Java Type Confusion and Fault Attacks

Olli Vertanen
University of Kuopio
Finland

FTDC 2006 Workshop
Pacifico Yokohama
October 10, 2006

About the Presentation

- What is this?
  - Some observations about the Java bytecode, Java virtual machine and design of fault attacks.
- Programming language view - not directly cryptography related.
- Emphasis on the embedded systems - also secure devices e.g. smart cards.

The Problem

Can we design fault attacks at the Java bytecode level that cause type confusion situations?

Type Confusion, ...

- Type confusion is, in effect, an illegal type cast:

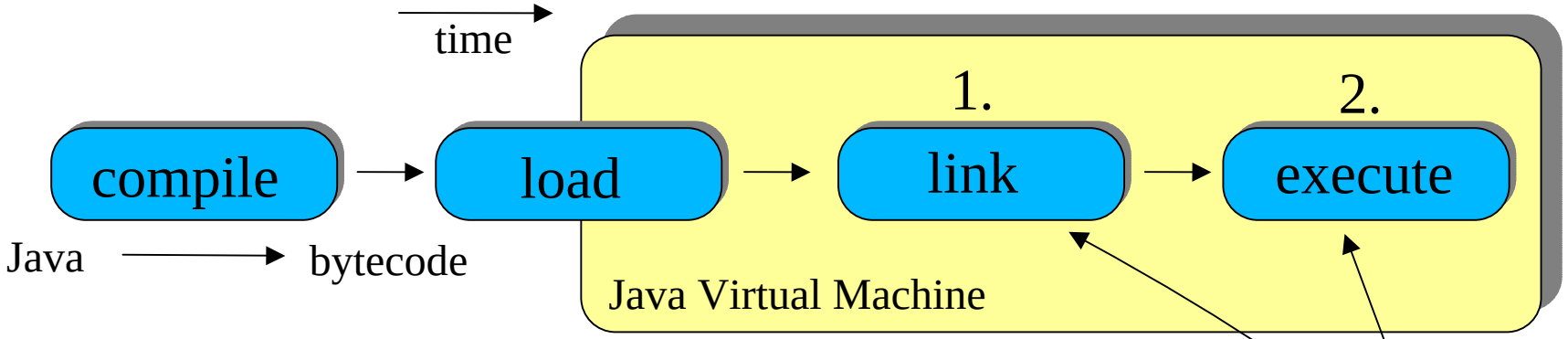
```
int illegalCast(Object ref)
{
   return ref;
}
```

- Java should be a type safe language, but many known Java exploits are based on type confusion.
- The verifier (a part of the Java virtual machine, JVM) plays crucial role in maintaining type safety.

... Java,

1. Verification takes place during the linkage phase.

2. Executed bytecode is presumed to be verified.



1. & 2. BUT: What if verified programs are modified?
Does the time cap introduce a vulnerability?

TOCTOU: “Time Of Check, Time Of Use”

... and Fault Attacks

- Inspiration 1: Bar-El et al. “The Sorcerer's Apprentice Guide to Fault Attacks” (FTDC 2004):

  “[...], the processor just skipped a number of instructions and resumed normal execution several microseconds after the glitch.”

- Inspiration 2: Govindavajhala & Appel, 2003, “Using Memory Errors to Attack Virtual Machine”:
  - Java type confusion using a 50W light bulb!
  - Required specially designed software.

The Problem (revised)

Given the background,

- TOCTOU introduced by the verifier in the JVM,
- focused “instruction skipping” glitches,

can we design type confusion attacks at the Java bytecode level? Attacks should:

- be well focused,
- use verifiable programs.

An Example (1)

Java:

```
int illegalCast(Object ref)
{
    Object o = ref;

    int i = 1;

    return i;

}
```

bytecode:

```
aload 1
astore 2
iconst 1
istore 3
iload 3
ireturn
```

Specially designed code ...

An Example (2)

... combined with a focused fault ...

```
int illegalCast(Object ref)
{
    Object o = ref;

    int i = 1;

    return i;
    return ref;
}
```

```
aload 1
astore 2
iconst 1
istore 3
iload 3
ireturn
```

... can lead to type confusion! (“operand snatching”)

Other Possible Java Targets

- `checkcast` instructions:
  - run-time type compatibility check
- Attacking program counter:
  - The bytecode has clear byte boundaries
  - Shifting the PC changes radically meaning of the program
- Sub-instructions (run-time checks):
  - Array boundary checks
  - Null pointer checks

Attacking Real Systems (1)

What kind of Java...

1.Interpreted?	a. switched b. direct threading c. in-line threading
2.Compiled?	a. just in time b. ahead of time c. HotSpot
3. Hardware?	a. hardware translation b. Java processor c. co-processor

... and how does the bytecode map to native instructions of the underlying architecture?

Attacking Real Systems (2)

- It looks like:
  - small embedded systems are more likely to maintain the stack model in computation (even when dynamically compiled), thus making operand snatching easier.

Counter-measures

- Fault-detection and recovery methods can be applied.
- Change from stack machine to register machine.
- The problem raised mainly because of TOCTOU condition in the JVM. Resolve by removing the TOCTOU condition?
  - Defensive Java Virtual Machine (work in progress).

Conclusions

- We have presented a method to attack Java type system using a combination of focused glitches and malicious programs.
- The attack scheme can confuse the Java type system when a normal Java virtual machine is used.
- Open questions:
  - How to apply the method in practice?
  - How to apply the method to an arbitrary program?

Thank You for Your time!

Comments?
Questions?