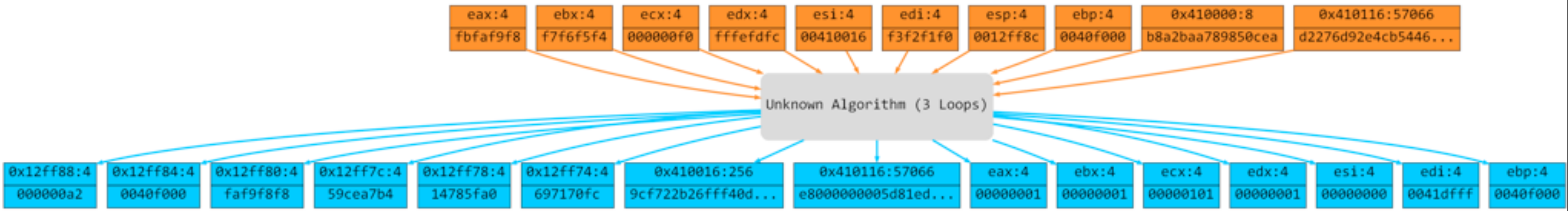
We still have the last mile to do...

Comparison Algorithm

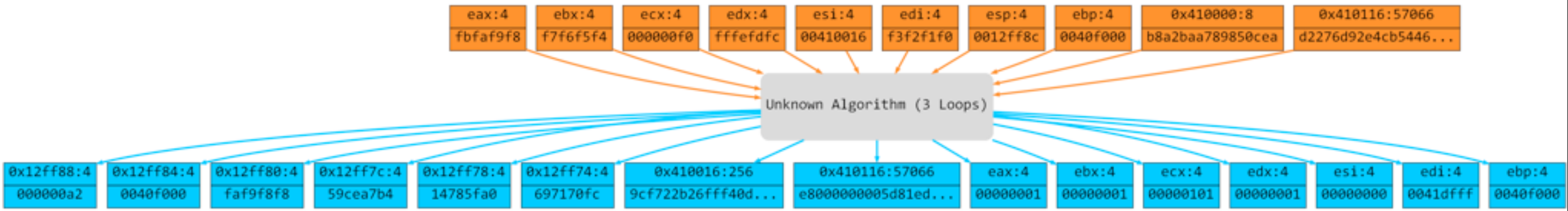
1. Build the set $\mathcal{I}$ of possible input values with all possible orderings of $\mathcal{A}$ input parameters.
2. Build the set $\mathcal{O}$ of possible output values in the same manner with $\mathcal{A}$ output parameters.
3. Evaluate each $\mathcal{S}$ function on all values in $\mathcal{I}$ and check if the result produced is in $\mathcal{O}$.

If yes, this is a success!

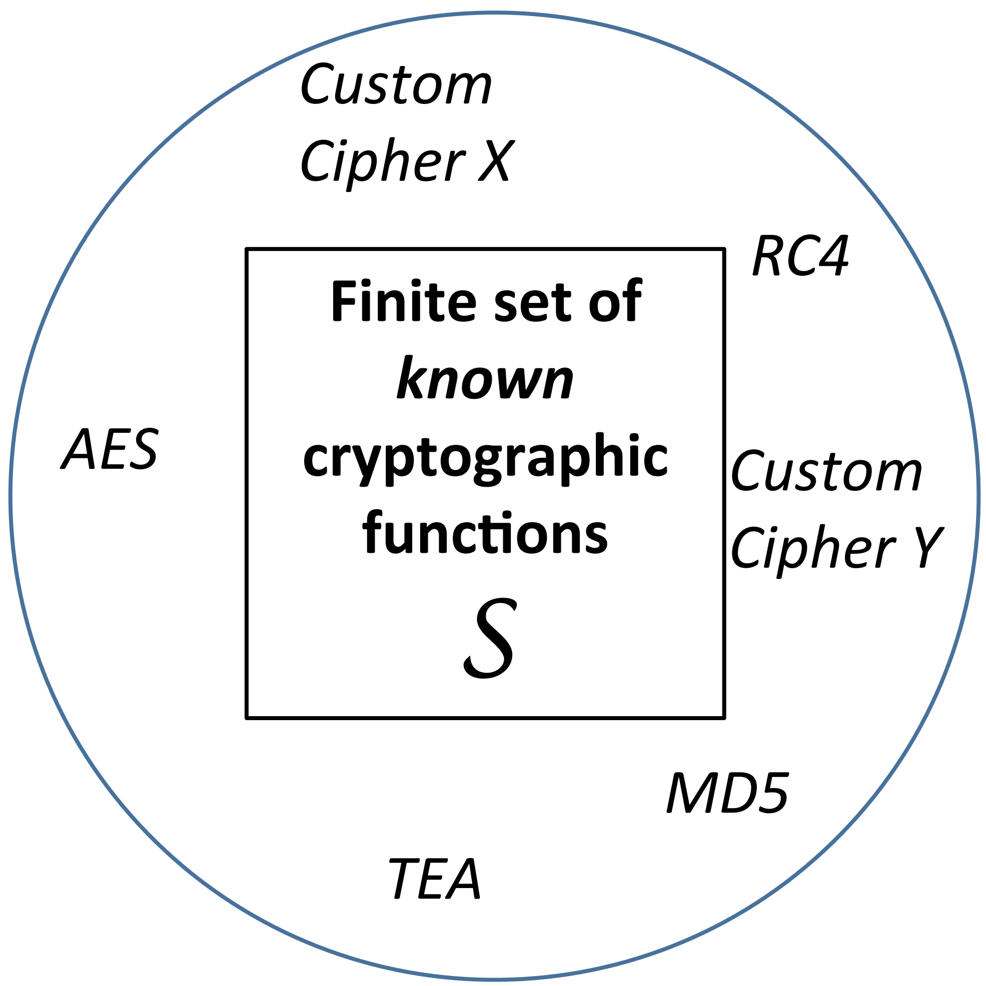
Input 1: unknown algorithm $\mathcal{A}$ with its parameter values



Input 1: unknown algorithm $\mathcal{A}$ with its parameter values



Input 2:



Question

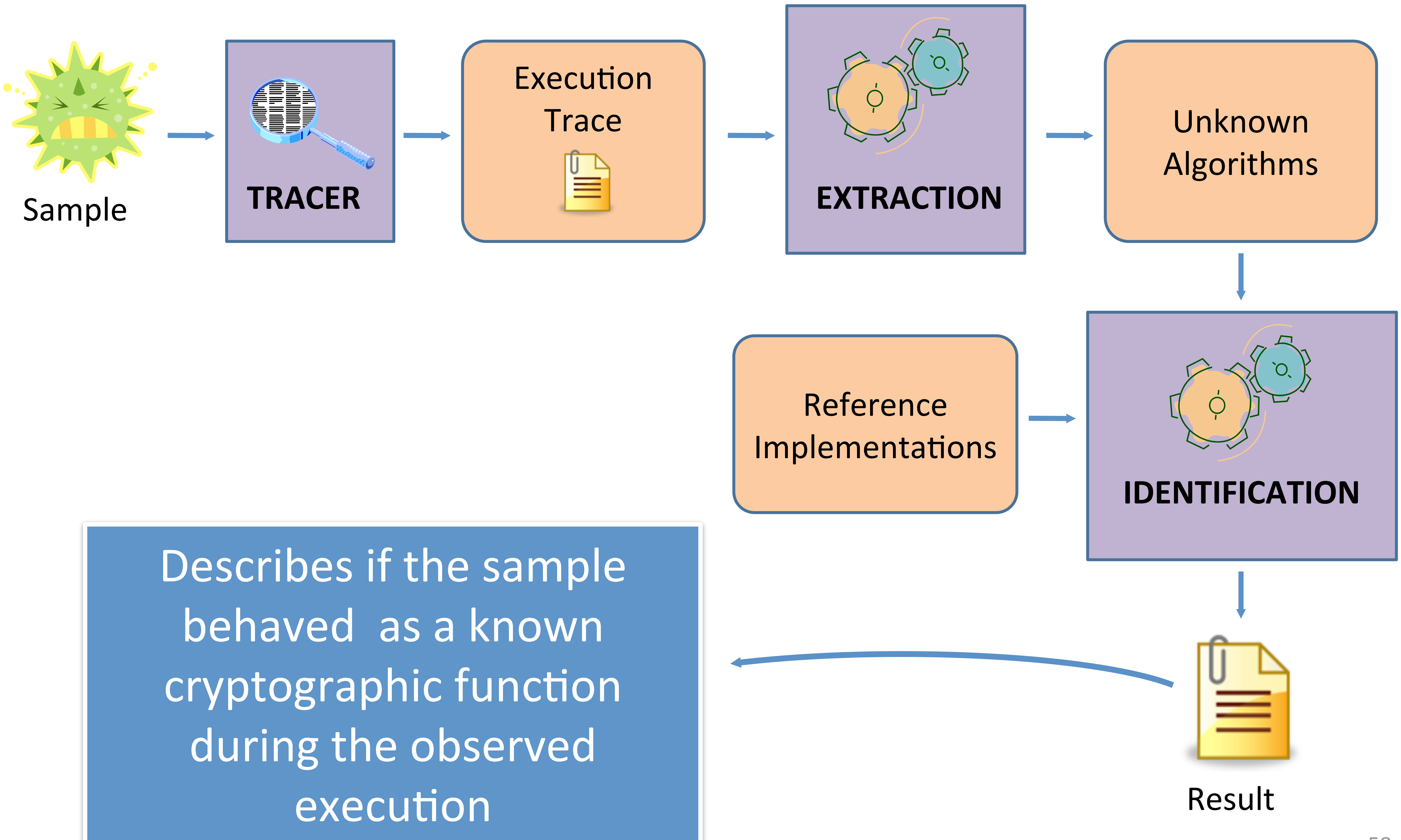
Question

- Some difficulties, for example:
  - –**Parameter division**: a same cryptographic parameter can be divided into several loop parameter.
  - –**Parameter number:** we collect more than the cryptographic parameters.

In practice, the complexity of this algorithm can be greatly reduced with some simple rules:

– Do not consider memory addresses as valid parameters.
– Do not consider common constant values (0,FF) as valid parameters.

Let’s Recap the Process



```
> Aligot identification module
> Start: 2012-10-10 13:50:38.963000
-----------------------------------
> No particular ciphers selected, test with all of them : ['tea', 'xte
a', 'russian_tea', 'rc4', 'aes_128_core', 'md5_core']

> Testing LDF 1 ...
  > Heuristic : Remove constants from parameters
  > Comparison with TEA... Fail!
  > Comparison with XTEA... Fail!
  > Comparison with RussianTEA... Fail!
  > Comparison with RC4...

!! Identification successful: RC4 encryption with:

 ==> Plain text (57066 bytes) : 0xd2276d92e4cb5446342196edd2a011b9ae1e
bbea51cebd7bc834d762d6ffa344f1f31d5b0ce29deb26dbc763fb23d23ee034148acf
63495ea0a117abdc4dc983b451b5a4d062207d95893199175366189 99a58f0cbd42ac4
9215809e5fa0c900ead39f5d9eb263de6fd5eb2b9c94c1f5a2a88ffd77def1f9f18d2e
512309309d9f3ec84656e30edad67cec88625d4c9476075c70e959cc912efc4126a9b7
959c7e6de2099a96e3c136f63317ffdf7ebc3f4a889ff331211f7f850accfb5d2e7278
cc96137c2f5eff27112646ec51d06c18bee4feb70c771ea577f7ec5bc73f1a0769fd8b
9f84c540ea1ec9fa563222d8919dd46e14b74ff56253fd994709dc0e...

 ==> Key (8 bytes) : 0xb8a2baa789850cea

 ==> Encrypted text (57066 bytes) : 0xe8000000005d81ed05104000582daad2
00008985ae11400080bd73274000007519c7859d1340002222222c7858c1340003333
3333e98200000033db64678b1e300085db780e8b5b0c8b5b1c8b1b8b5b08f8eb0a8b5b
348d5b7c8b5b3cf866813b4d5a7405e942010000 8bf303763c813e504500007405e930
010000899d791340008d859013400050ffb579134000e849010000e81e010000089859d
1340008d857d13400050ffb579134000e82c010000e80101000089858c1340008d858f
15400050ff959d134000e8e900000089857913400 08dbda11340008b9579134000e8d7
000000680280000 0ff95f71340008dbd771540005768008000006a006a04...
```

More Results

	$\mathcal{B}_1$	$\mathcal{B}_2$	*Storm**	*SBank**	$\mathcal{B}_3$	*Sal**	$\mathcal{B}_4$	$\mathcal{B}_5{}^{\dagger}$	$\mathcal{B}_6$	$\mathcal{B}_7{}^{\dagger}$	*Wal**	$\mathcal{B}_8{}^{\dagger}$
Aligot	TEA	TEA	RTEA	RTEA	RC4	RC4	AES	AES	MD5	MD5	AES, MD5	MOD MUL
CryptoSearcher	TEA	×	×	×	×	×	AES	×	MD5	×	×	×
Draca	TEA	×	×	TEA	×	×	×	×	MD5	SHA-1	×	×
Findcrypt2	×	×	×	×	×	×	AES	×	MD4	×	×	×
H&C Detector	TEA	×	×	TEA	×	×	×	×	MD5	SHA-1	×	×
Kerckhoffs	×	×	×	×	×	×	×	×	×	×	×	×
PEiD KANAL	TEA	×	×	TEA	×	×	AES	×	MD5	×	×	×
Signsrch	TEA	×	×	TEA	×	×	AES	×	×	×	×	×
SnDCryptoS	×	×	×	×	×	×	AES	×	MD5	×	×	×

(cf. paper)

Limitations

- As we analyze execution traces, we have to know how to exhibit interesting execution paths.
- The tool is as good as the reference implementation database.
- It is easy to bypass, like any program analysis technique.

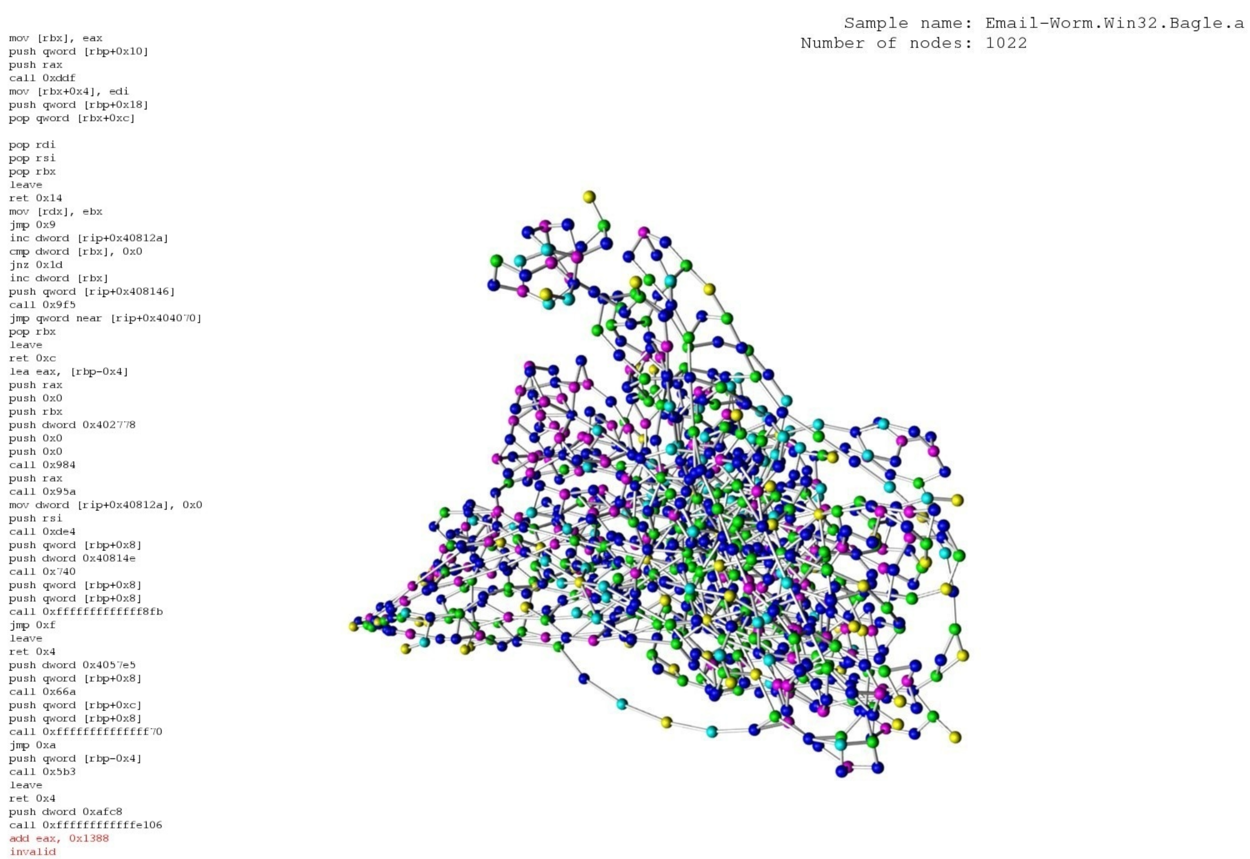
Future Work

- Extract reference implementations directly from binary programs.
- Implement other extraction criteria than our loop model.

Code is available at

https://code.google.com/p/aligot/

High Security Lab @ Nancy.fr



Other subjects

- Morphological analysis
- Botnet counter- attacks

Telescope & honeypots

In vitro experiment clusters

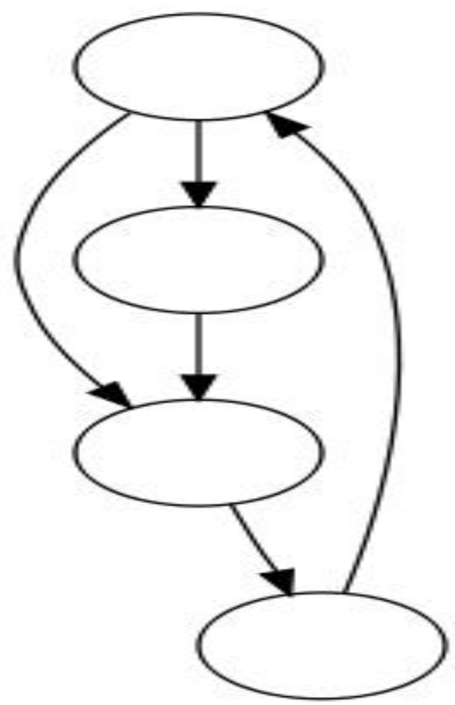
Thanks !

BONUS SLIDES

Morphological analysis in a nutshell

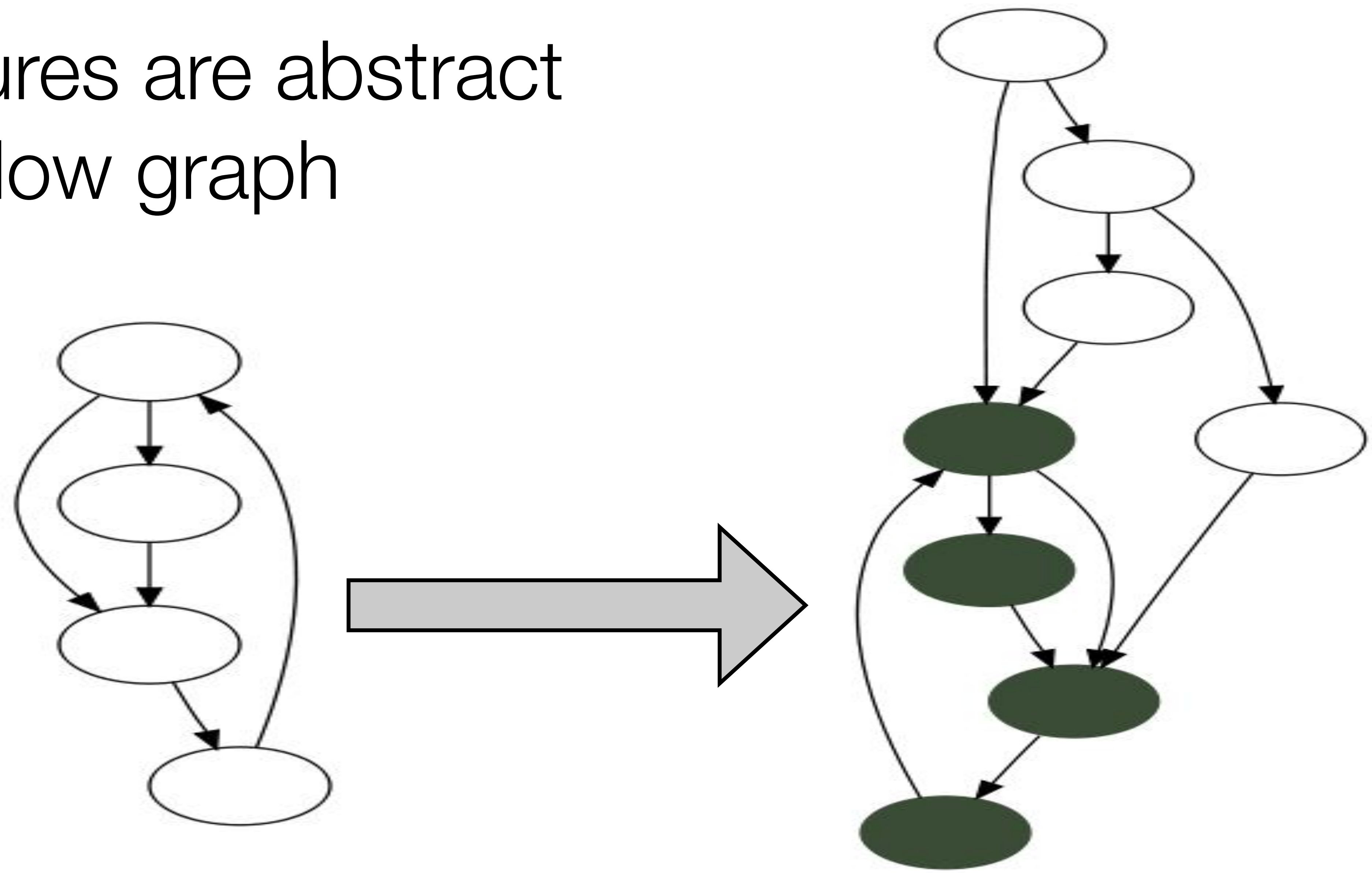
Morphological analysis in a nutshell

Signatures are abstract
flow graph



Morphological analysis in a nutshell

Signatures are abstract
flow graph

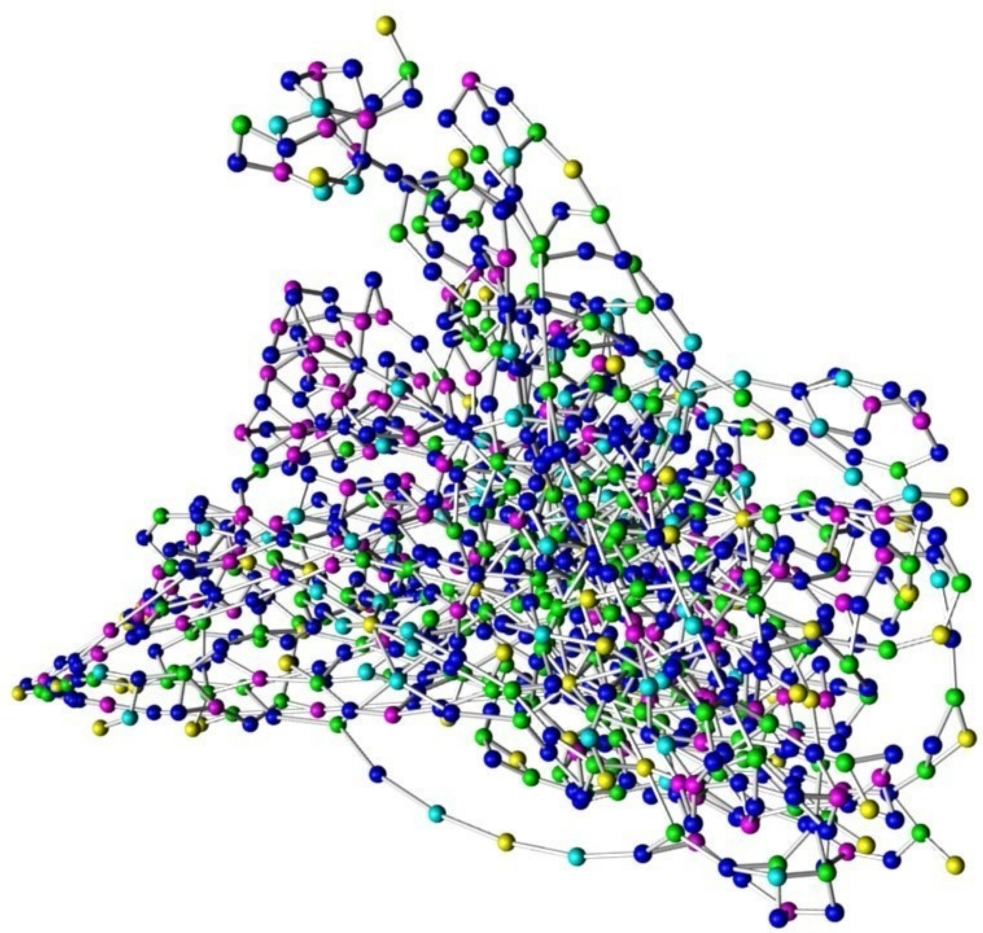


Detection of subgraph in program flow graph
abstraction

Automatic construction of signatures

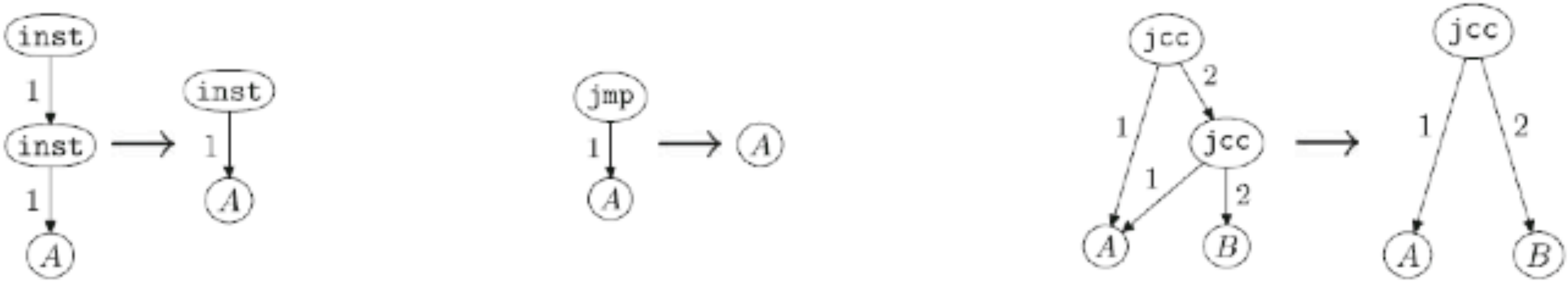
```
mov [rbx], eax
push qword [rbp+0x10]
push rax
call 0xddf
mov [rbx+0x4], edi
push qword [rbp+0x18]
pop qword [rbx+0xc]

pop rdi
pop rsi
pop rbx
leave
ret 0x14
mov [rdx], ebx
jmp 0x9
inc dword [rip+0x40812a]
cmp dword [rbx], 0x0
jnz 0x1d
inc dword [rbx]
push qword [rip+0x408146]
call 0x9f5
jmp qword near [rip+0x404070]
pop rbx
leave
ret 0xc
lea eax, [rbp-0x4]
push rax
push 0x0
push rbx
push dword 0x402778
push 0x0
push 0x0
call 0x984
push rax
call 0x95a
mov dword [rip+0x40812a], 0x0
push rsi
call 0xde4
push qword [rbp+0x8]
push dword 0x40814e
call 0x740
push qword [rbp+0x8]
push qword [rbp+0x8]
call 0xfffffffffffff8fb
jmp 0xf
leave
ret 0x4
push dword 0x4057e5
push qword [rbp+0x8]
call 0x66a
push qword [rbp+0xc]
push qword [rbp+0x8]
call 0xffffffffffffff70
jmp 0xa
push qword [rbp-0x4]
call 0x5b3
leave
ret 0x4
push dword 0xafc8
call 0xffffffffffffe106
add eax, 0x1388
invalid
```

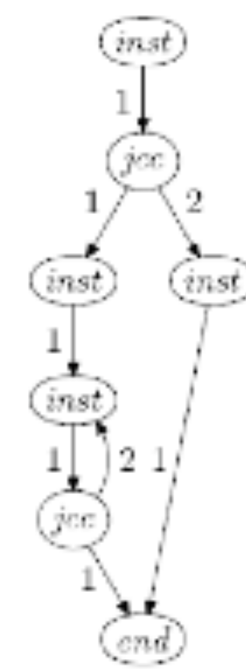
Sample name: Email-Worm.Win32.Bagle.a
Number of nodes: 1022



Reduction of signatures by graph rewriting

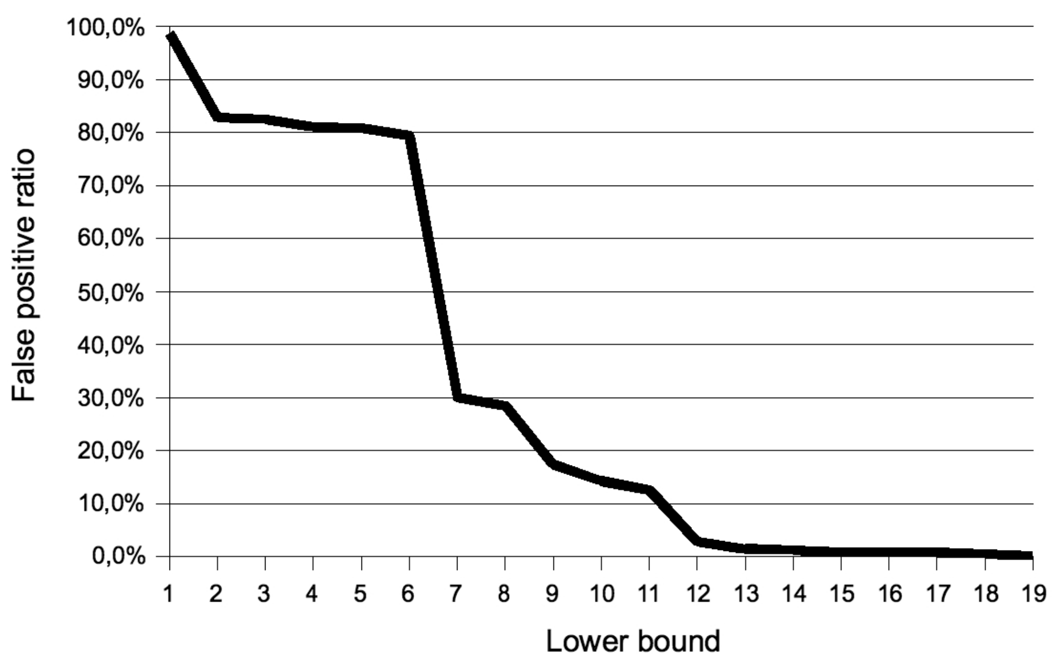
Concatenate instructions	Realign code	Merge jcc

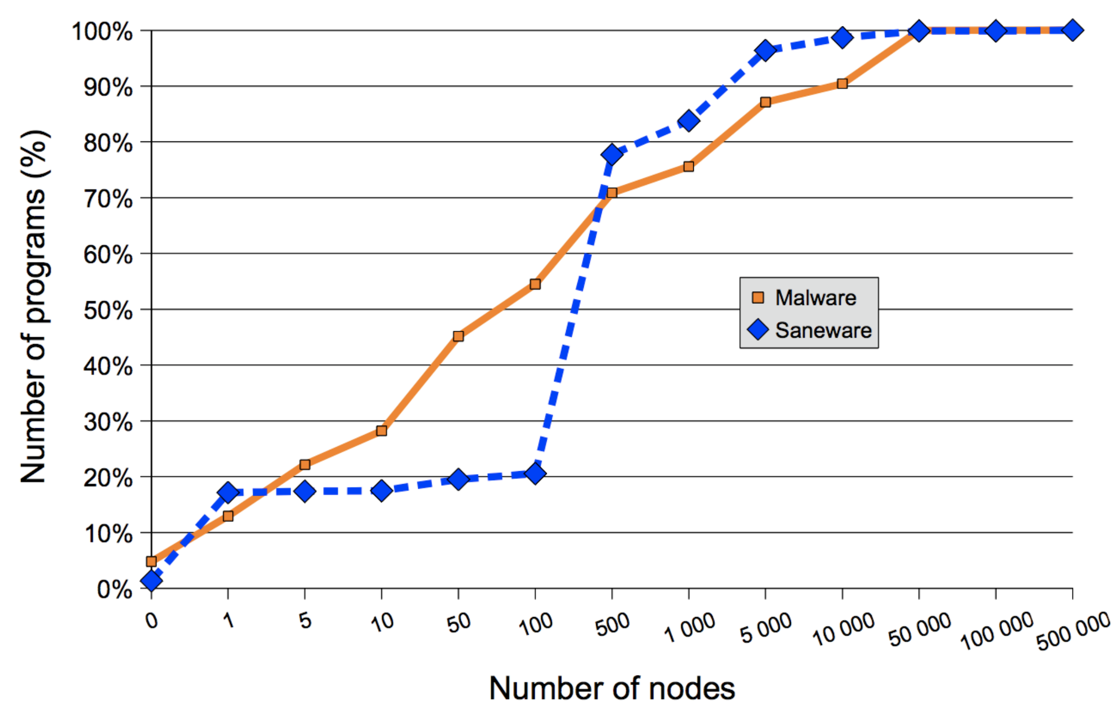


Original	One-one substitution	Substitution	Permutation	Jcc obfuscation	All in one	Normalised CFG
0 : cmp eax 0 1 : jne +7 2 : mov ecx eax 3 : dec ecx 4 : mul eax ecx 5 : cmp ecx 1 6 : jne -3 7 : jmp +2 8 : inc ecx 9 : ret	0 : cmp eax 0 1 : jne +7 2 : mov ecx eax 3 : *sub ecx 1* 4 : mul eax ecx 5 : cmp ecx 1 6 : jne -3 7 : jmp +2 8 : *add ecx 1* 9 : ret	0 : cmp eax 0 1 : jne +8 2 : *push eax* 3 : *pop ecx* 4 : dec ecx 5 : mul eax ecx 6 : cmp ecx 1 7 : jne -3 8 : jmp +2 9 : inc ecx 10 : ret	0 : cmp eax 0 1 : jne +7 2 : mov ecx eax 3 : dec ecx 4 : mul eax ecx 5 : cmp ecx 1 6 : jne -3 9 : *ret* 8 : inc ecx 9 : *jmp -2*	0 : cmp eax 0 1 : jne +9 2 : mov ecx eax 3 : dec ecx 4 : mul eax ecx 5 : cmp ecx 2 6 : *ja -3* 7 : cmp ecx 1 8 : *jne -5* 9 : jmp +2 10 : inc ecx 11 : ret	0 : cmp eax 0 1 : *je +2* 2 : *jmp +10* 2 : *push eax* 3 : *pop ecx* 4 : *sub ecx 1* 5 : mul eax ecx 6 : *cmp ecx 2* 7 : *ja -3* 8 : cmp ecx 1 9 : *jne -5* 10 : *ret* 11 : *add ecx 1* 12 : *jmp -2*	

Morphological detection : Results

- False negative
  - No experiment on unknown malware
  - Signatures with < 18 nodes are potential false negative
  - Restricted signatures of 20 nodes are efficient
- Less than 3 sec. for signatures of 500 nodes





Conclusion about morphological detection

- Benchmarks are good
- Pro
  - More robust on local mutation and obfuscation
  - Detect easily variants of the same malware family
  - Try to take into account program semantics
  - Quasi-automatic generation of signatures
- Cons
  - Difficult to determine flow graph statically of self-modifying programs
  - Use of combination of static and dynamic analysis

Reference

- Guillaume Bonfante, Matthieu Kaczmarek and Jean-Yves Marion, Architecture of a malware morphological detector, *Journal in Computer Virology*, Springer 2008.

- Recon 2012 and Malware 2012

Performances

	Sality 1	Sality 2
Trace Size (instructions)	~1M	~4M
Time To Trace	5mn	10mn
Time To Extract Crypto Algoritm	4h	10h
Time To Identify	3mn	4mn

- The tool is just a PoC, no optimization **at all**.

- When the analysts knows where the algorithm is, it will reduce the trace size.

Existing Tools For Crypto Identification

Tools	Answers on Sality sample
Crypto Searcher	Ø
Draca v0.5.7b	Ø
Findcrypt v2	Ø
Hash & Crypto Detector v1.4	Ø
PEiD KANAL v2.92	Ø
Kerckhoffs	Ø
Signsrch 0.1.7	Ø
SnD Crypto Scanner v0.5b	Ø