Behavior abstraction in Malware analysis

Jean-Yves Marion
LORIA
INRIA

Joint works with

Philippe Beaucamps

Isabelle Gnaedig

Daniel Reynaud (Berkeley U.)

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

✴ So 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

✳ So how to detect a malware ?

✳ How to protect a system from 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

✳ So how to detect a malware ?

✳ How to protect a system from a malware ?

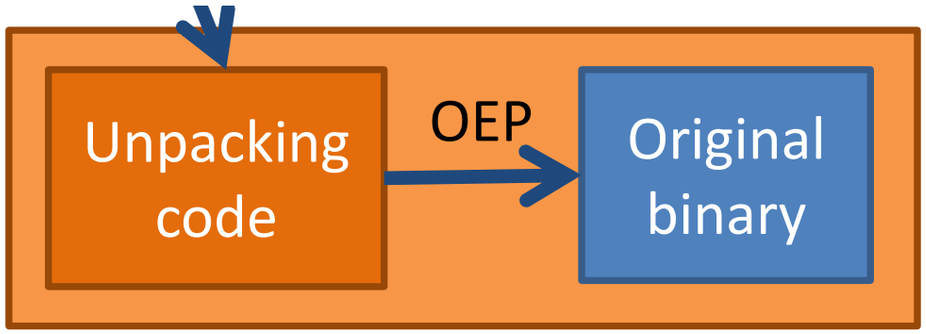
Code protection

Detection is hard because malware are protected

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

Protections: Self-Modification

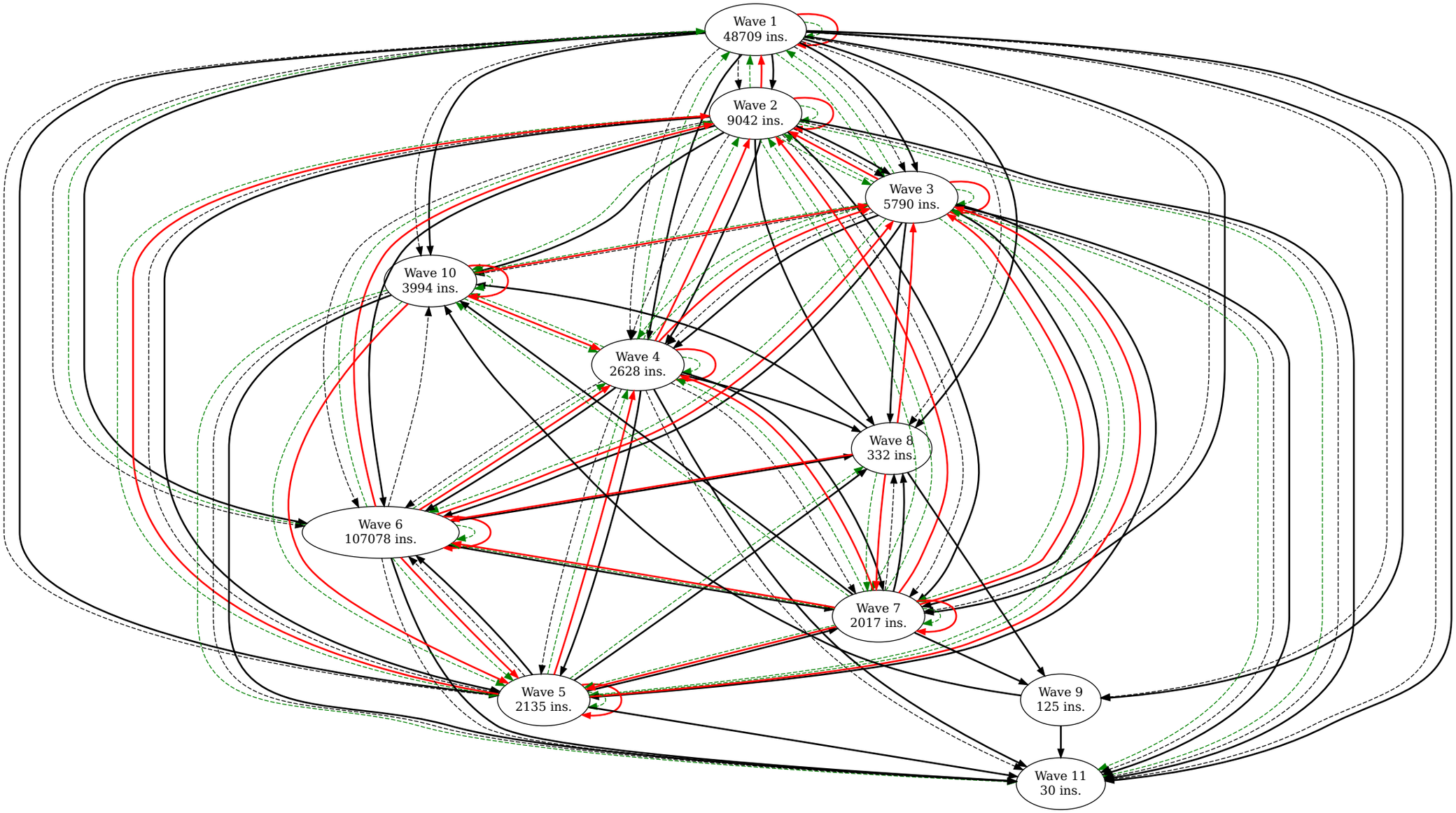
- A lot of malware families use home-made obfuscations, like packers to protect their binaries, following a standard model



➡ It is difficult to perform a static analysis

➡ Dynamic analysis by code monitoring (emulation, instrumentation, ...)

A program generating codes



Protections: Obfuscation

✓ Several possible implementations of a high level action

Protections: Obfuscation

✓ Several possible implementations of a high level action

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»)

Malware detection methods in a tiny nutshell

Malware detection by string scanning

- Signature is a regular expression denoting a sequence of bytes

Worm.Y
Your mac is now under our control !

- Signature : «Your * is now under our control

Worm.Y
Your PC is now under our control !

Malware detection by string scanning

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 :

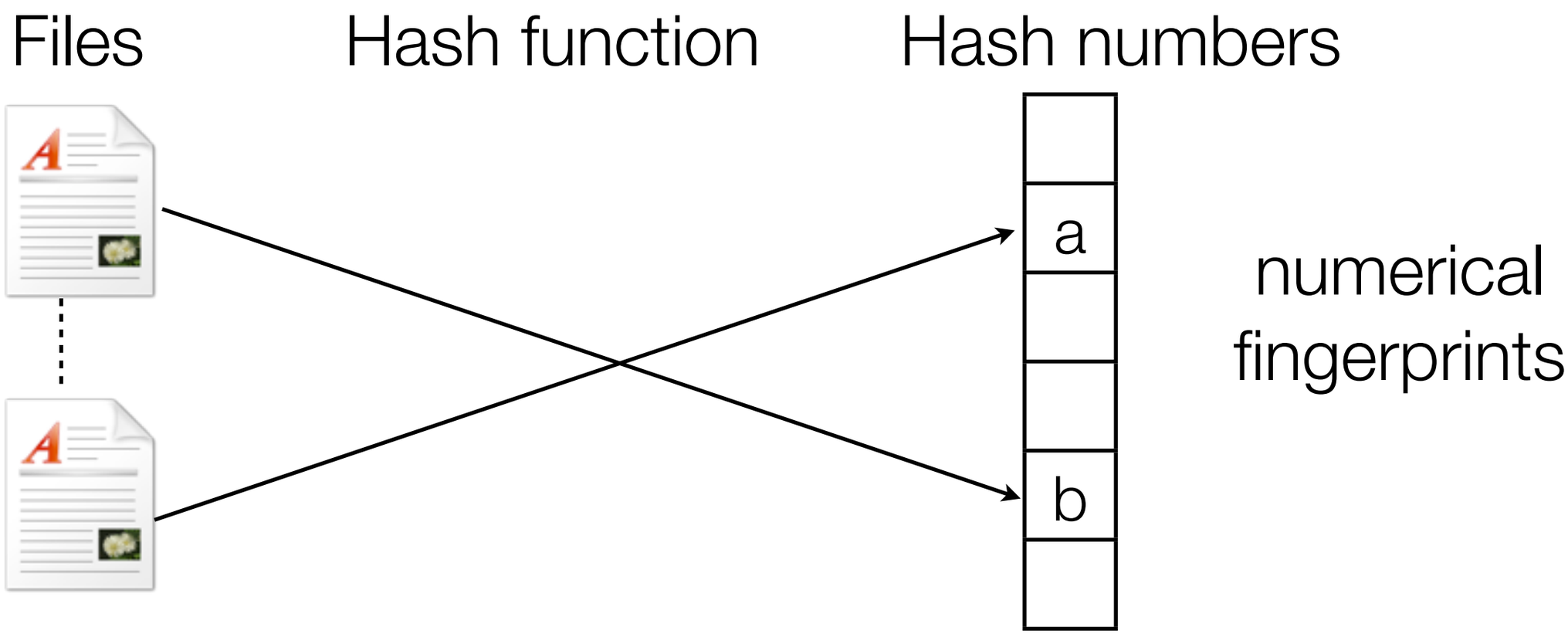
- Signature are quasi-manually constructed
- Signatures are not robust to malware protections

  ➡ Mutations

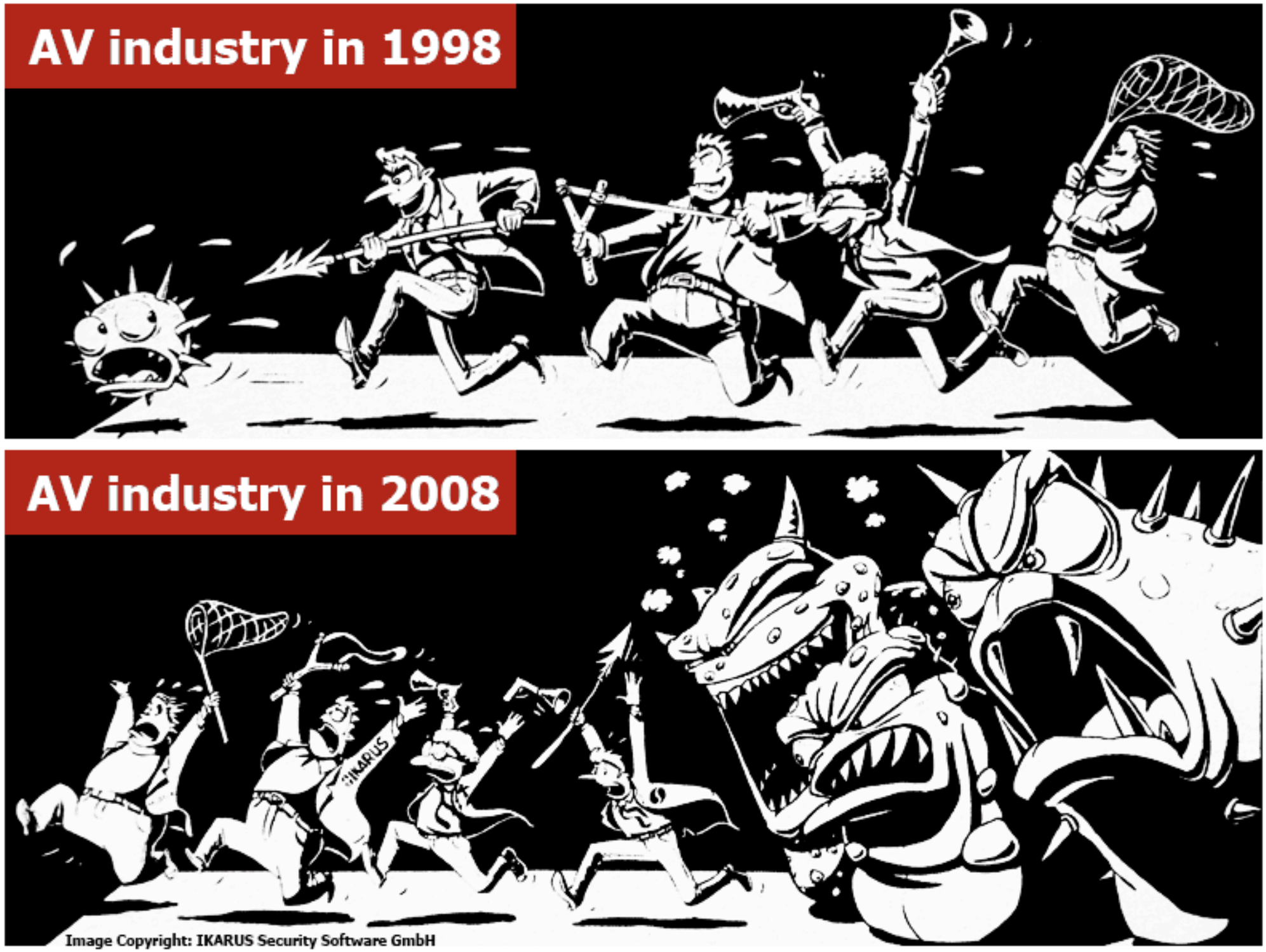
  ➡ Code obfuscations

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)


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

Detection based on higher level of abstraction

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

-Bonfante, Kaczmarek and M. 2009 : Morphological analysis

Behavioral analysis and detection

Behavioral detection

- Identification of a sequence of actions :
    - System calls or library calls
    - File systems interactions
    - Network interactions
    - Sequence of instructions

Allaple excerpt

```
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 excerpt

```
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 excerpt

```
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 excerpt

```
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 excerpt

```
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 excerpt

```
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];
```

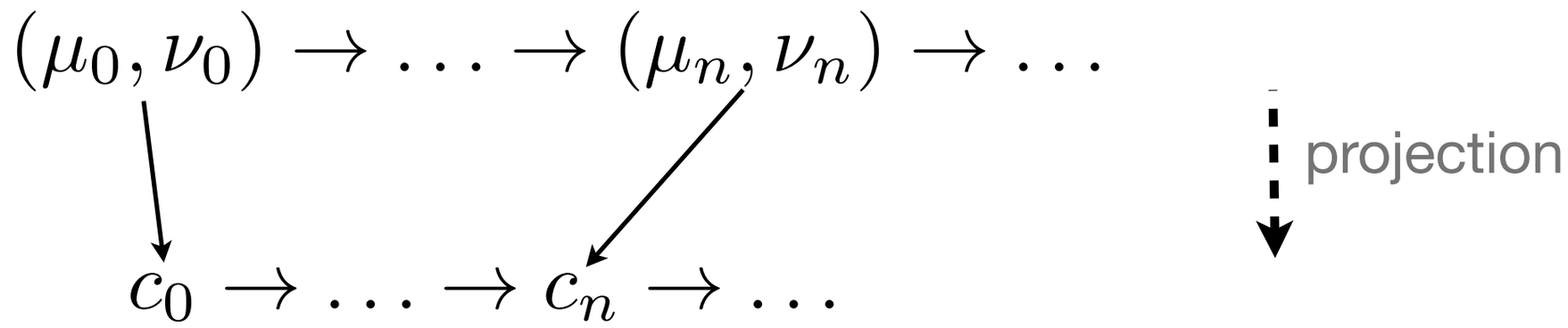
```
/* Behavior pattern: ping of a remote host */
hIcmp = IcmpCreateFile();
for(int i = 0; i < 2; ++i)
  IcmpSendEcho(hIcmp, ipaddr, icmpData, 10,
               NULL, reply, 128, 1000);
IcmpCloseHandle(hIcmp);

/* 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;
}
}
```

Example execution trace of library calls:

```
...GetLogicalDriveStrings.GetDriveType.FindFirstFile.FindFirstFile.
                                    FindNextFile...
```

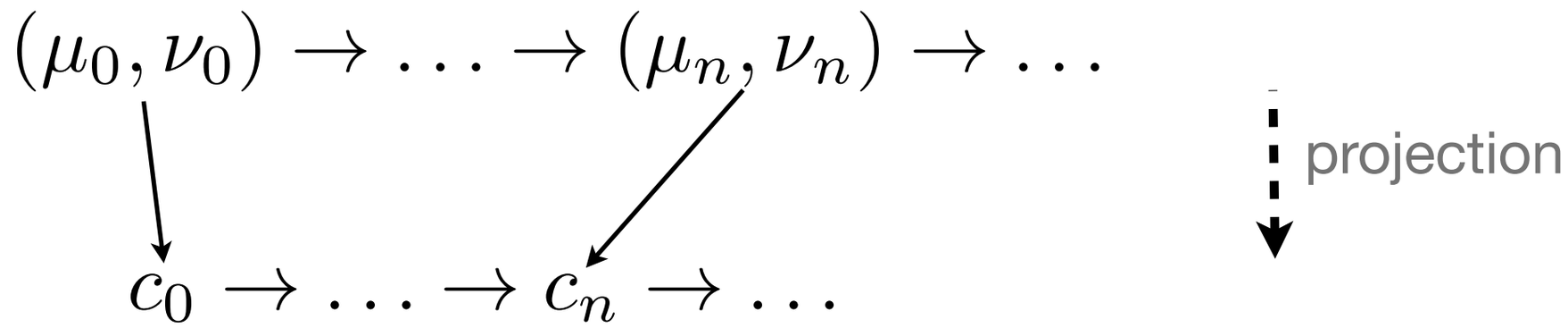
Program Traces

An execution is a sequence of configurations



Program Traces

An execution is a sequence of configurations



- A trace is a sequence of actions
- Approximation by a regular language
- A trace automaton is a finite approximation of a trace language

Program Traces

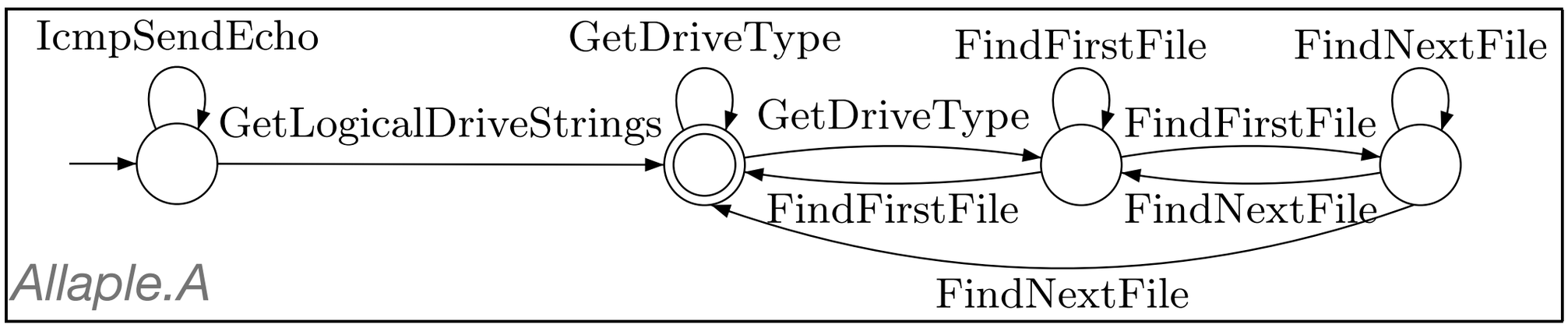
An execution is a sequence of configurations

$$(\mu_0, \nu_0) \to \ldots \to (\mu_n, \nu_n) \to \ldots$$

projection

A trace $c_0 \to \ldots \to c_n \to \ldots$

- A trace is a sequence of actions
- Approximation by a regular language
- A trace automaton is a finite approximation of a trace language



Computing program traces

1.Dynamic analysis

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

2.Static analysis

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

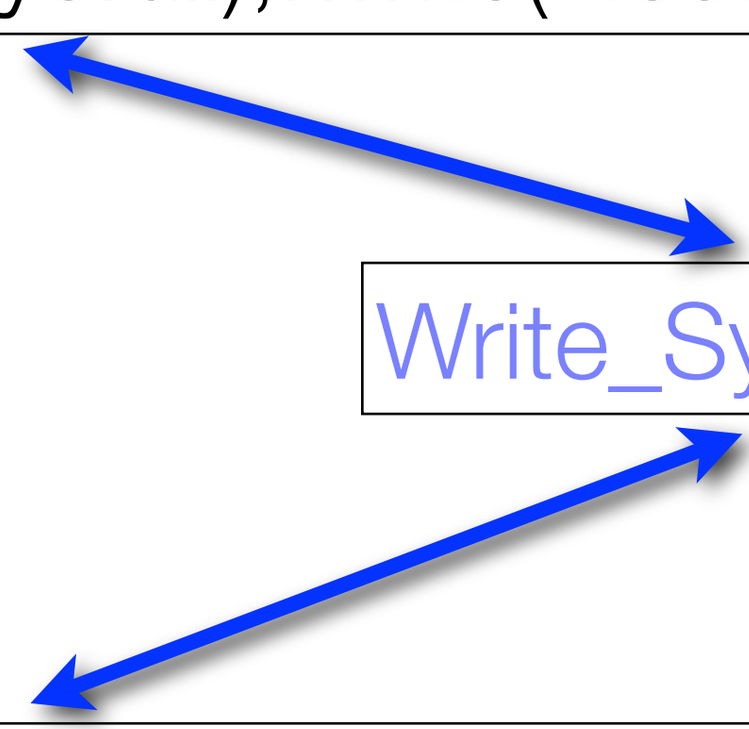
Behavior abstraction

Goal : Provide a behavior analysis technique by expressing traces in term of high level, implementation-independant functionalities

Behavior abstraction

Goal : Provide a behavior analysis technique by expressing traces in term of high level, implementation-independant functionalities

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



Write_System_File

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

Behavior abstraction

Goal : Provide a behavior analysis technique by expressing traces in term of high level, implementation-independant functionalities

Our works

- Expressing set of traces by regular languages from static or dynamic analysis
- Abstracting behavior patterns by string rewriting systems
- Efficient analysis (quasi-linear time)
-Detection of several abstract behaviors from a set of traces

Behavior abstraction

Goal : Provide a behavior analysis technique by expressing traces in term of high level, implementation-independant functionalities

Our works

- Expressing set of traces by regular languages from static or dynamic analysis
- Abstracting behavior patterns by string rewriting systems
- Efficient analysis (quasi-linear time)
-Detection of several abstract behaviors from a set of traces

Related works

- Martignoni et al. 2008: multi-layered abstraction on a single trace
-Jacob et al., 2009: low-level functionalities, exponential-time detection

Behavior patterns

Behavior patterns

A behavior pattern is a monadic String Rewriting System

GetLogicalDriveString.GetDriveType.(FindFirstFile+FindFirstFileEx)
→ Scandrive

Behavior patterns

A behavior pattern is a monadic String Rewriting System

GetLogicalDriveString.GetDriveType.(FindFirstFile+FindFirstFileEx)
→ Scandrive

- A behavior pattern B is a SRS :

$B_1 \rightarrow \lambda_1$

$B_n \rightarrow \lambda_n$

Monadic String rewriting systems

Rules of a monadic SRS are of the form

$$\alpha \to \beta \qquad \text{where} \qquad |\alpha| > |\beta| \qquad |\beta| \leq 1$$

Theorem:
The set of descendants of a regular language, by using monadic SRS is regular

(Book & Otto)

Abstract trace language

Abstract a trace language *L* by reducing it w.r.t. a behavior pattern B

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

Abstract trace language

Abstract a trace language *L* by reducing it w.r.t. a behavior pattern B

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

Theorem : Let B be a regular behavior pattern and L be a trace language.

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

There is a quadradic-time procedure to compute $L^{\downarrow}$.

Abstract trace language

Abstract a trace language *L* by reducing it w.r.t. a behavior pattern B

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

Theorem : Let B be a regular behavior pattern and L be a trace language.

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

There is a quadradic-time procedure to compute $L^{\downarrow}$.

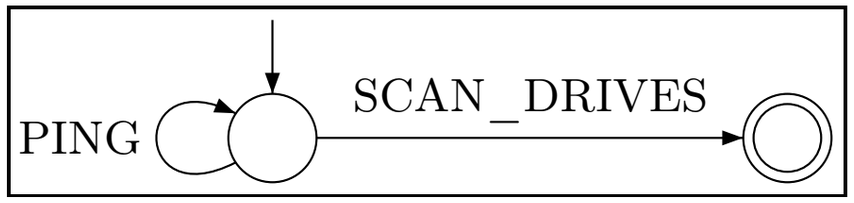
GetLogicalDriveString.GetDriveType.FindFirstFile.FindNextFile....

Scandrive.FindNextFile...

Malicious behavior detection

- A malicious behavior (signature) is a regular behavior pattern B

Allaple.A signature:

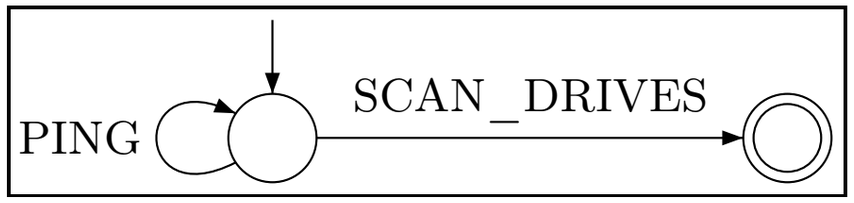


PING.*.SCAN_DRIVE

Malicious behavior detection

- A malicious behavior (signature) is a regular behavior pattern B

Allaple.A signature:



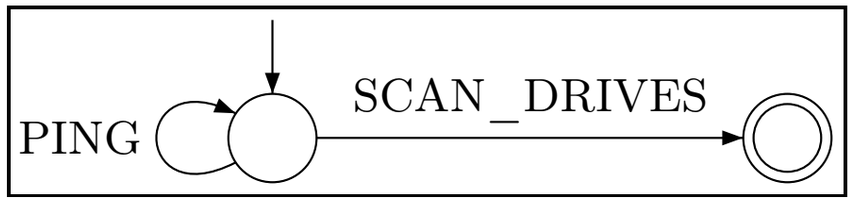
PING.*.SCAN_DRIVE

- **Detection** : comparing the abstract trace language $L^{\downarrow}$ with B

Malicious behavior detection

- A malicious behavior (signature) is a regular behavior pattern B

Allaple.A signature:



PING.*.SCAN_DRIVE

- **Detection** : comparing the abstract trace language $L^{\downarrow}$ with B

Theorem : Let B a regular behavior pattern and $L^{\downarrow}$ an abstract trace language.

There is a linear-time procedure deciding $L^{\downarrow}$ is infected by B.

On going researches and conclusions

Tracking arguments ...

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

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

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

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

Tracking arguments ...

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

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

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

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

- Bad traces satisfy a FO-LTL formula

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

- Behavior abstraction is made by term rewriting systems

- Linear detection algorithms based on tree-automata

A keylogger or a sms message leaking app

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

Keystroke or IMEI capture

Network send functionality

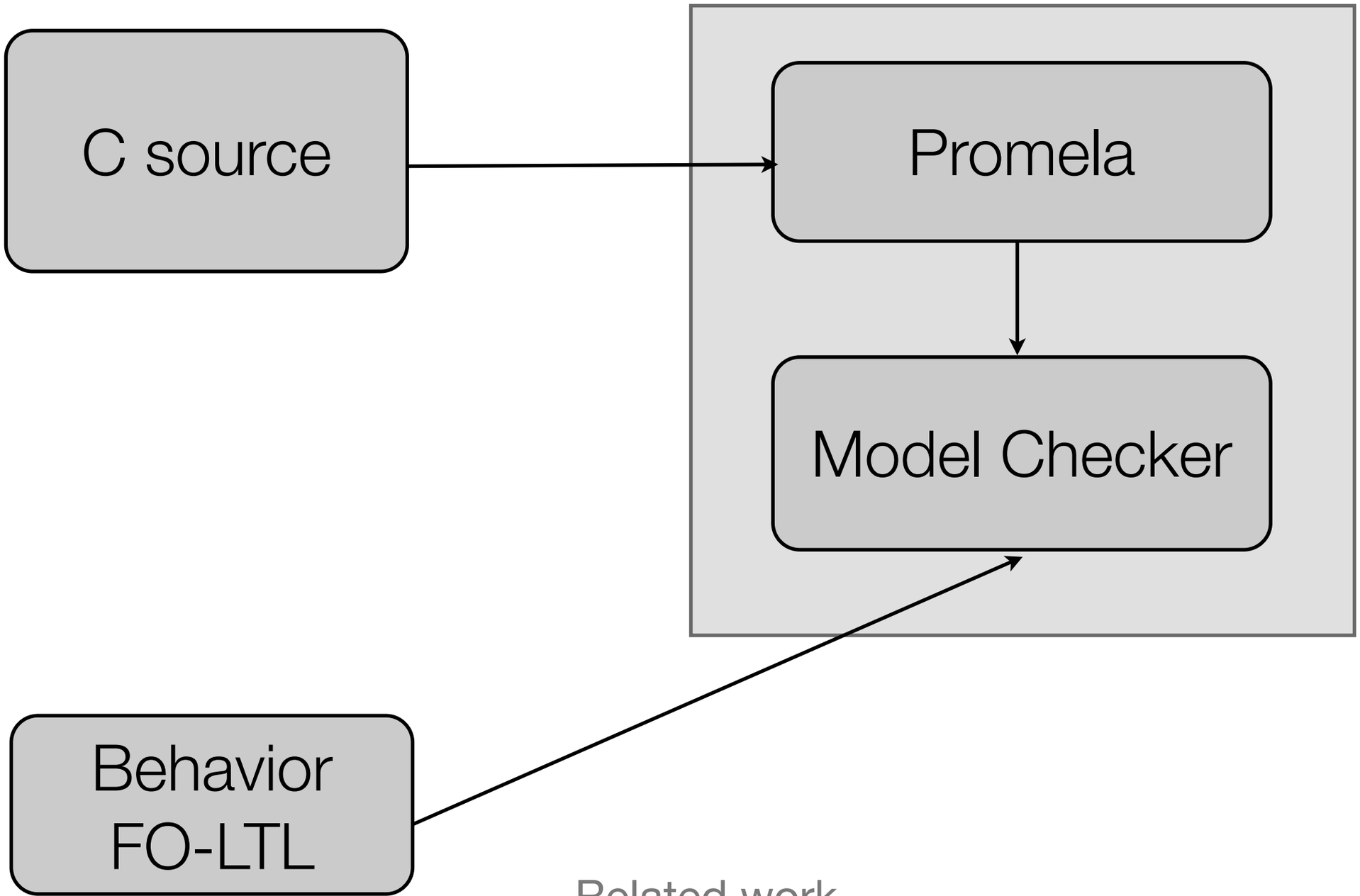
invalidated

$$\lambda_{steal}(x) := GetAsyncKeyState(x) \vee (RegisterDev(\texttt{KBD}, \texttt{SINK}) \odot GetInputData(x, \texttt{INPUT})) \vee (\exists y.\, SetWindowsHookEx(y, \texttt{WH\_KEYBOARD\_LL}) \wedge \neg UnhookWindowsHookEx(y)\, \mathbf{U}\, HookCalled(y, x)) \vee \exists y. TelephonyManager\_getDeviceId(x, y).$$

$$\lambda_{leak}(x) := \exists y, z.\, sendto(z, x, y) \vee \exists y, z.\, (connect\,(z, y) \wedge \neg close(z)\ \mathbf{U}\ send(z, x)) \vee \exists c, s.\, HttpURLConnection\_getOutputStream(s, c) \wedge \neg OutputStream\_close(s)\, \mathbf{U}\, OutputStream\_write(s, x)$$

$$\lambda_{inval}(x) := free(x) \vee \exists y.\, sprintf_0(x, y) \vee GetInputData(x, \texttt{INPUT}) \vee \ldots$$

Tool chains



Related work
Kinder & al, Detecting malicious code by model checking

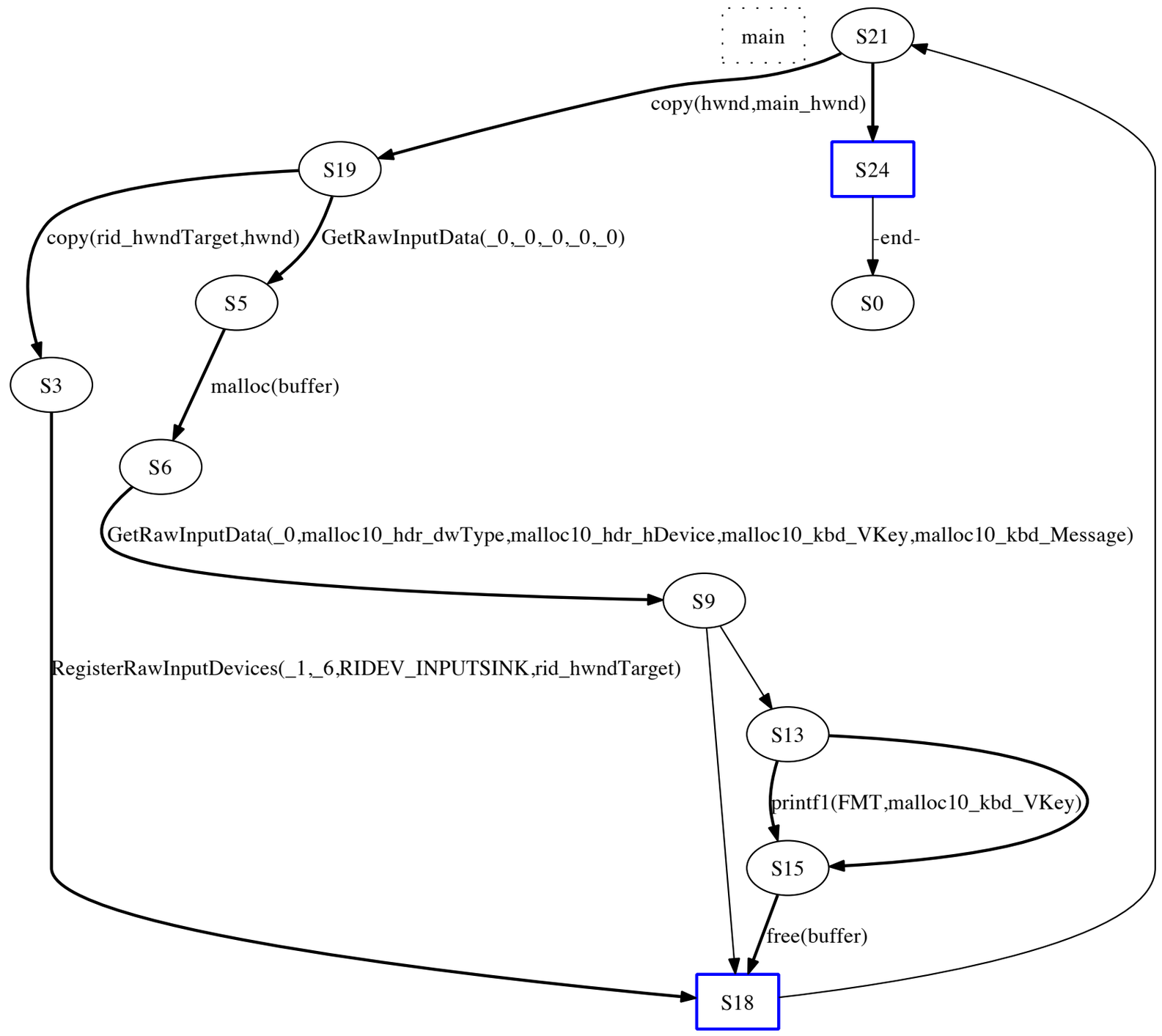
A C Keylogger from EADS

```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;
  }
  /* ... */
}
```

Promela translation

```
mtype = { CALL_COPY, CALL_RegisterRawInputDevices, CALL_GetRawInputData,
          CALL_malloc, CALL_printf1, CALL_free }
chan lib_call = [0] of {mtype, int, int, int, int, int, int};
proctype lib_loop() {
  xr lib_call;
  do
  :: lib_call ? _,_,_,_,_,_,_;
  od;
}
init {
  run lib_loop();
  run main();
}

#define COPY(v1, v2) \
  lib_call ! CALL_COPY, v1, v2;
#define RegisterRawInputDevices(pRID_usUsagePage, pRID_usUsage, pRID_dwFlags,\
  pRID_hwndTarget) \
  lib_call ! CALL_RegisterRawInputDevices, pRID_usUsagePage, pRID_usUsage,\
  pRID_dwFlags, pRID_hwndTarget;
/* ... */
```

Behavior graph



Malicious behavior detection

- Detection of malicious behaviors
  - Abstraction provides a high level notion of signature which is robust with respect to some functional obfuscation methods
  - Behavior analysis from a set of traces which comes from static or dynamic analysis
  - Detections algorithms are efficient based on (word/tree) automata techniques
  - Tests with Allaple, Virus, Agent, Rbot, Afcore and Mimail

High Security Lab @ Nancy

lhs.loria.fr

Telescope & honeypots
In vitro experiment clusters

Thanks !