A Generic Approach for Protecting Java Card^{TM} Smart Card Against Software Attacks

Guillaume BOUFFARD

Smart Secure Devices (SSD) Team
XLIM/University of Limoges

PhD Defence

10^{th} of October, 2014

Outline

Outline

The Smart Card

- Tamper-Resistant Computer;
- Securely stores and processes information;
- Used in our everyday life:
  - Credit Card;
  - (U)SIM Card;
  - Health Card (French Vitale card);
  - Pay TV;
  - ...
- Most of the smart cards are based on Java Card technology.

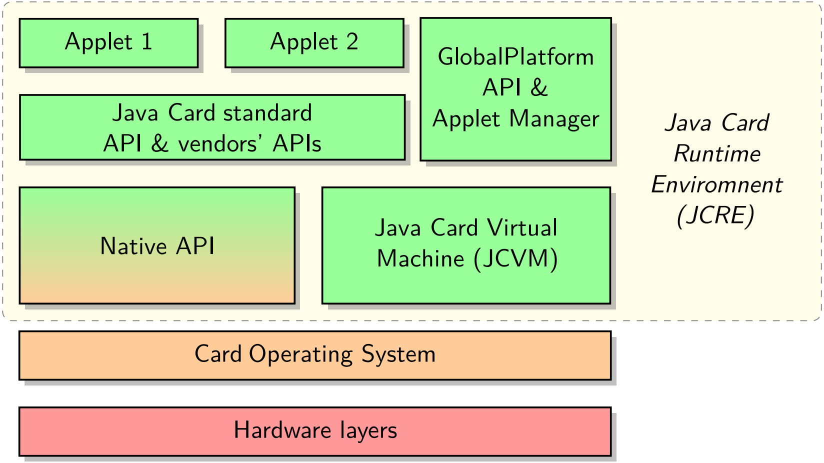
This device contains sensitive data

Java Card Technology

- Created by Schlumberger in 1996;
- **Specified** by Oracle;
- Provide a **friendly** environment to develop **secure** Java-applications.

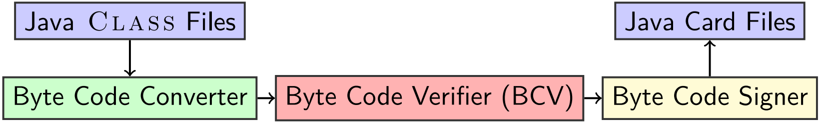
[From B. Basquin's presentation at Cartes ASIA 2014]

Java Card Technology (Cont.)

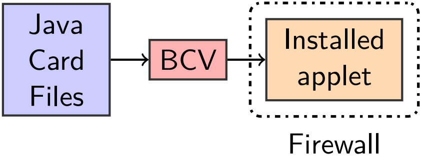


Java Card Security Model

- Off-card security



- On-card security



Java Card Attacks

Physical attacks

- Side Channel attacks (timing attacks, power analysis attack, etc.);
- Fault attacks (electromagnetic injection, laser beam injection, etc.).

Logical attacks

- Execution of malicious Java Card byte codes.

Combined attacks

- Mix of physical and logical attacks.

Problematic

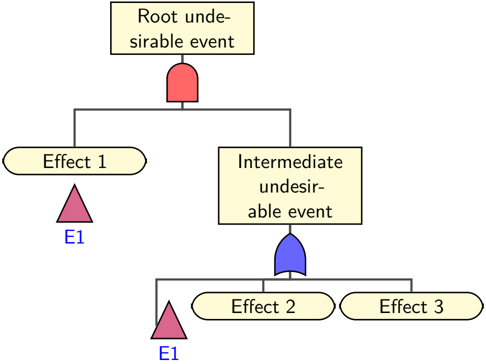
- Inductive Approach:
  - 1 attack = 1 countermeasure;
  - **Bottom-up** approach.

Problematic

- Inductive Approach:
  - 1 attack = 1 countermeasure;
  - **Bottom-up** approach.
- Thesis Objectives:
  - Find and prevent each undesirable events;
  - Global vision to protect the smart card's assets;
  - Design a **top-down analytic approach**.

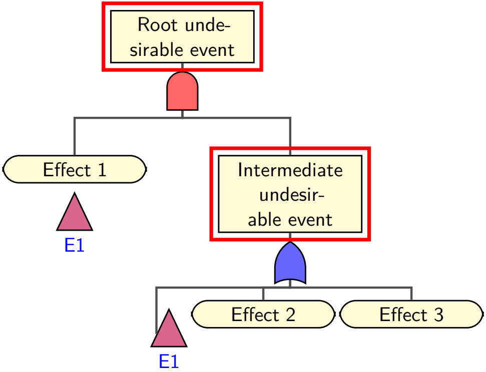
Outline

The Fault Tree Analysis (FTA)



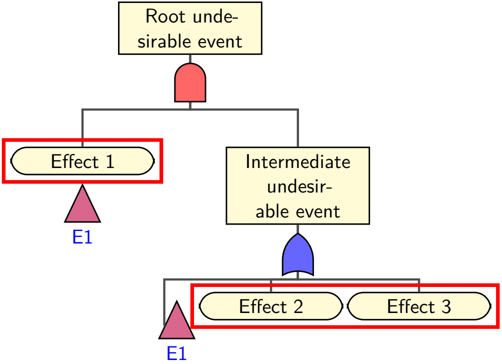
- Undesirable events;
- Initial causes;
- Gate connectors.

The Fault Tree Analysis (FTA)



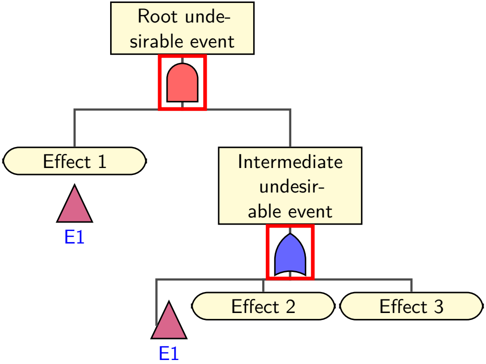
- **Undesirable events;**
- Initial causes;
- Gate connectors.

The Fault Tree Analysis (FTA)



- Undesirable events;
- **Initial causes;**
- Gate connectors.

The Fault Tree Analysis (FTA)



- Undesirable events;
- Initial causes;
- **Gate connectors.**

Smart Card's Assets

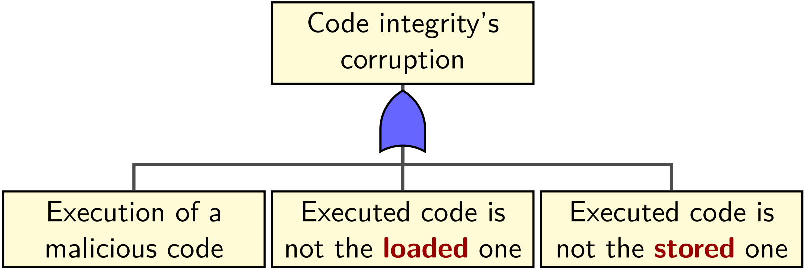
- The smart card's assets are the **code** and the **data**;
- Security properties:
  - Integrity;
  - Confidentiality;
- Undesirable events can affect:
  - Code integrity;
  - Data integrity;
  - Code confidentiality;
  - Data confidentiality;

Smart Card's Assets

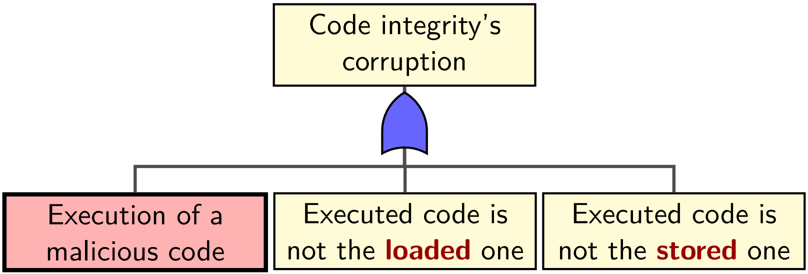
- The smart card's assets are the **code** and the data;
- Security properties:
  - Integrity;
  - Confidentiality;
- Undesirable events can affect:
  - **Code integrity**;
  - Data integrity;
  - Code confidentiality;
  - Data confidentiality;

An attack offers the execution of a malicious byte code.

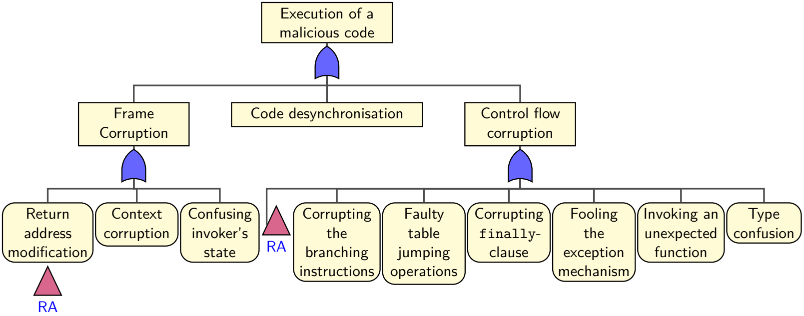
Code Integrity's Fault Tree



Code Integrity's Fault Tree

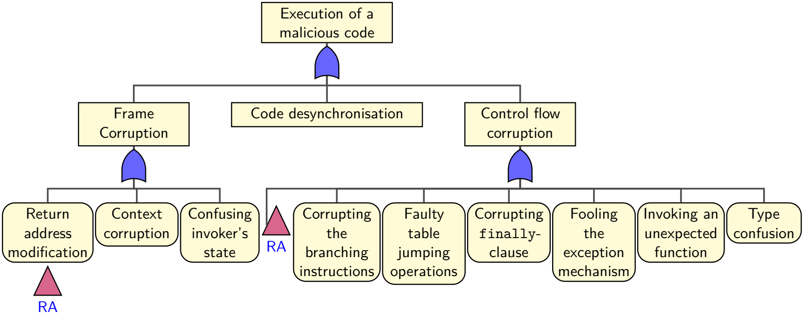


Execution of a malicious code



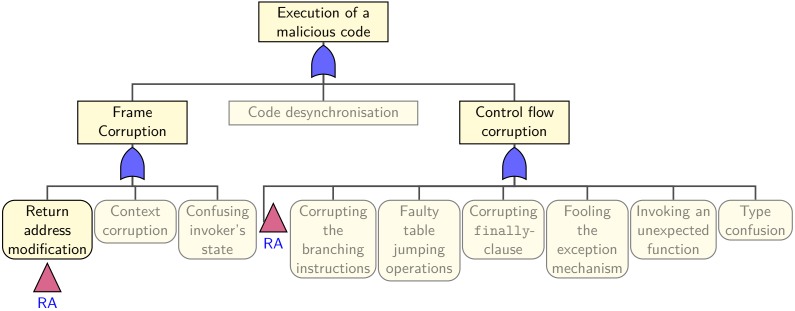
- Published in [Bouffard et al., SAFECOMP 2013];

Execution of a malicious code



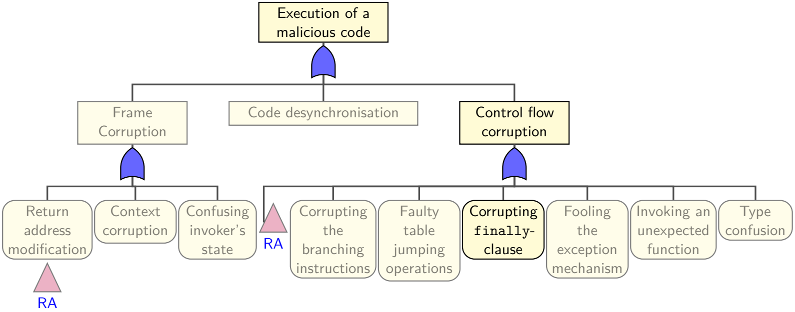
- Published in [Bouffard et al., SAFECOMP 2013];
- For this presentation, **two vulnerabilities** will be introduced:
  - Modifying the method's return address;
  - Corrupting the `finally`-clause.
- Thanks to minimal cut set, a countermeasure to protect the execution flow was developed: the security automatons.

Execution of a malicious code



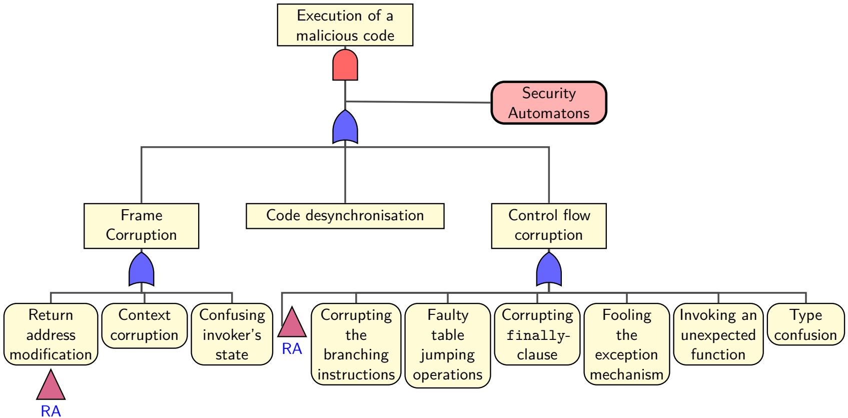
- Published in [Bouffard et al., SAFECOMP 2013];
- For this presentation, **two vulnerabilities** will be introduced:
  - **Modifying the method's return address;**
  - Corrupting the `finally`-clause.
- Thanks to minimal cut set, a countermeasure to protect the execution flow was developed: the security automatons.

Execution of a malicious code



- Published in [Bouffard et al., SAFECOMP 2013];
- For this presentation, **two vulnerabilities** will be introduced:
  - Modifying the method's return address;
  - **Corrupting the `finally`-clause.**
- Thanks to minimal cut set, a countermeasure to protect the execution flow was developed: the security automatons.

Execution of a malicious code



- Published in [Bouffard et al., SAFECOMP 2013];
- For this presentation, two vulnerabilities will be introduced:
  - Modifying the method's return address;
  - Corrupting the `finally`-clause.
- Thanks to **minimal cut set**, a countermeasure to protect the execution flow was developed: **the security automatons**.

The Java Method Return

> *The current frame is used in this case to* ***restore the state of the invoker****, including its local variables and operand stack, with the* ***program counter of the invoker*** *appropriately incremented to skip past the method invocation instruction. Execution then continues normally in the invoking method's frame with the returned value (if any) pushed onto the operand stack of that frame.* (source: Java 8 Virtual Machine Specification)

- A frame header may include:
  - Previous frame's size;
  - Program counter of the invoker;
  - Security context of the invoker.

Java Card Stack

```
public void caller (short l1) {
  // The function callee is called
  short l2 = l1 +
             this.callee(l1);
}
```

```
void caller (short l1) {
  sload  1
  aload  0
  sload  1
  invokevirtual @callee
  sadd
  sstore 2
  return
}
```

```
public void callee (short l1) {
  short l2 = l1;
  short l3 = (short) 0xCAFE;
  return l3;
}
```

```
void callee (short l1) {
  sload  1
  sstore 2
  sspush 0xCAFE
  sstore 3
  sload  3
  sreturn
}
```

Java code

Java Card byte code

Java Card Stack

```java
public void caller (short l1) {
  // The function callee is called
  short l2 = l1 +
             this.callee(l1);
}
```

```
void caller (short l1) {
  sload  1
  aload  0
  sload  1
  invokevirtual @callee
  sadd
  sstore 2
  return
}
```

```java
public void callee (short l1) {
  short l2 = l1;
  short l3 = (short) 0xCAFE;
  return l3;
}
```

```
void callee (short l1) {
  sload  1
  sstore 2
  sspush 0xCAFE
  sstore 3
  sload  3
  sreturn
}
```

Java code

Java Card byte code

Java Card Stack

```
public void caller (short l1) {
  // The function callee is called
  short l2 = l1 +
             this.callee(l1);
}
```

```
void caller (short l1) {
  sload   1
  aload   0
  sload   1
  invokevirtual @callee
  sadd
  sstore 2
  return
}
```

```
public void callee (short l1) {
  short l2 = l1;
  short l3 = (short) 0xCAFE;
  return l3;
}
```

```
void callee (short l1) {
  sload   1
  sstore 2
  sspush 0xCAFE
  sstore 3
  sload   3
  sreturn
}
```

Java code

Java Card byte code

Java Card Stack

```java
public void caller (short l1) {
  // The function callee is called
  short l2 = l1 +
             this.callee(l1);
}
```

```
void caller (short l1) {
  sload   1
  aload   0
  sload   1
  invokevirtual @callee
  sadd
  sstore 2
  return
}
```

```java
public void callee (short l1) {
  short l2 = l1;
  short l3 = (short) 0xCAFE;
  return l3;
}
```

```
void callee (short l1) {
  sload   1
  sstore 2
  sspush 0xCAFE
  sstore 3
  sload   3
  sreturn
}
```

Java code

Java Card byte code

Java Card Stack

```java
public void caller (short l1) {
  // The function callee is called
  short l2 = l1 +
             this.callee(l1);
}
```

```
void caller (short l1) {
  sload  1
  aload  0
  sload  1
  invokevirtual @callee
  sadd
  sstore 2
  return
}
```

```java
public void callee (short l1) {
  short l2 = l1;
  short l3 = (short) 0xCAFE;
  return l3;
}
```

```
void callee (short l1) {
  sload  1
  sstore 2
  sspush 0xCAFE
  sstore 3
  sload  3
  sreturn
}
```

Java code

Java Card byte code

Java Card Stack

```java
public void caller (short l1) {
  // The function callee is called
  short l2 = l1 +
             this.callee(l1);
}
```

```
void caller (short l1) {
  sload   1
  aload   0
  sload   1
  invokevirtual @callee
  sadd
  sstore 2
  return
}
```

```java
public void callee (short l1) {
  short l2 = l1;
  short l3 = (short) 0xCAFE;
  return l3;
}
```

```
void callee (short l1) {
  sload   1
  sstore 2
  sspush 0xCAFE
  sstore 3
  sload   3
  sreturn
}
```

Java code

Java Card byte code

Java Card Stack

```java
public void caller (short l1) {
  // The function callee is called
  short l2 = l1 + 
             this.callee(l1);
}
```

```
void caller (short l1) {
  sload   1
  aload   0
  sload   1
  invokevirtual @callee
  sadd
  sstore 2
  return
}
```

```java
public void callee (short l1) {
  short l2 = l1;
  short l3 = (short) 0xCAFE;
  return l3;
}
```

```
void callee (short l1) {
  sload   1
  sstore 2
  sspush 0xCAFE
  sstore 3
  sload   3
  sreturn
}
```

Java code

Java Card byte code

Java Card Stack

```
public void caller (short l1) {
  // The function callee is called
  short l2 = l1 +
             this.callee(l1);
}
```

```
void caller (short l1) {
  sload  1
  aload  0
  sload  1
  invokevirtual @callee
  sadd
  sstore 2
  return
}
```

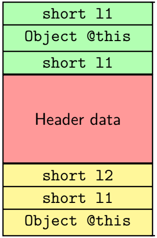
```
public void callee (short l1) {
  short l2 = l1;
  short l3 = (short) 0xCAFE;
  return l3;
}
```

```
void callee (short l1) {
  sload  1
  sstore 2
  sspush 0xCAFE
  sstore 3
  sload  3
  sreturn
}
```

Java code

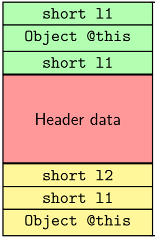
Java Card byte code

Java Card Stack: Pushing a Frame

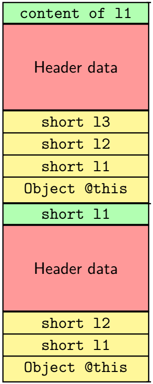



```
sload 1 // Pushing l1
aload 0 // Pushing @this
sload 1 // Pushing l1
invokevirtual @callee
sadd
sstore 2 // Saving to l2
return
```

Java Card Stack: Pushing a Frame



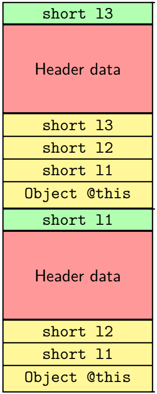

```
sload 1 // Pushing l1
aload 0 // Pushing @this
sload 1 // Pushing l1
invokevirtual @callee
sadd
sstore 2 // Saving to l2
return
```




```
sload 1 // Pushing l1
sstore 2 // Storing to l2
sspush 0xCAFE
sstore 3 // Storing to l3
sload 3 // pushing l3
sreturn
```

```
sload 1 // Pushing l1
aload 0 // Pushing @this
sload 1 // Pushing l1
invokevirtual @callee
sadd
sstore 2 // Saving to l2
return
```

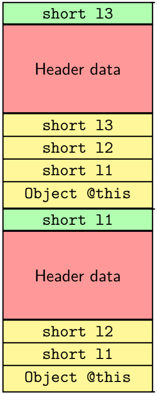
Java Card Stack: Popping a Frame




```
sload  1 // Pushing l1
sstore 2 // Storing to l2
sspush 0xCAFE
sstore 3 // Storing to l3
sload  3 // pushing l3
sreturn
```

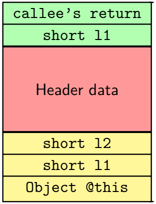
```
sload 1 // Pushing l1
aload 0 // Pushing @this
sload 1 // Pushing l1
invokevirtual @callee
sadd
sstore 2 // Saving to l2
return
```

Java Card Stack: Popping a Frame




```
sload  1 // Pushing l1
sstore 2 // Storing to l2
sspush 0xCAFE
sstore 3 // Storing to l3
sload  3 // pushing l3
sreturn
```

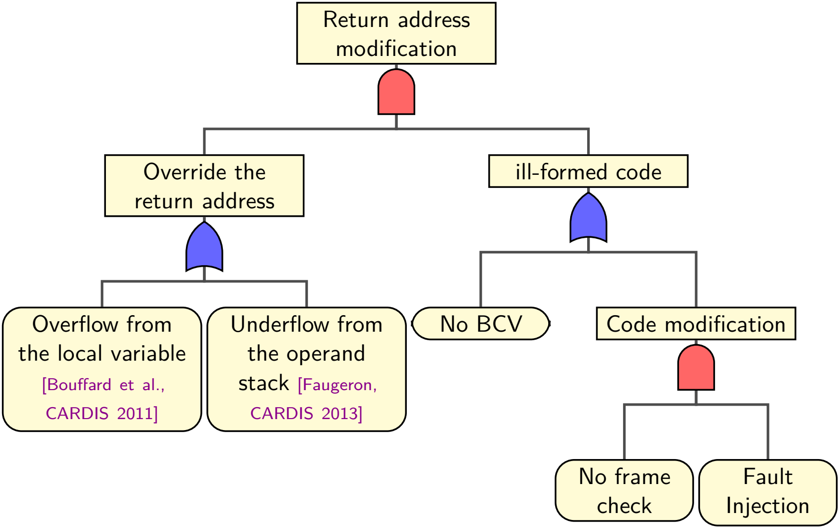
```
sload 1 // Pushing l1
aload 0 // Pushing @this
sload 1 // Pushing l1
invokevirtual @callee
sadd
sstore 2 // Saving to l2
return
```




```
sload 1 // Pushing l1
aload 0 // Pushing @this
sload 1 // Pushing l1
invokevirtual @callee
sadd
sstore 2 // Saving to l2
return
```

EMAN2: *A Ghost In the Stack*

- Modifying the return address;



EMAN2: *A Ghost In the Stack*

- Presented in [Bouffard et al., CARDIS 2011];
- **Overflow** from the local variables area.

Stack
short l1
Header data
short l3
short l2
short l1
Object @this

```
sload 1 // Pushing l1
sstore 2 // Saving to l2
sspush 0xCAFE
sstore 3 // Saving to l3
sload 3 // Pushing l3
sreturn
```

Stack
short l1
Header data
short l2
short l1
Object @this

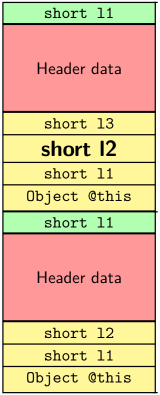
```
sload 1 // Pushing l1
aload 0 // Pushing @this
sload 1 // Pushing l1
invokevirtual @callee
sadd
sstore 2 // Saving to l2
return
```

Stack
callee's return
short l1
Header data
short l2
short l1
Object @this

```
sload 1 // Pushing l1
aload 0 // Pushing @this
sload 1 // Pushing l1
invokevirtual @callee
sadd
sstore 2 // Saving to l2
return
```

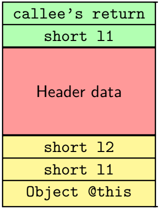
EMAN2: *A Ghost In the Stack*

- Presented in [Bouffard et al., CARDIS 2011];
- **Overflow** from the local variables area.




```
sload 1 // Pushing l1
sstore 2 // Saving to l2
sspush 0xCAFE
sstore 3 // Saving to l3
sload 3 // Pushing l3
sreturn
```

```
sload 1 // Pushing l1
aload 0 // Pushing @this
sload 1 // Pushing l1
invokevirtual @callee
sadd
sstore 2 // Saving to l2
return
```




```
sload 1 // Pushing l1
aload 0 // Pushing @this
sload 1 // Pushing l1
invokevirtual @callee
sadd
sstore 2 // Saving to l2
return
```

EMAN2: *A Ghost In the Stack*

- Presented in [Bouffard et al., CARDIS 2011];
- **Overflow** from the local variables area.

short l1
Header data
short l3
short l2
short l1
Object @this

```
sload 1 // Pushing l1
sstore 4 // Saving to l4
sspush 0xCAFE
sstore 3 // Saving to l3
sload 3 // Pushing l3
sreturn
```

short l1
Header data
short l2
short l1
Object @this

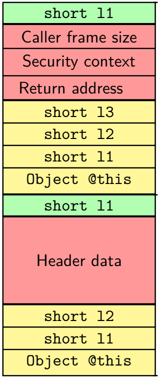
```
sload 1 // Pushing l1
aload 0 // Pushing @this
sload 1 // Pushing l1
invokevirtual @callee
sadd
sstore 2 // Saving to l2
return
```

callee's return
short l1
Header data
short l2
short l1
Object @this

```
sload 1 // Pushing l1
aload 0 // Pushing @this
sload 1 // Pushing l1
invokevirtual @callee
sadd
sstore 2 // Saving to l2
return
```

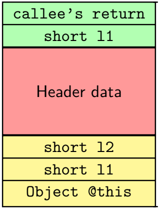
EMAN2: *A Ghost In the Stack*

- Presented in [Bouffard et al., CARDIS 2011];
- **Overflow** from the local variables area.




```
sload 1 // Pushing l1
sstore 4 // Saving to l4
sspush 0xCAFE
sstore 3 // Saving to l3
sload 3 // Pushing l3
sreturn
```

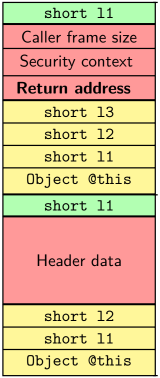
```
sload 1 // Pushing l1
aload 0 // Pushing @this
sload 1 // Pushing l1
invokevirtual @callee
sadd
sstore 2 // Saving to l2
return
```




```
sload 1 // Pushing l1
aload 0 // Pushing @this
sload 1 // Pushing l1
invokevirtual @callee
sadd
sstore 2 // Saving to l2
return
```

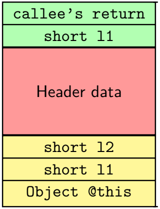
EMAN2: *A Ghost In the Stack*

- Presented in [Bouffard et al., CARDIS 2011];
- **Overflow** from the local variables area.




```
sload 1 // Pushing l1
sstore 4 // Saving to l4
sspush 0xCAFE
sstore 3 // Saving to l3
sload 3 // Pushing l3
sreturn
```

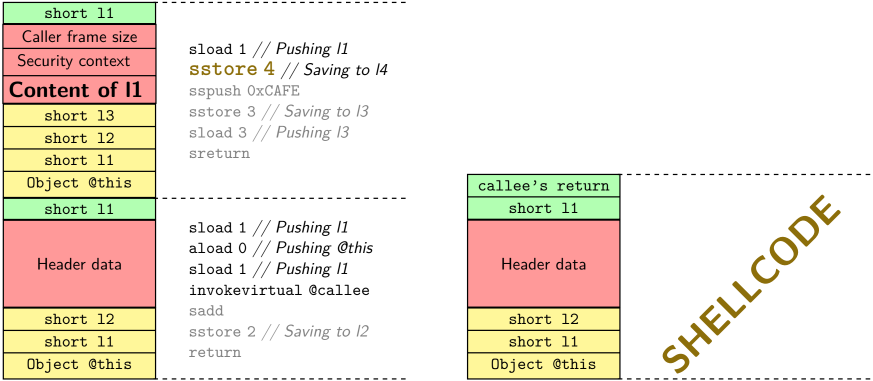
```
sload 1 // Pushing l1
aload 0 // Pushing @this
sload 1 // Pushing l1
invokevirtual @callee
sadd
sstore 2 // Saving to l2
return
```




```
sload 1 // Pushing l1
aload 0 // Pushing @this
sload 1 // Pushing l1
invokevirtual @callee
sadd
sstore 2 // Saving to l2
return
```

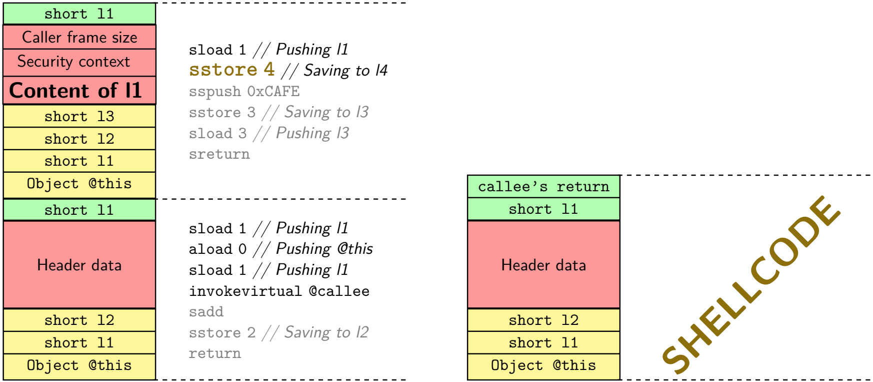
EMAN2: *A Ghost In the Stack*

- Presented in [Bouffard et al., CARDIS 2011];
- **Overflow** from the local variables area.



EMAN2: *A Ghost In the Stack*

- Presented in [Bouffard et al., CARDIS 2011];
- **Overflow** from the local variables area.



- Countermeasures from the literature:
  - Checking the integrity of the frame's header data;
  - Verifying each access to the frame's areas [Lackner et al., CARDIS 2012];
  - Scrambling the memory [Barbu's PhD Thesis, 2012] [Razafindralambo et al., SNDS 2012].

EMAN2 and Its Avatars

- Stack **overflow** from the local variables [Bouffard et al., CARDIS 2011]
  - `sstore`, `sinc`, etc.;
- Stack **underflow** from the operand stack [Faugeron, CARDIS 2013]
  - `dup_x`, `swap_x`, etc.;

EMAN2 and Its Avatars

- Stack **overflow** from the local variables [Bouffard et al., CARDIS 2011]
  - `sstore`, `sinc`, etc.;
- Stack **underflow** from the operand stack [Faugeron, CARDIS 2013]
  - `dup_x`, `swap_x`, etc.;
- This attack modifies the Java Program Counter value upon the return address register. New smart cards embed countermeasures against this attack! . . . **only the path is protected**;

The `finally`-Clause

- A `finally`-statement used the `jsr` (“***jump to subroutine***”) and `ret` (“***return from subroutine***”) instructions (deprecated since Java 6) ;
- The `jsr` pushes the address of the instruction immediately following it (typed as **`ReturnAddress`**);
- Saves the return value (if any) in a local variable;
- The `ret` instruction continues the execution from the value saved in the local variable.

Compiling `finally`-Clause

```java
void tryCatchFinally() {
  try {
    tryItOut();
  } finally {
    wrapItUp();
  }
}
```

Exception table:

From	To	Target	Type
0	4	8	any

```
Method void tryFinally()
0    aload_0          // Beginning of try block
1    invokevirtual tryItOut()
4    jsr 14           // Call finally block
7    return           // End of try block

8    astore_1         // Beginning of handler
                      // for any throw
9    jsr 14           // Call finally block
12   aload_1          // Push thrown value
13   athrow           // ... and rethrow value
                      // to the invoker

14   astore_2         // Beginning of finally block
15   aload_0          // Push this
16   invokevirtual wrapItUp()
19   ret 2            // Return from finally block
```

Illustration inspired from the Java 8 Virtual Machine Specification

Compiling `finally`-Clause

```
void tryCatchFinally() {
  try {
    tryItOut();
  } finally {
    wrapItUp();
  }
}
```

Exception table:

From	To	Target	Type
0	4	8	any

```
Method void tryFinally()
0    aload_0          // Beginning of try block
1    invokevirtual tryItOut()
4    jsr 14           // Call finally block
7    return           // End of try block

8    astore_1         // Beginning of handler
                      // for any throw
9    jsr 14           // Call finally block
12   aload_1          // Push thrown value
13   athrow           // ... and rethrow value
                      // to the invoker

14   astore_2         // Beginning of finally block
15   aload_0          // Push this
16   invokevirtual wrapItUp()
19   ret 2            // Return from finally block
```

Illustration inspired from the Java 8 Virtual Machine Specification

Compiling `finally`-Clause

```
void tryCatchFinally() {
  try {
    tryItOut();
  } finally {
    wrapItUp();
  }
}
```

Exception table:

From	To	Target	Type
0	4	8	any

```
Method void tryFinally()
0   aload_0           // Beginning of try block
1   invokevirtual tryItOut()
4   jsr 14            // Call finally block
7   return            // End of try block

8   astore_1          // Beginning of handler
                      // for any throw
9   jsr 14            // Call finally block
12  aload_1           // Push thrown value
13  athrow            // ... and rethrow value
                      // to the invoker

14  astore_2          // Beginning of finally block
15  aload_0           // Push this
16  invokevirtual wrapItUp()
19  ret 2             // Return from finally block
```

Illustration inspired from the Java 8 Virtual Machine Specification

Compiling `finally`-Clause

```java
void tryCatchFinally() {
  try {
    tryItOut();
  } finally {
    wrapItUp();
  }
}
```

Exception table:			
From	To	Target	Type
0	4	8	any

```
Method void tryFinally()
0    aload_0            // Beginning of try block
1    invokevirtual tryItOut()
4    jsr 14             // Call finally block
7    return             // End of try block

8    astore_1           // Beginning of handler
                        // for any throw
9    jsr 14             // Call finally block
12   aload_1            // Push thrown value
13   athrow             // ... and rethrow value
                        // to the invoker

14   astore_2           // Beginning of finally block
15   aload_0            // Push this
16   invokevirtual wrapItUp()
19   ret 2              // Return from finally block
```

Illustration inspired from the Java 8 Virtual Machine Specification

Compiling `finally`-Clause

```
void tryCatchFinally() {
  try {
    tryItOut();
  } finally {
    wrapItUp();
  }
}
```

Exception table:			
From	To	Target	Type
0	4	8	any

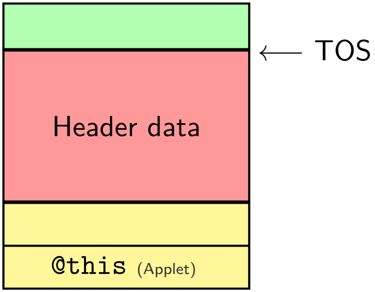
```
Method void tryFinally()
0   aload_0          // Beginning of try block
1   invokevirtual tryItOut()
4   jsr 14           // Call finally block
7   return           // End of try block

8   astore_1         // Beginning of handler
                     // for any throw
9   jsr 14           // Call finally block
12  aload_1          // Push thrown value
13  athrow           // ... and rethrow value
                     // to the invoker

14  astore_2         // Beginning of finally block
15  aload_0          // Push this
16  invokevirtual wrapItUp()
19  ret 2            // Return from finally block
```

Illustration inspired from the Java 8 Virtual Machine Specification

Executing a `finally`-Clause

```
 method_info [2] // @0051 = {
   01 // flags: 0 max_stack : 1
   11 // nargs: 1 max_locals: 1
⇒ /*0x53*/ L0: jsr L2
   /*0x56*/ L1: sspush  0xCAFE
   /*0x59*/     sreturn
   /*0x5A*/ L2: astore_1
   /*0x5B*/     ret 0x1 // -> L1
 }
```

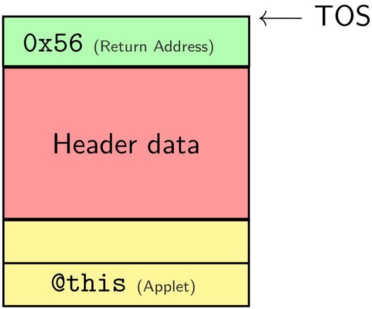
$\boxed{\mathrm{PC} = \mathrm{0x53}}$



Executing a finally-Clause

```
method_info [2] // @0051 = {
  01 // flags: 0 max_stack : 1
  11 // nargs: 1 max_locals: 1
  /*0x53*/ L0: jsr L2
  /*0x56*/ L1: sspush  0xCAFE
  /*0x59*/     sreturn
⇒ /*0x5A*/ L2: astore_1
  /*0x5B*/     ret 0x1 // -> L1
}
```

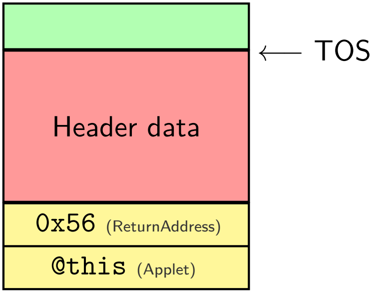
$\boxed{\text{PC} = \text{0x5A}}$



Executing a `finally`-Clause

```
  method_info [2] // @0051 = {
    01 // flags: 0 max_stack : 1
    11 // nargs: 1 max_locals: 1
    /*0x53*/ L0: jsr L2
    /*0x56*/ L1: sspush  0xCAFE
    /*0x59*/      sreturn
    /*0x5A*/ L2: astore_1
⇒   /*0x5B*/      ret 0x1 // -> L1
  }
```

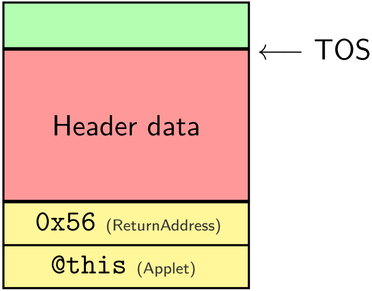
$$\boxed{\mathrm{PC} = \mathrm{0x5B}}$$



Executing a `finally`-Clause

```
method_info [2] // @0051 = {
  01 // flags: 0 max_stack : 1
  11 // nargs: 1 max_locals: 1
  /*0x53*/ L0: jsr L2
⇒ /*0x56*/ L1: sspush  0xCAFE
  /*0x59*/     sreturn
  /*0x5A*/ L2: astore_1
  /*0x5B*/     ret 0x1 // -> L1
}
```

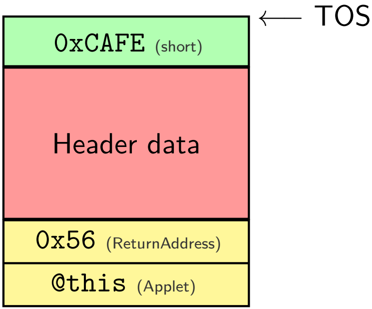
$\boxed{\mathrm{PC} = 0x56}$



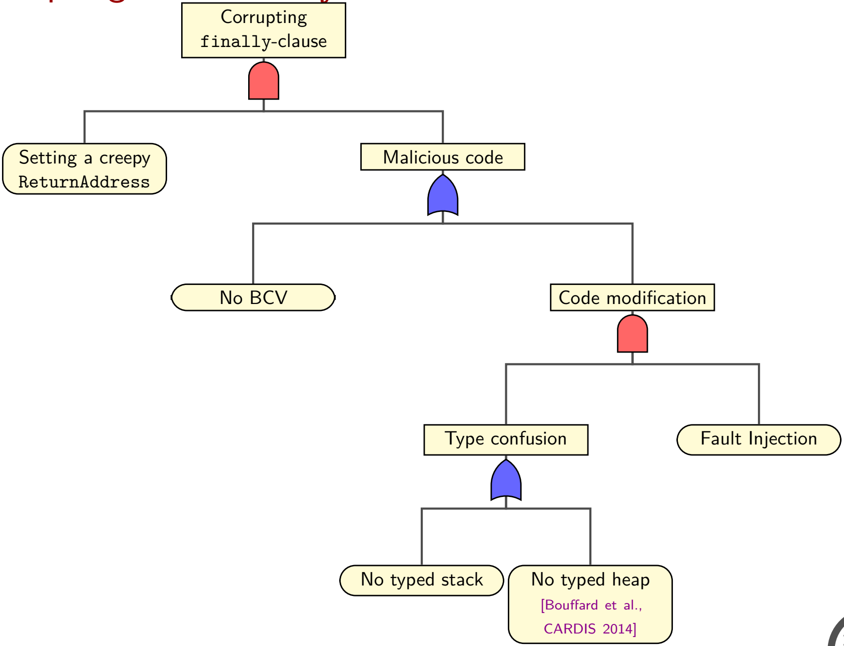
Executing a finally-Clause

```
method_info [2] // @0051 = {
  01 // flags: 0 max_stack : 1
  11 // nargs: 1 max_locals: 1
  /*0x53*/ L0: jsr L2
  /*0x56*/ L1: sspush  0xCAFE
⇒ /*0x59*/     sreturn
  /*0x5A*/ L2: astore_1
  /*0x5B*/     ret 0x1 // -> L1
}
```

$\boxed{\text{PC} = \text{0x59}}$



Corrupting the `finally`-Clause



How to Exploit the jsr instruction?

- Hypothesis:
  - No verified by a BCV
  - No typed stack

```
short jsrAttack () {
  01 // flags: 0 max_stack : 1
  11 // nargs: 1 max_locals: 1
  /*0x53*/ L0: jsr L2
  /*0x56*/ L1: sspush 0xCAFE
  /*0x59*/     sreturn
  /*0x5A*/     sspush 0xBEEF
  /*0x5D*/     sreturn
  /*0x5E*/ L2: astore_1
  /*0x5F*/     sinc 0x1, 0x4
  /*0x62*/     ret 1 // -> L1
}
```

How to Exploit the jsr instruction?

- Hypothesis:
  - No verified by a BCV
  - No typed stack

```
short jsrAttack () {
  01 // flags: 0 max_stack : 1
  11 // nargs: 1 max_locals: 1
  /*0x53*/ L0: jsr L2
  /*0x56*/ L1: sspush 0xCAFE
  /*0x59*/     sreturn
  /*0x5A*/     sspush 0xBEEF
  /*0x5D*/     sreturn
  /*0x5E*/ L2: astore_1
  /*0x5F*/     sinc 0x1, 0x4   ←—— Type confusion
  /*0x62*/     ret 1 // -> L1
}
```

Cheating the BCV component

- The BCV checks the structure and the semantics of the application;
- To verify the byte code semantics, the BCV starts its analyse from an **entry point**;
- Unreachable code has no entry point $\Rightarrow$ ⚠ it is **not checked** by the BCV!
- A malicious byte code can be hidden through the BCV verification!

An Unreachable Code...

```
void cheatingBCV () {
  04 // flags: 0 max_stack : 4
  03 // nargs: 0 max_locals: 3
  /*0x05B*/ L0: jsr L1
  // ...
  /*0x066*/ L1: astore_3
            L2: ... // Set of instructions
  /*0x163*/     if_scmpeq_w 0xFF05 // -> L2
  /*0x166*/     return
  /*0x167*/     sinc 0x3, 0x4
  /*0x16A*/     ret 0x3
}
```

An Unreachable Code. . .

```
void cheatingBCV () {
  04 // flags: 0 max_stack : 4
  03 // nargs: 0 max_locals: 3
  /*0x05B*/ L0: jsr L1
  // ...
  /*0x066*/ L1: astore_3
              L2: ... // Set of instructions
  /*0x163*/       if_scmpeq_w 0xFF05 // -> L2
  /*0x166*/       return
  /*0x167*/       sinc 0x3, 0x4
  /*0x16A*/       ret 0x3
}
```

Checked by the BCV

Unchecked by the BCV

An Unreachable Code...

```
void cheatingBCV () {
  04 // flags: 0 max_stack : 4
  03 // nargs: 0 max_locals: 3
  /*0x05B*/ L0: jsr L1
  // ...
  /*0x066*/ L1: astore_3
            L2: ... // Set of instructions
  /*0x163*/     if_scmpeq_w 0xFF05 // -> L2
  /*0x166*/     return
  /*0x167*/     sinc 0x3, 0x4
  /*0x16A*/     ret 0x3
}
```

Checked by the BCV

Unchecked by the BCV

```
verifycap api_export_files/**/*.exp maliciousCAPFile.cap
[ INFO: ] Verifier [v3.0.4]
[ INFO: ]  Copyright (c) 2011, Oracle and/or its affiliates.
           All rights reserved.

[ INFO: ] Verifying CAP file maliciousCAPFile.cap
[ INFO: ] Verification completed with 0 warnings and 0 errors.
```

. . . Can Be Executed

- EMAN4 [Bouffard et al., CARDIS 2011] introduced a way to change an instruction's parameter upon a laser beam injection;
  - This attack focuses on wide instructions;
  - `goto_w`, `if_*_w`, . . .

… Can Be Executed

- EMAN4 [Bouffard et al., CARDIS 2011] introduced a way to change an instruction's parameter upon a laser beam injection;
  - This attack focuses on wide instructions;
  - `goto_w`, `if_*_w`, …
- `if_scmpeq_w 0xFF05`

. . . Can Be Executed

- EMAN4 [Bouffard et al., CARDIS 2011] introduced a way to change an instruction's parameter upon a laser beam injection;
  - This attack focuses on wide instructions;
  - `goto_w`, `if_*_w`, . . .
- `if_scmpeq_w 0xFF05` ⇒ `if_scmpeq_w 0x0005`

… Can Be Executed

- EMAN4 [Bouffard et al., CARDIS 2011] introduced a way to change an instruction's parameter upon a laser beam injection;
  - This attack focuses on wide instructions;
  - `goto_w`, `if_*_w`, …
- `if_scmpeq_w 0xFF05` ⇒ `if_scmpeq_w 0x0005`
- That can be viewed as a logical attack enabler.

An Unreachable Code… Becomes Reachable

```
void cheatingBCV () {
  04 // flags: 0 max_stack : 4
  03 // nargs: 0 max_locals: 3
  /*0x85B*/ L0: jsr L1
  // ...
  /*0x866*/ L1: astore_3
            L2: ... // Set of instructions
  /*0x963*/     if_scmpeq_w 0x0005 // -> L3
  /*0x966*/     return
  /*0x967*/ L3: sinc 0x3, 0x4
  /*0x96A*/     ret 0x3
}
```

Preventing any `finally`-clause Corruption

- The Java 8 Virtual Machine specification defines basic ideas:
  - Each instruction **keeps track** of the list of `jsr` targets needed to reach that instruction.
  - When executing the `ret` instruction, there must be only **one possible subroutine** from which the instruction can be returning.

Preventing any `finally`-clause Corruption

- The Java 8 Virtual Machine specification defines basic ideas:
  - Each instruction **keeps track** of the list of `jsr` targets needed to reach that instruction.
  - When executing the `ret` instruction, there must be only **one possible subroutine** from which the instruction can be returning.
- How to include that in the JCVM?

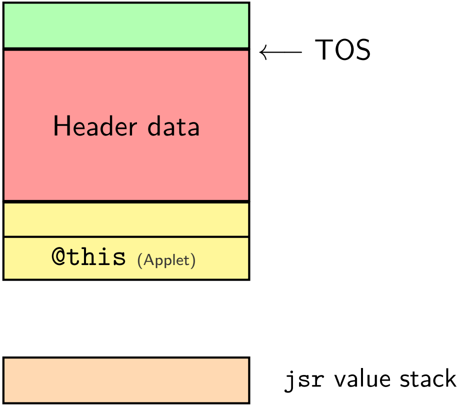
Preventing any `finally`-clause Corruption

- The Java 8 Virtual Machine specification defines basic ideas:
  - Each instruction **keeps track** of the list of `jsr` targets needed to reach that instruction.
  - When executing the `ret` instruction, there must be only **one possible subroutine** from which the instruction can be returning.
- How to include that in the JCVM?
- **Solution**: a `jsr` value stack.

Preventing any finally-clause Corruption (Cont.)

```
short jsrAttack () {
  01 // flags: 0 max_stack : 1
  11 // nargs: 1 max_locals: 1
⇒ /*0x53*/ L0: jsr L2
  /*0x56*/ L1: sspush 0xCAFE
  /*0x59*/     sreturn
  /*0x5A*/     sspush 0xBEEF
  /*0x5D*/     sreturn
  /*0x5E*/ L2: astore_1
  /*0x5F*/     sinc 0x1, 0x4
  /*0x62*/     ret 1 // -> L1
}
```

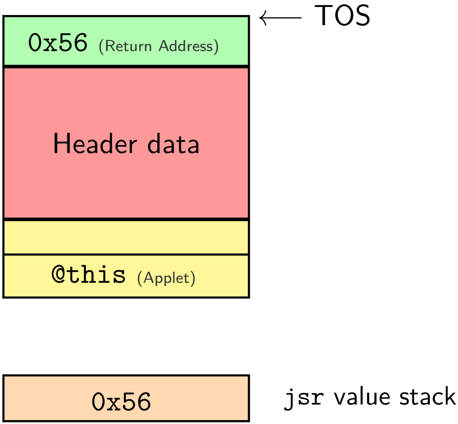
$\boxed{\text{PC} = \text{0x53}}$



Preventing any `finally`-clause Corruption (Cont.)

```
short jsrAttack () {
   01 // flags: 0 max_stack : 1
   11 // nargs: 1 max_locals: 1
   /*0x53*/ L0: jsr L2
   /*0x56*/ L1: sspush 0xCAFE
   /*0x59*/     sreturn
   /*0x5A*/     sspush 0xBEEF
   /*0x5D*/     sreturn
⇒ /*0x5E*/ L2: astore_1
   /*0x5F*/     sinc 0x1, 0x4
   /*0x62*/     ret 1 // -> L1
}
```

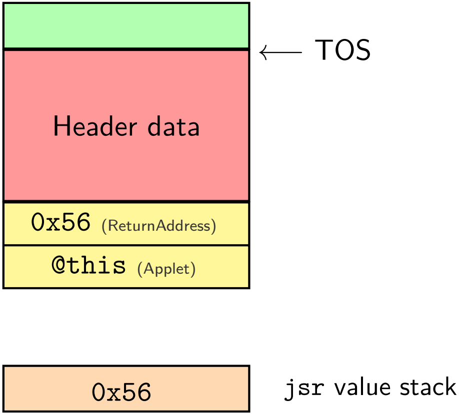
$\boxed{\mathrm{PC} = \mathrm{0x5E}}$



Preventing any finally-clause Corruption (Cont.)

```
short jsrAttack () {
  01 // flags: 0 max_stack : 1
  11 // nargs: 1 max_locals: 1
  /*0x53*/ L0: jsr L2
  /*0x56*/ L1: sspush 0xCAFE
  /*0x59*/     sreturn
  /*0x5A*/     sspush 0xBEEF
  /*0x5D*/     sreturn
  /*0x5E*/ L2: astore_1
⇒ /*0x5F*/     sinc 0x1, 0x4
  /*0x62*/     ret 1 // -> L1
}
```

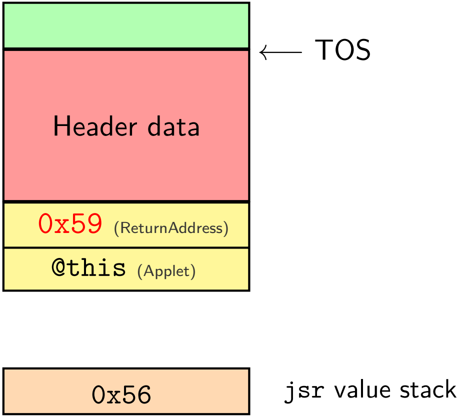
$\boxed{\mathrm{PC} = 0\mathrm{x}5\mathrm{F}}$



Preventing any `finally`-clause Corruption (Cont.)

```
short jsrAttack () {
  01 // flags: 0 max_stack : 1
  11 // nargs: 1 max_locals: 1
  /*0x53*/ L0: jsr L2
  /*0x56*/ L1: sspush 0xCAFE
  /*0x59*/     sreturn
  /*0x5A*/     sspush 0xBEEF
  /*0x5D*/     sreturn
  /*0x5E*/ L2: astore_1
  /*0x5F*/     sinc 0x1, 0x4
⇒ /*0x62*/     ret 1 // -> L1
}
```

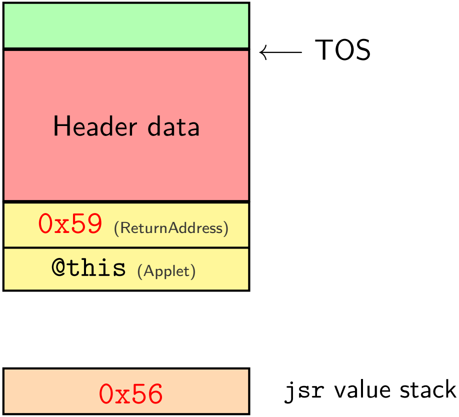
$\boxed{\mathrm{PC} = \mathrm{0x62}}$



Preventing any finally-clause Corruption (Cont.)

```
short jsrAttack () {
  01 // flags: 0 max_stack : 1
  11 // nargs: 1 max_locals: 1
  /*0x53*/ L0: jsr L2
  /*0x56*/ L1: sspush 0xCAFE
  /*0x59*/     sreturn
  /*0x5A*/     sspush 0xBEEF
  /*0x5D*/     sreturn
  /*0x5E*/ L2: astore_1
  /*0x5F*/     sinc 0x1, 0x4
⇒ /*0x62*/     ret 1 // -> L1
}
```

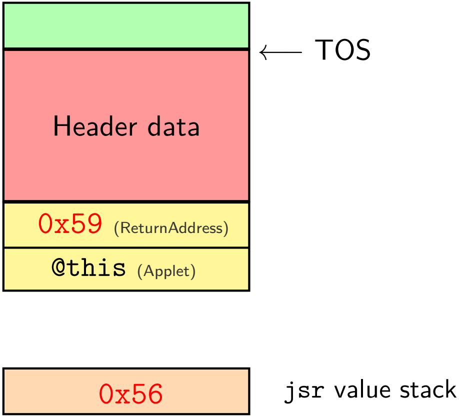
$\boxed{\mathrm{PC} = 0\mathrm{x}62}$



Preventing any finally-clause Corruption (Cont.)

```
short jsrAttack () {
  01 // flags: 0 max_stack : 1
  11 // nargs: 1 max_locals: 1
  /*0x53*/ L0: jsr L2
  /*0x56*/ L1: sspush 0xCAFE
  /*0x59*/     sreturn
  /*0x5A*/     sspush 0xBEEF
  /*0x5D*/     sreturn
  /*0x5E*/ L2: astore_1
  /*0x5F*/     sinc 0x1, 0x4
⇒ /*0x62*/     ret 1 // -> L1
}
```

$\boxed{\text{PC} = \text{0x62}}$



Attack detected!

How to Protect the Execution Flow?

- Presented attacks:
  - EMAN2: cheating the return address;
  - `finally`-clause corruption: direct modification of the program counter;
- Each of them sets up the Java program counter;
- How to ensure the **execution flow**?

Protecting the Execution Flow

- Direct modification:
  - Integrity $\rightarrow$ can be bypassed when the JPC is updated by the JCVM;
- Transient fault:
  - Executing twice the same piece of code;
  - It is a very expensive solution;
- Solution: dynamically check the applet's CFG:
  - Séré's countermeasures [Séré's PhD thesis, 2010] based on Field of bits, Basic block method or Path check technique;
  - This kind of countermeasure can be **computed in the card**?

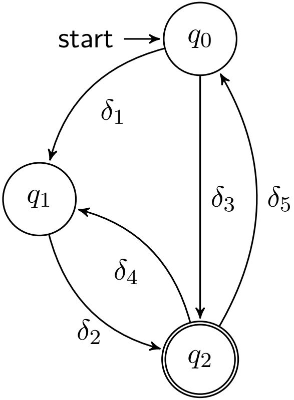
Security Automatons and Execution Monitor

Principle

- Detecting a deviant behaviour ⇒ **safety property** "*nothing bad happens*";
- Preventing some attacks: several partial traces of events are defined:
  - Property can be encoded by a finite state automaton;
- Schneider automatons: (Q, q_0, δ), where Q is a set of states, q_o is the initial state and δ is a transition function $(Q \cdot I) \to 2^Q)$;
- The CFG can be computed during the loading process;
- When interpreting a byte code, the monitor checks:
  - If the transition generates an authorized partial trace;
  - If not, it takes an appropriate countermeasure.

Security Automatons and Execution Monitor (Cont.)

Principle



State	q_0	q_1	q_2
q_0		δ_1	δ_3
q_1			δ_2
q_2	δ_5	δ_4	

Security automaton (computed inside the card)

State matrix (binary implementation of the security automaton)

- [Bouffard et al., SSCC 2013], [Bouffard et al., SAR-SSI 2013] and extended in [Bouffard et al., IJTMCC 2014].

Security Automaton in Practice

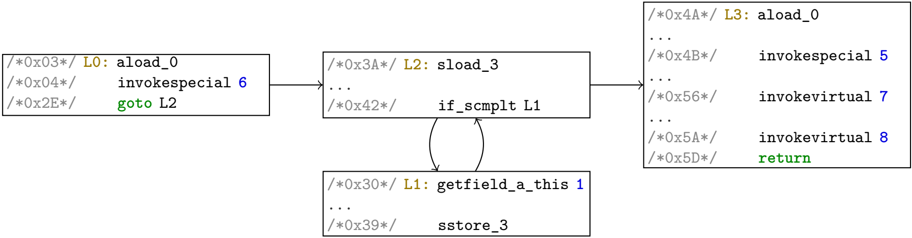
```java
protected Protocolpayment (byte[] buffer, short offset, byte length) {
  A[0] = 0; // initialisation of array A
  for (byte j = 0; j < buffer[(byte)(offset+12)]; j++) {
    D[j] = 0; // initialisation of array D
  }
  pin = new OwnerPIN((byte) TRY_LIMIT, (byte) MAX_PIN_SIZE);
  // Initialisation of pin
  pin.update(myPin, (short) START_OFFSET, (byte) myPin.length);
  register(); // registering this instance
} // source: (Girard et al., CRiSIS 2010)
```

Security Automaton in Practice

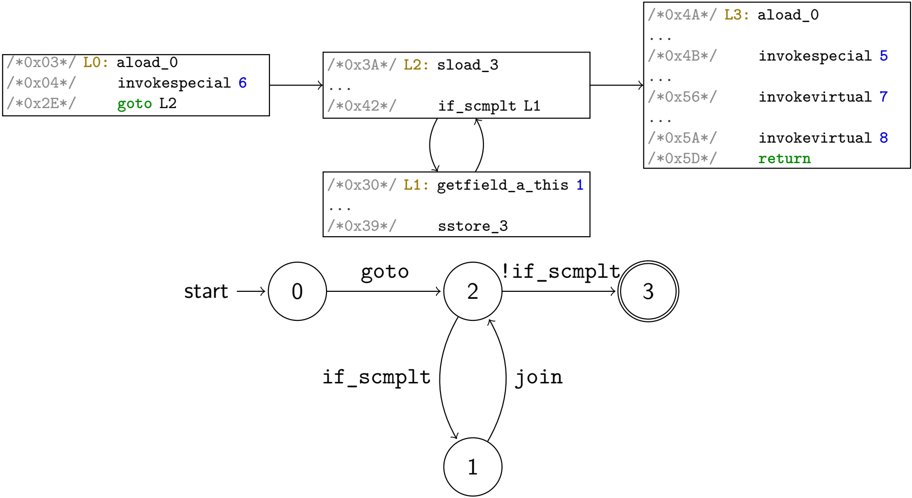
```java
protected Protocolpayment (byte[] buffer, short offset, byte length) {
  A[0] = 0; // initialisation of array A
  for (byte j = 0; j < buffer[(byte)(offset+12)]; j++) {
    D[j] = 0; // initialisation of array D
  }
  pin = new OwnerPIN((byte) TRY_LIMIT, (byte) MAX_PIN_SIZE);
  // Initialisation of pin
  pin.update(myPin, (short) START_OFFSET, (byte) myPin.length);
  register(); // registering this instance
} // source: (Girard et al., CRiSIS 2010)
```

- To create the security automaton:
  - Local view of the method's CFG;
  - The set S contains the element of a language which expresses the control flow integrity policy:
    - `ifeq`, `ifne`, `goto`, `invoke`, `return`, etc.;
    - plus the dummy instruction `join` representing any other instruction pointed by a label.

Security Automaton included in the JCVM



Security Automaton included in the JCVM



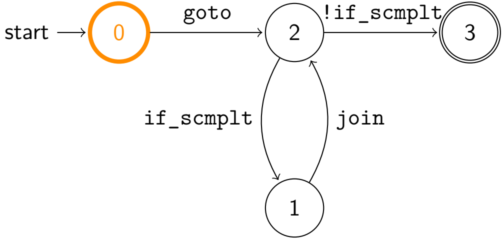
Security Automaton included in the JCVM

```
/*0x03*/ L0: aload_0
/*0x04*/     invokespecial 6
/*0x2E*/     goto L2
```

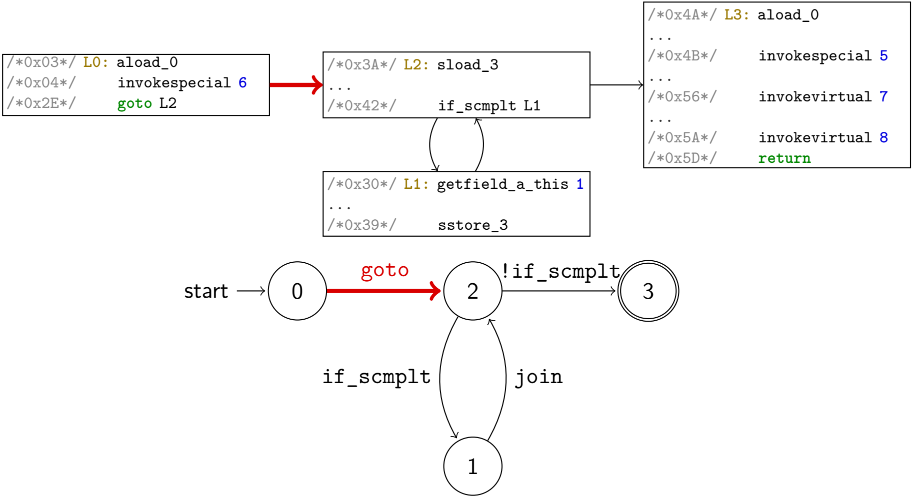
```
/*0x3A*/ L2: sload_3
...
/*0x42*/     if_scmplt L1
```

```
/*0x30*/ L1: getfield_a_this 1
...
/*0x39*/     sstore_3
```

```
/*0x4A*/ L3: aload_0
...
/*0x4B*/     invokespecial 5
...
/*0x56*/     invokevirtual 7
...
/*0x5A*/     invokevirtual 8
/*0x5D*/     return
```



Security Automaton included in the JCVM



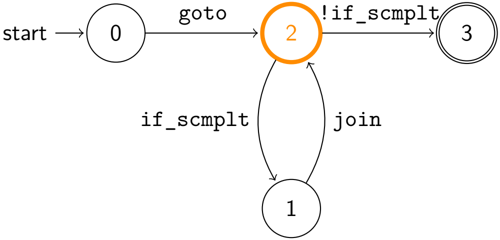
Security Automaton included in the JCVM

```
/*0x03*/ L0: aload_0
/*0x04*/     invokespecial 6
/*0x2E*/     goto L2
```

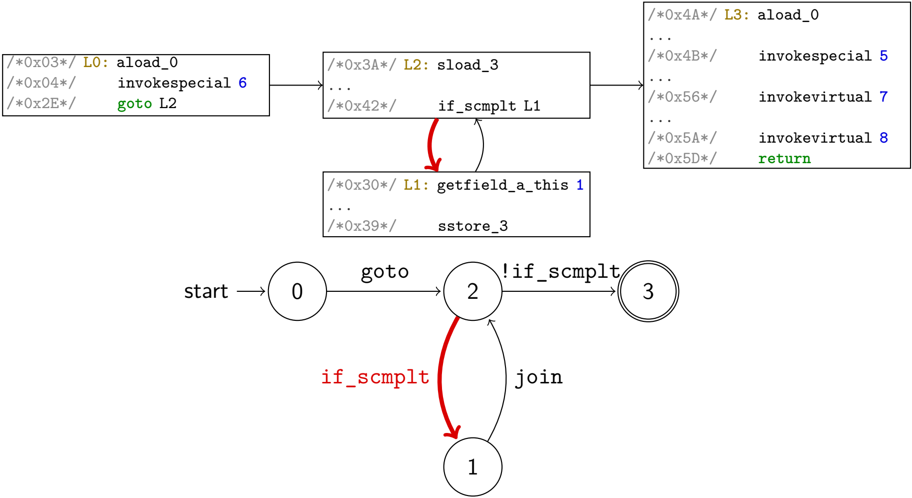
```
/*0x3A*/ L2: sload_3
...
/*0x42*/     if_scmplt L1
```

```
/*0x4A*/ L3: aload_0
...
/*0x4B*/     invokespecial 5
...
/*0x56*/     invokevirtual 7
...
/*0x5A*/     invokevirtual 8
/*0x5D*/     return
```

```
/*0x30*/ L1: getfield_a_this 1
...
/*0x39*/     sstore_3
```



Security Automaton included in the JCVM



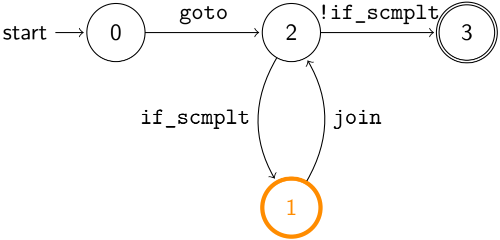
Security Automaton included in the JCVM

```
/*0x03*/ L0: aload_0
/*0x04*/     invokespecial 6
/*0x2E*/     goto L2
```

```
/*0x3A*/ L2: sload_3
...
/*0x42*/     if_scmplt L1
```

```
/*0x4A*/ L3: aload_0
...
/*0x4B*/     invokespecial 5
...
/*0x56*/     invokevirtual 7
...
/*0x5A*/     invokevirtual 8
/*0x5D*/     return
```

```
/*0x30*/ L1: getfield_a_this 1
...
/*0x39*/     sstore_3
```



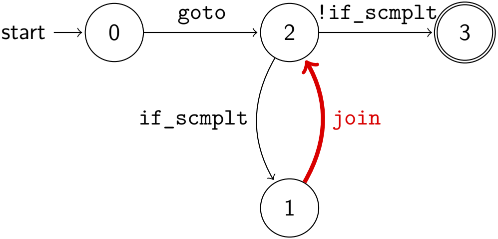
Security Automaton included in the JCVM

```
/*0x03*/ L0: aload_0
/*0x04*/     invokespecial 6
/*0x2E*/     goto L2
```

```
/*0x3A*/ L2: sload_3
...
/*0x42*/     if_scmplt L1
```

```
/*0x30*/ L1: getfield_a_this 1
...
/*0x39*/     sstore_3
```

```
/*0x4A*/ L3: aload_0
...
/*0x4B*/     invokespecial 5
...
/*0x56*/     invokevirtual 7
...
/*0x5A*/     invokevirtual 8
/*0x5D*/     return
```



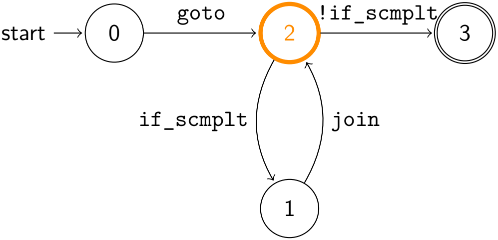
Security Automaton included in the JCVM

```
/*0x03*/ L0: aload_0
/*0x04*/     invokespecial 6
/*0x2E*/     goto L2
```

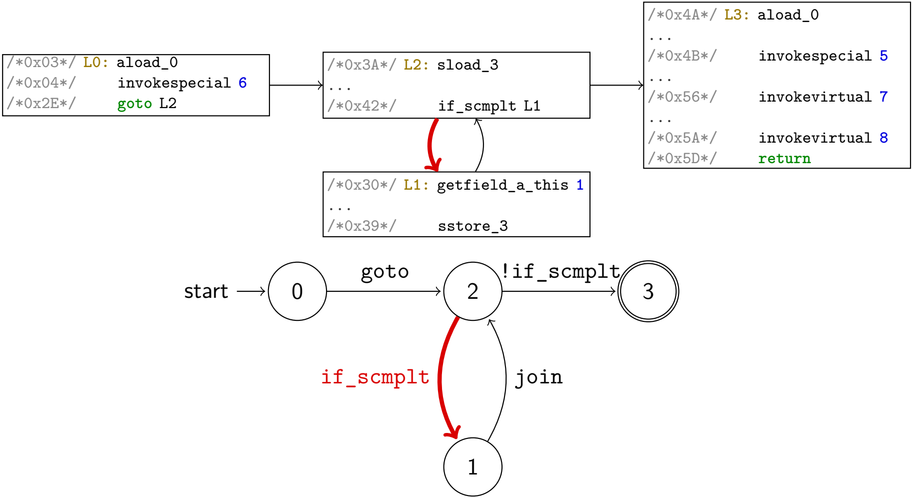
```
/*0x3A*/ L2: sload_3
...
/*0x42*/     if_scmplt L1
```

```
/*0x30*/ L1: getfield_a_this 1
...
/*0x39*/     sstore_3
```

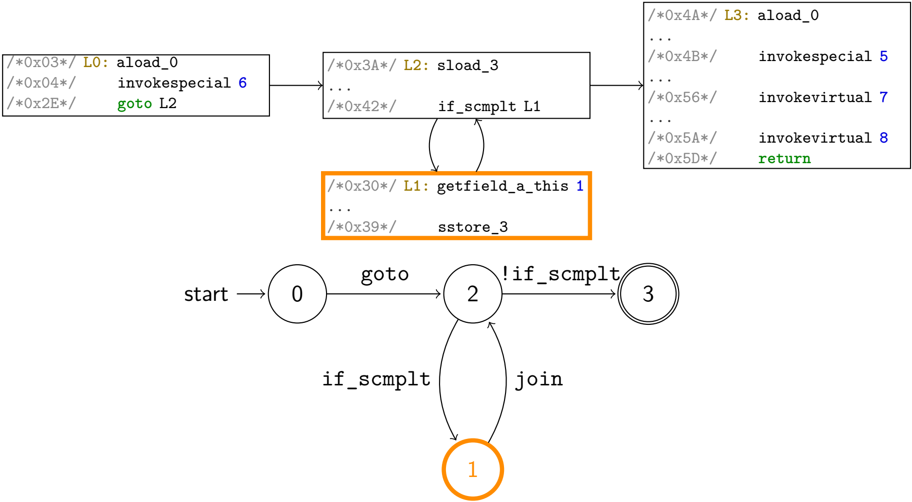
```
/*0x4A*/ L3: aload_0
...
/*0x4B*/     invokespecial 5
...
/*0x56*/     invokevirtual 7
...
/*0x5A*/     invokevirtual 8
/*0x5D*/     return
```



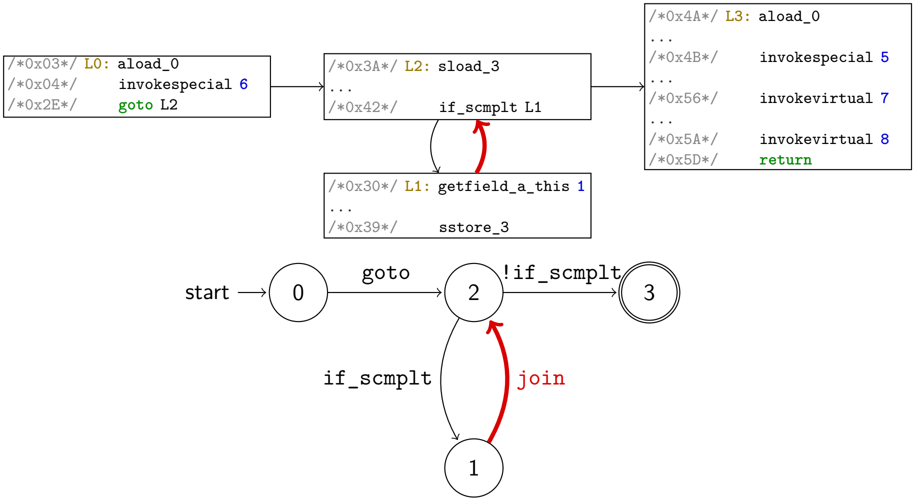
Security Automaton included in the JCVM



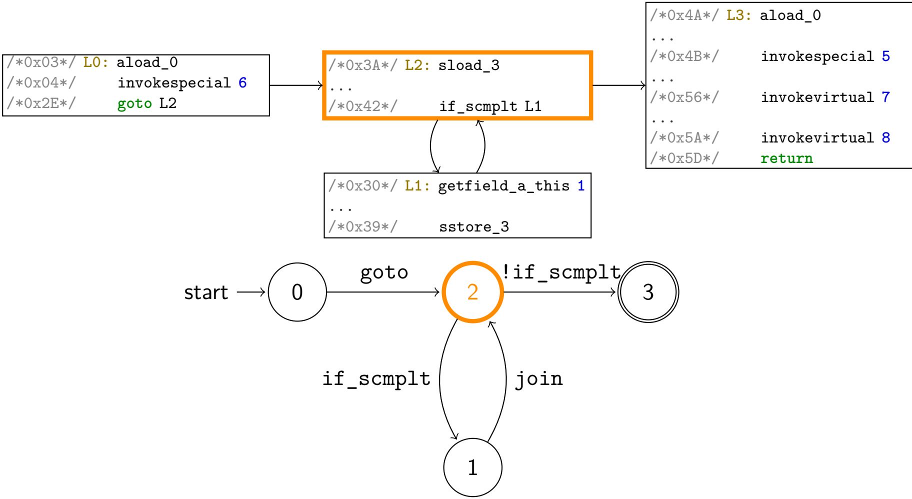
Security Automaton included in the JCVM



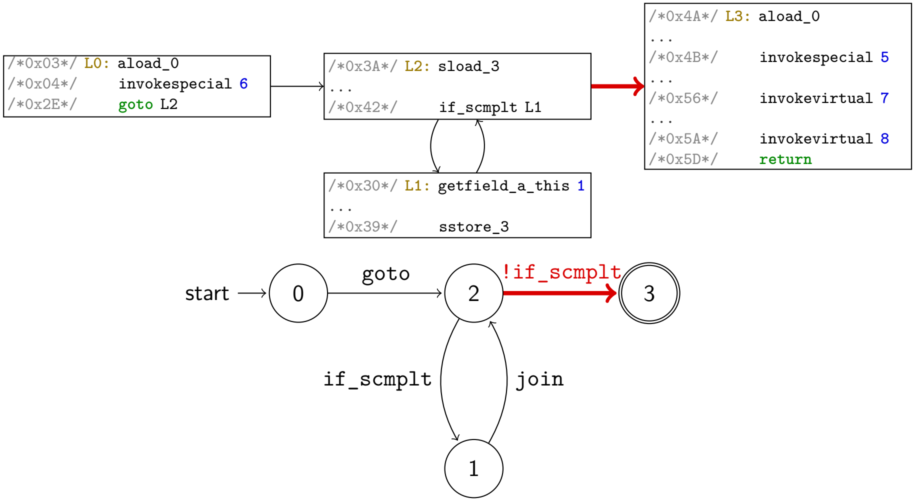
Security Automaton included in the JCVM



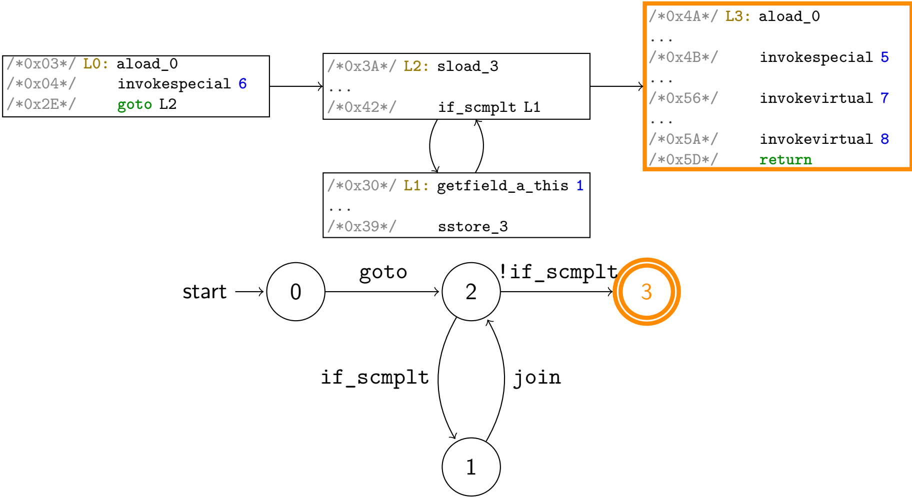
Security Automaton included in the JCVM



Security Automaton included in the JCVM



Security Automaton included in the JCVM



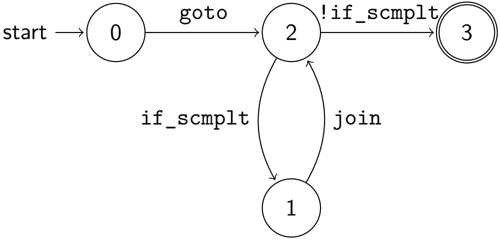
Security Automaton included in the JCVM

```
/*0x03*/ L0: aload_0
/*0x04*/     invokespecial 6
/*0x2E*/     goto L2
```

```
/*0x3A*/ L2: sload_3
...
/*0x42*/     if_scmplt L1
```

```
/*0x30*/ L1: getfield_a_this 1
...
/*0x39*/     sstore_3
```

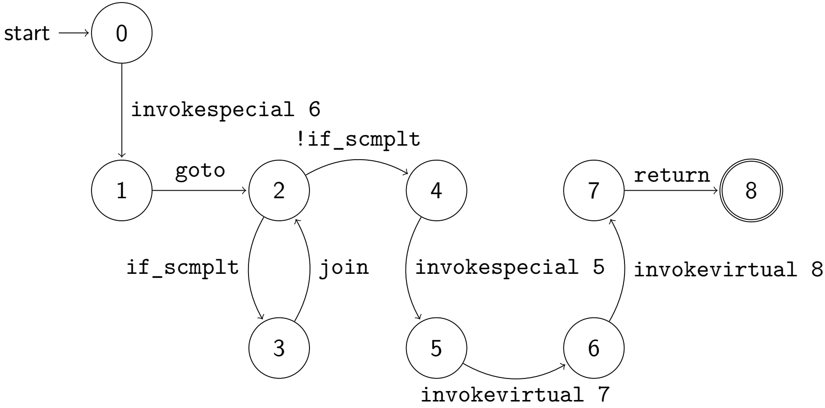
```
/*0x4A*/ L3: aload_0
...
/*0x4B*/     invokespecial 5
...
/*0x56*/     invokevirtual 7
...
/*0x5A*/     invokevirtual 8
/*0x5D*/     return
```



The trace recognised would be:

(goto, (if_scmplt, join)*, !if_scmplt, return)

Security Automaton included in the JCVM (Cont.)



Security Automaton included in the JCVM (Cont.)

δ \ q	q_0	q_1	q_2	q_3	q_4	q_5	q_6	q_7
`invokespecial 6`	q_1							
`goto`		q_2						
`join`				q_2				
`if_scmplt`			$q_{3,4}$					
`invokespecial 5`					q_5			
`invokevirtual 7`						q_6		
`invokevirtual 8`							q_7	
`return`								$+$

The Security Automaton

- The execution flow is checked by the security automaton upon a finite state machine;
- Each transition is verified by the execution monitor;
- The CFG can be automatically computed by the loading process;
- The CFG can be encoded upon a sparse matrix $\rightarrow$ optimised solution to store the CFG
- The JCVM and the loader should be modified to handle automatons.

Outline

Experimental Results

Reference	Java Card	GP	Characteristics
a-21a	2.1.1	2.0.1	256 kB EEPROM, SIM card
a-21b	2.1.1	2.0.1	Same as a-21a plus RSA
a-22a	2.2	2.1	64 kB EEPROM, RSA
a-22b	2.1.1	2.0.1	32 kB EEPROM, dual interface, RSA
a-22c	2.2.1	2.1.1	36 kB EEPROM,
b-21a	2.1.1	2.1.2	16 kB EEPROM, dual interface
b-22a	2.1.1	2.0.1	16 kB EEPROM, hardware DES
b-22b	2.2.1	2.1.1	72 kB EEPROM, dual interface
c-22a	2.1.1	2.0.1	64 kB EEPROM, RSA
c-22b	2.2	2.1.1	64 kB EEPROM, dual interface, RSA
c-22c	2.2	2.1.1	72 kB EEPROM, dual interface, RSA
d-21	2.1	2.0.1	32 kB EEPROM, RSA
d-22	2.2.1	2.1.1	16 kB EEPROM
e-22	2.2	2.1	72 kB EEPROM, RSA

Developed Tools

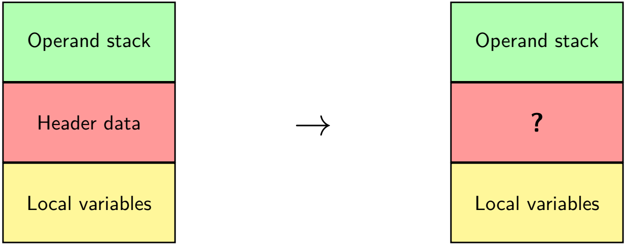
- CapMap
  - Java-framework;
  - Provides reading and modification of CAP files;
  - Correcting CAP file interdependencies.
- OPAL
  - Java-Library and GUI;
  - Supports Global Platform 2.x specification;
  - Open-source project (available on Bitbutcket)

CAP MAP
Cap File Manipulator

OPAL
Open Platform Access
An open source implementation of the Global Platform Card specifications

Experimental Results: EMAN2

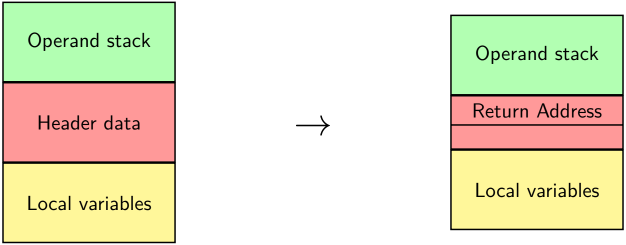
Characterisation of the Stack Implementation



Reference	Header size	Return Address
a-21a	2 entries	+2
a-21b	2 entries	+2
a-22a	2 entries	+2
a-22b	3 entries	+1
a-22c	3 entries	+1
d-22	✗	✗
e-22	✗	✗

Experimental Results: EMAN2

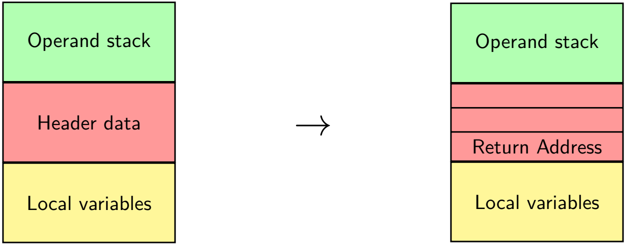
Characterisation of the Stack Implementation



Reference	Header size	Return Address
a-21a	2 entries	+2
a-21b	2 entries	+2
a-22a	2 entries	+2
a-22b	3 entries	+1
a-22c	3 entries	+1
d-22	✗	✗
e-22	✗	✗

Experimental Results: EMAN2

Characterisation of the Stack Implementation



Reference	Header size	Return Address
a-21a	2 entries	+2
a-21b	2 entries	+2
a-22a	2 entries	+2
a-22b	3 entries	+1
a-22c	3 entries	+1
d-22	✗	✗
e-22	✗	✗

Experimental Results: EMAN2

The Attack

```
short setReturnAddress(short new_address) {
  01 // flags: 2 max_stack : 1
  20 // nargs: 2 max_locals: 0
  aload_1  // pushing the new_address value
  sstore Y // Overwriting the return address
           // with the new_address parameter
  return   // jumping to the shellcode ;-)
}
```

Reference	Header size	Y (Return Address)	EMAN2
a-21a	2 entries	nargs+max_locals+2	✓
a-21b	2 entries	nargs+max_locals+2	✓
a-22a	2 entries	nargs+max_locals+2	✓
a-22b	3 entries	nargs+max_locals+1	✓
a-22c	3 entries	nargs+max_locals+1	✓
d-22	✗	✗	✗
e-22	✗	✗	✗

Experimental Results: `finally`-Clause Corruption

```
short jsrAttack () {
  01 // flags: 0 max_stack : 1
  11 // nargs: 1 max_locals: 1
  /*0x53*/ L0: jsr L1
  /*0x56*/ L2: sspush 0xCAFE
  /*0x59*/     sreturn
  /*0x5A*/ L3: sspush 0xBEEF
  /*0x5D*/     sreturn
  /*0x5E*/ L1: astore_1
  /*0x5F*/     sinc 0x1, 0x4
  /*0x62*/     ret 1 // -> L3
}
```

Reference	Result
a-21a	✓
a-21b	✓
a-22a	✓
a-22b	✓
a-22c	✓
b-21a	✓
b-22a	✓
b-22b	✓
c-22a	✓
c-22b	✓
c-22c	✓
d-21	✓
d-22	✓
e-22	✓

Experimental Results: Comparison

Reference	EMAN2	`finally`-Clause Corruption
`a-21a`	✓	✓
`a-21b`	✓	✓
`a-22a`	✓	✓
`a-22b`	✓	✓
`a-22c`	✓	✓
`b-21a`	-	✓
`b-22a`	-	✓
`b-22b`	-	✓
`c-22a`	-	✓
`c-22b`	-	✓
`c-22c`	-	✓
`d-21`	-	✓
`d-22`	✗	✓
`e-22`	✗	✓

Experimental Results: The Security Automatons

- A modification of the JCVM is required;
- The loading process computes the state matrix:
  - Processing time depends on the CFG granularity;
  - The state matrix is stored in the EEPROM;
- During the execution, the execution monitor checks the transition:
  - `if_scmplt`: **21%**
  - General case: **5,13%**
    - 45 on 184 instructions are overloaded
  - Real case: **1,58%**
    - 7 on 93 instructions are overloaded

Outline

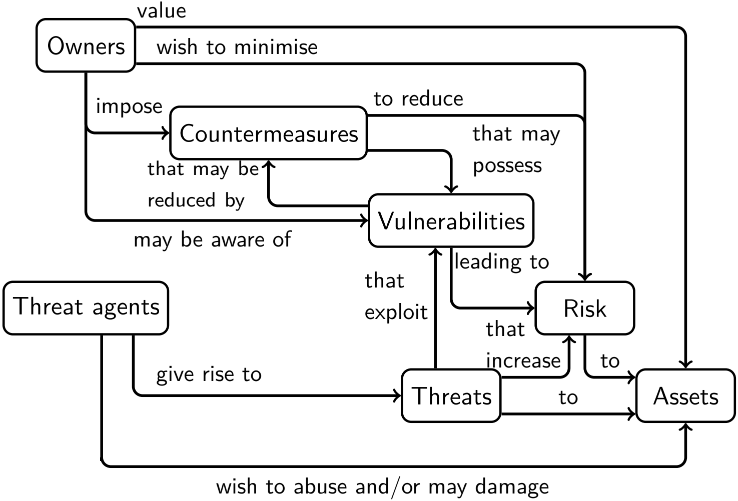
Conclusion

- This thesis aimed at designing **efficient** and **affordable** countermeasure using a **top-down** approach;
- It is based on the **Fault Tree Analysis** which this approach aims at being **generic**;
- We identified major undesirable events:
    - We discovered new attack paths, someones are generic;
    - And introduced high level-countermeasures.

Conclusion (Cont.)

- We focused on the **code integrity**:
  - Modification of the control flow;
  - Corruption of the Java Card Linker [Hamadouche et al., SAR-SSI 2012], [Razafindralambo et al., SNDS 2012] and [Bouffard et al., CRiSIS 2013];
- Each evaluated attacks succeeded on different cards
  - Bottom-up approach ?
  - We wear a white hat;
- Our approach aims at helping card manufacturers to clearly identify the assets to protect.

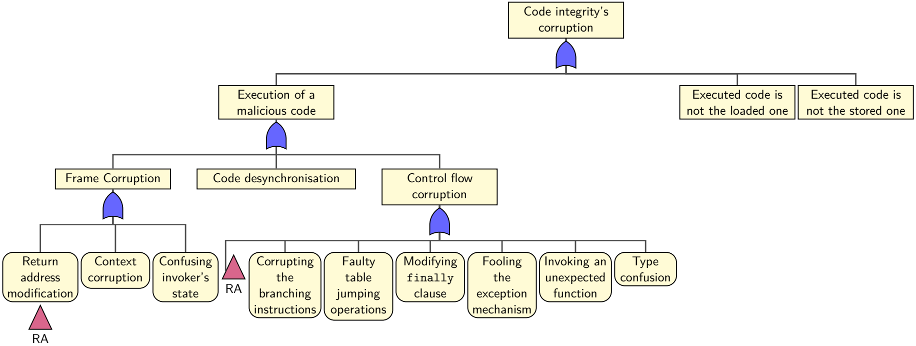
Common Criteria and the Fault Tree Analysis

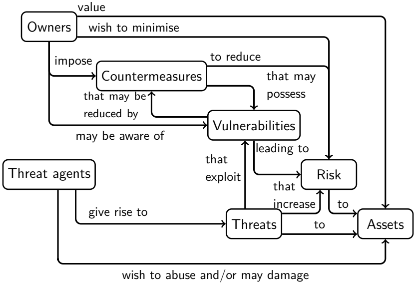


Common Criteria for Information Technology Security Evaluation

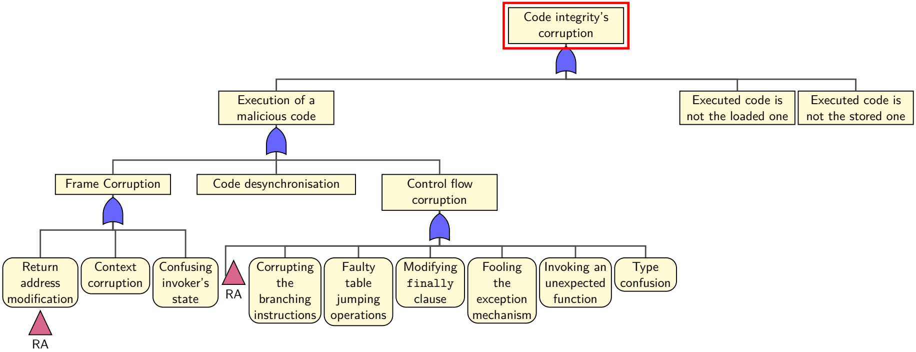
ISO/IEC 15048: Evaluation criteria for IT security – Part 1: Introduction and general model

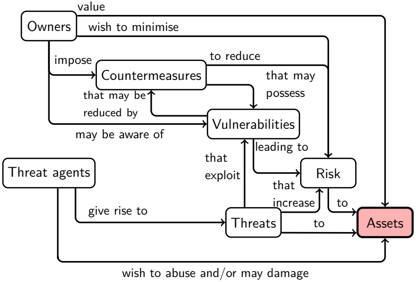
Common Criteria and the Fault Tree Analysis



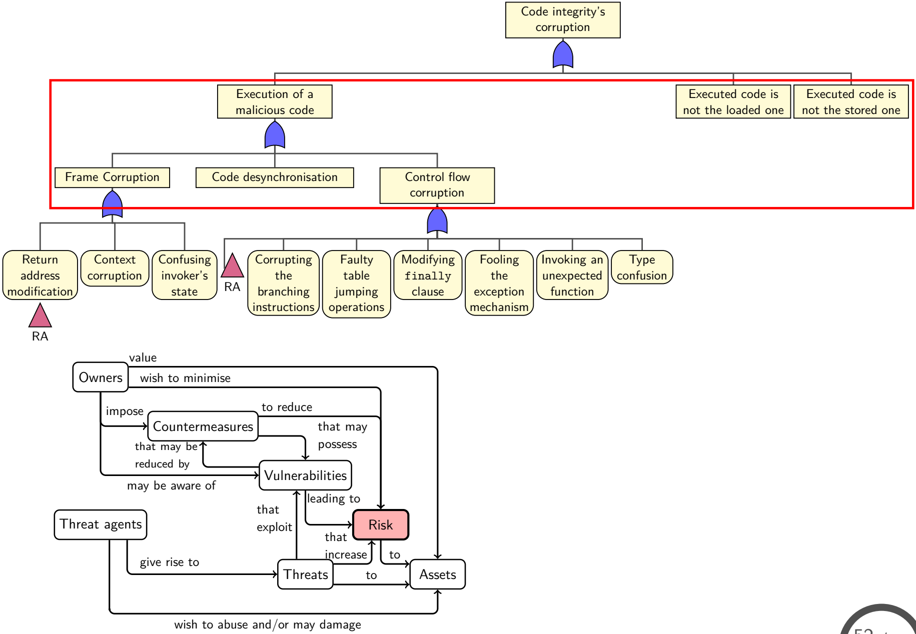


Common Criteria and the Fault Tree Analysis

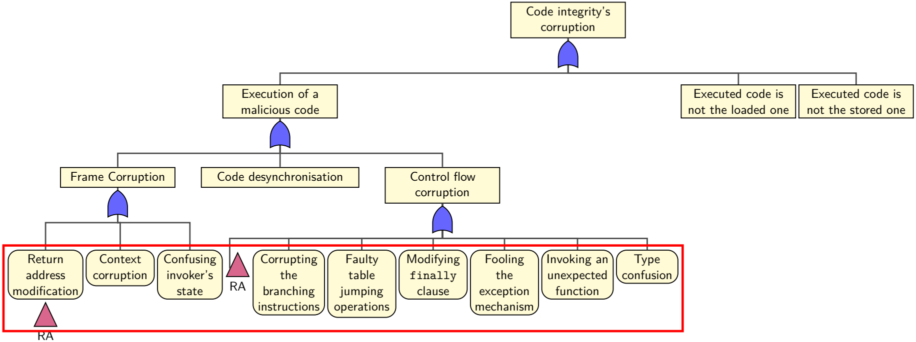


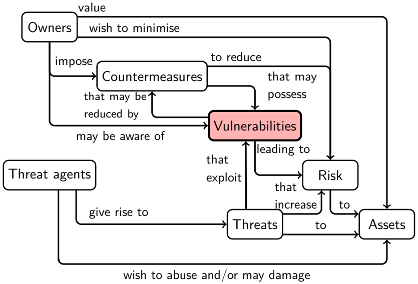


Common Criteria and the Fault Tree Analysis

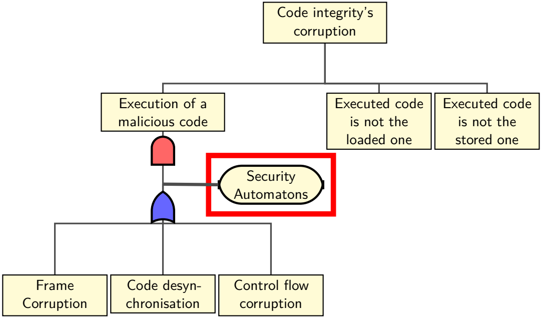


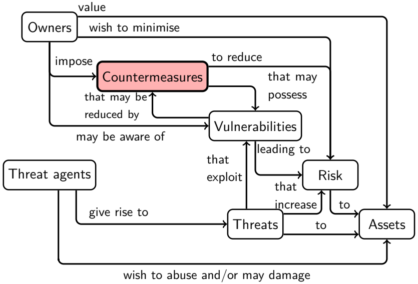
Common Criteria and the Fault Tree Analysis





Common Criteria and the Fault Tree Analysis





Future Works

- The dissertation focused on ***How to execute ill-formed code?***;
  - To do: checking the **installation process**;
- Analysing the code **data integrity tree** and the code and data **confidentiality trees**;
- Designing **dynamic FTA** to take into account events' order;
- Considering **quantification** of the probability for an attacker to reach his objective:
  - Given time or overall mean time for the attack to overcome it;
  - On-going work based on Boolean logic Driven Markov Process.

Thank you for your attention!
Questions?

Publications

During my PhD thesis, I have co-written **25 publications**:

- 2 book chapters;
- 4 journal articles and 1 in the reviewing process;
- 3 invited conferences;
- 10 articles in international conferences with review and proceedings;
- 4 articles in national conferences with review and proceedings;
- 1 articles in national conferences with review and without proceeding;
- 1 posters.

Scrambling the memory

$$ins_{hidden} = ins \oplus K_{bytecode}$$ [Barbu's PhD Thesis, 2012]

$$ins_{hidden} = ins \oplus K_{bytecode} \oplus JPC$$ [Razafindralambo et al., SNDS 2012]

Scrambling the memory

$$\boxed{ins_{hidden} = ins \oplus K_{bytecode}}$$

[Barbu's PhD Thesis, 2012]

$$ins_{hidden} = ins \oplus K_{bytecode} \oplus JPC$$

[Razafindralambo et al., SNDS 2012]

```
0x8068:   0x00 nop
0x8069:   0x02 sconst_1
0x806A:   0x02 sconst_1
0x806B:   0x3C pop2
0x806C:   0x04 sconst_1
0x806D:   0x3B pop
```

Original code

```
0x8068:   0x42 nop
0x8069:   0x40 sconst_1
0x806A:   0x40 sconst_1
0x806B:   0x7E pop2
0x806C:   0x46 sconst_1
0x806D:   0x79 pop
```

[Barbu's PhD Thesis, 2012] with $K_{bytecode} = 0x42$

Scrambling the memory

$$ins_{hidden} = ins \oplus K_{bytecode}$$ [Barbu's PhD Thesis, 2012]

$$ins_{hidden} = ins \oplus K_{bytecode} \oplus JPC$$ [Razafindralambo et al., SNDS 2012]

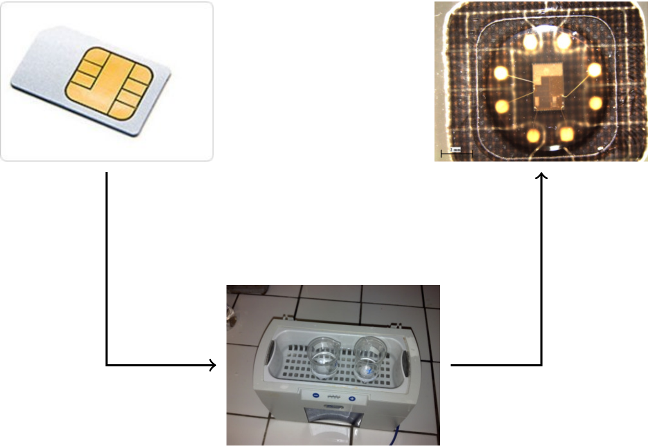
```
0x8068:   0x42 nop
0x8069:   0x40 sconst_1
0x806A:   0x40 sconst_1
0x806B:   0x7E pop2
0x806C:   0x46 sconst_1
0x806D:   0x79 pop
```

[Barbu's PhD Thesis, 2012] with $K_{bytecode} = 0x42$

```
0x8068:   0x2A nop
0x8069:   0x29 sconst_1
0x806A:   0x2A sconst_1
0x806B:   0x15 pop2
0x806C:   0x2D sconst_1
0x806D:   0x12 pop
```

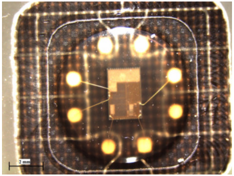
[Razafindralambo et al., SNDS 2012] with $K_{bytecode} = 0x42$

Chip Extraction



Acetone solution in a ultrasonic tank.

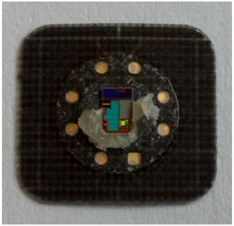
How to remove the resin?

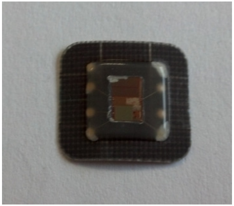


Solution to extract:

- Oxygenated water or
- 50/50 vol/vol methanol/chloroform

Simmer during 3 hours in a ultrasonic tank.





Chips Analysed with a Scanning Electron Microscope