# Java Card Applet Firewall Exploration and Exploitation

Wojciech Mostowski and Erik Poll
Digital Security
Radboud University Nijmegen
The Netherlands

http://www.cs.ru.nl/~{woj,erikpoll}/

# Introduction

Study of the Java Card firewall mechanism in connection with research on Java Card malicious code vulnerabilities:

- Firewall specification study
- Firewall compliance tests
- Shareable Interface Object as a way to introduce type confusion on the card
- Type confusion + firewall weakness → AID exploit
- Experimental studies on 8 cards (4 producers)

# Java Card Specifications

- Specifications assume type correctness, i.e. bytecode is type correct.
- Not always clear at first sight – cause of implementation mistakes
- Followed carefully to construct a compliance test
- Smaller and bigger noncompliance:
  - Smaller: security is preserved, but the specification not followed to the letter
  - Bigger: possible security (or at least robustness) problems
- Java Card 3.0 Classic Edition essentially the same as for 2.2.X

# Java Card Firewall

- Runtime protection mechanism
- Provides applet data separation: each reference belongs and is confined to a context (applet), foreign reference is not accessible, including type information
- Provides applet data sharing: a reference can be explicitly tagged as shareable – declared methods accessible to anyone
- The Java Card Runtime Environment has root privilege: can read and write anything
- JCRE data not accessible to anyone, unless it is special, e.g. JCRE entry points
- Again: specs assume type correctness – can we exploit the firewall with broken bytecode?

# Java Card Firewall Test

The firewall compliance test:

- Test all firewall features / requirements one by one
- Only features testable from the applet level are tested
- Give warnings in human readable form
- A few ideas borrowed from Riscure's JCWorkBench, a few ideas transferred to JCWorkBench
- Out of 8 cards 5 were testable, the rest refused to install code using shareable interfaces (probable cause: bytecode verifier, loader parameters)

# Noncompliance #1

Query the `Shareable` interface status:

```
if (o instanceof Shareable) ...
```

# Noncompliance #1

Query the `Shareable` interface status:

```
if (o instanceof Shareable) ...
```

## Specification on **`instanceof`**

- o belongs to other context and is not shareable → `SecurityException`

# Noncompliance #1

Query the `Shareable` interface status:

```
if (o instanceof Shareable) ...
```

## Specification on **`instanceof`**

o belongs to other context and is not shareable → `SecurityException`

## Cards

Only one card non-compliant: it says **`false`**.

# Noncompliance #1

Query the `Shareable` interface status:

```
if (o instanceof Shareable) ...
```

## Specification on **`instanceof`**

o belongs to other context and is not shareable → `SecurityException`

## Cards

Only one card non-compliant: it says **`false`**.

## Severity

None: the overall check results are equivalent

# Noncompliance #2

Privileged API methods (system owned AID instance):

```
public boolean equals(Object o);
```

# Noncompliance #2

Privileged API methods (system owned AID instance):

```
public boolean equals(Object o);
```

## Required checks

1. firewall check: o is accessible to the calling context
2. o is an AID? if not return **`false`**
3. compare the AID bytes: return **`true`** or **`false`**

# Noncompliance #2

Privileged API methods (system owned AID instance):

```
public boolean equals(Object o);
```

## Required checks

1. firewall check: o is accessible to the calling context
2. o is an AID? if not return **`false`**
3. compare the AID bytes: return **`true`** or **`false`**

## Cards

Two cards do 2-1-3, others 1-2-3

# Noncompliance #2

Privileged API methods (system owned AID instance):

```
public boolean equals(Object o);
```

## Required checks

1. firewall check: o is accessible to the calling context
2. o is an AID? if not return **`false`**
3. compare the AID bytes: return **`true`** or **`false`**

## Cards

Two cards do 2-1-3, others 1-2-3

## Severity

Very Mild: 2-1-3 can reveal that o **is** an AID

# Noncompliance #3

Accessing an array belonging to another context:

```
a[i] = x;
```

# Noncompliance #3

Accessing an array belonging to another context:

```
a[i] = x;
```

Specification

Should result in **SecurityException**

# Noncompliance #3

Accessing an array belonging to another context:

```
a[i] = x;
```

## Specification

Should result in **Security**Exception

## Cards

One card reports **System**Exception

# Noncompliance #3

Accessing an array belonging to another context:

```
a[i] = x;
```

## Specification

Should result in **SecurityException**

## Cards

One card reports **SystemException**

## Severity

None: the overall result is the same

# Noncompliance #4

Creation of and accessing clear-on-deselect arrays

# Noncompliance #4

Creation of and accessing clear-on-deselect arrays

## Specification

Forbidden when the context is not the currently selected applet context

# Noncompliance #4

Creation of and accessing clear-on-deselect arrays

## Specification

Forbidden when the context is not the currently selected applet context

## Cards

One card overdoes this: creation of clear-on-reset arrays is also not possible, while only clear-on-deselect should not be

# Noncompliance #4

Creation of and accessing clear-on-deselect arrays

## Specification

Forbidden when the context is not the currently selected applet context

## Cards

One card overdoes this: creation of clear-on-reset arrays is also not possible, while only clear-on-deselect should not be

## Severity

Very mild: limits the functionality of the card

# Noncompliance #5

Non-multiselectable applets and SIOs

# Noncompliance #5

Non-multiselectable applets and SIOs

## Specification

Access to SIO is forbidden if the server is not multiselectable and is active on another logical channel

# Noncompliance #5

Non-multiselectable applets and SIOs

## Specification

Access to SIO is forbidden if the server is not multiselectable and is active on another logical channel

## Cards

One card ignores this: access always granted

# Noncompliance #5

Non-multiselectable applets and SIOs

## Specification

Access to SIO is forbidden if the server is not multiselectable and is active on another logical channel

## Cards

One card ignores this: access always granted

## Severity

Semi serious: the applet has to keep track of its selections by itself to prevent problems with multiple access from outside

# Unexplained Specifications

Relates to multiselectable applets and clear-on-deselect arrays

# Unexplained Specifications

Relates to multiselectable applets and clear-on-deselect arrays

Spec: Rule X applies.

# Unexplained Specifications

Relates to multiselectable applets and clear-on-deselect arrays

Spec: Rule X applies.

Spec: Rule Y applies (even if condition A is met).

# Unexplained Specifications

Relates to multiselectable applets and clear-on-deselect arrays

Spec: Rule X applies.

Spec: Rule Y applies (even if condition A is met).

Problem: Seemingly condition A cannot possibly take place in scenario Y, because rule X forbids this in the first place.

# Unexplained Specifications

Relates to multiselectable applets and clear-on-deselect arrays

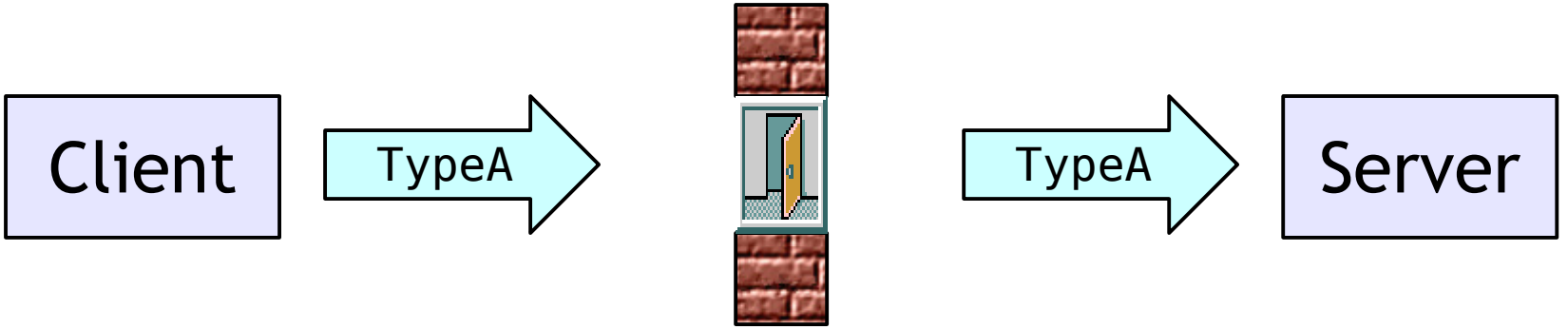
Spec: Rule X applies.

Spec: Rule Y applies (even if condition A is met).

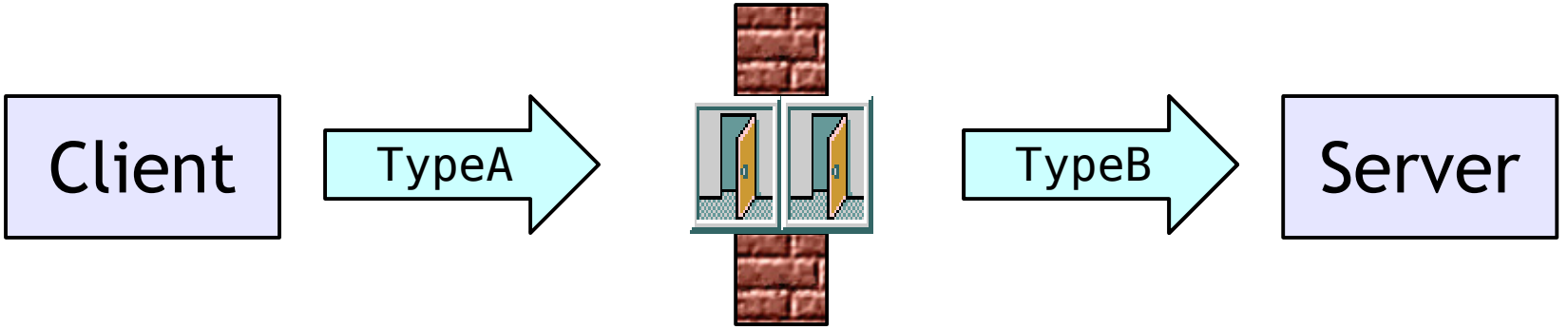
Problem: Seemingly condition A cannot possibly take place in scenario Y, because rule X forbids this in the first place.

Only very careful analysis reveals the other condition for A to be met in scenario Y. But the short comment *"(even if condition A is met)"* is not given a detailed explanation.

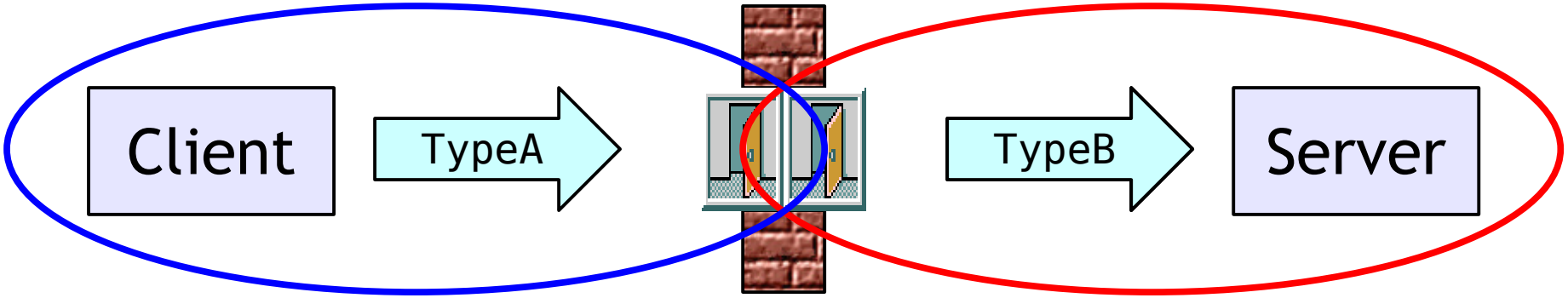
# Type Confusion via Shareable Interfaces



# Type Confusion via Shareable Interfaces

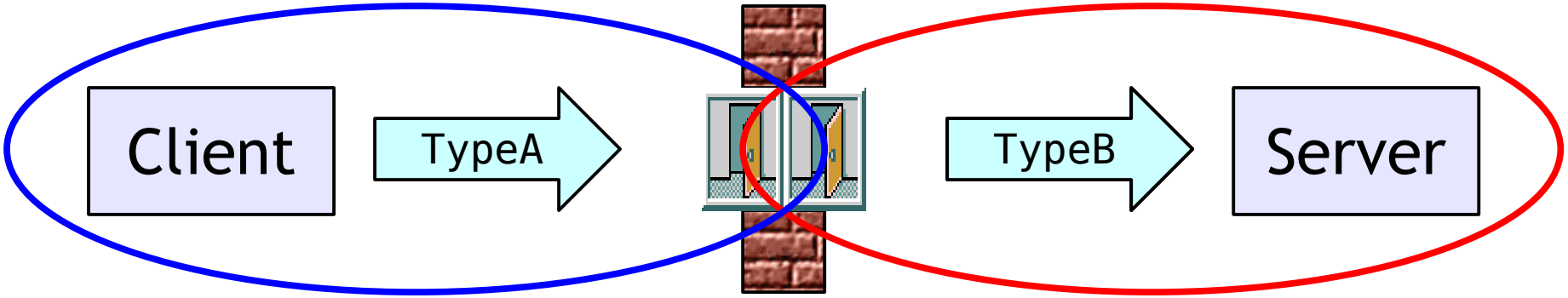


# Type Confusion via Shareable Interfaces




- Client and server compiled and installed at different times
- Change the definition of the shareable interface in the meantime
- The loader does not catch such changes, BCV does, but then, forbids SIOs altogether (Non-compliance #6?!)
- Two interfaces ➞ two types ➞ type confusion

# Type Confusion via Shareable Interfaces




- Client and server compiled and installed at different times
- Change the definition of the shareable interface in the meantime
- The loader does not catch such changes, BCV does, but then, forbids SIOs altogether (Non-compliance #6?!)
- Two interfaces ➞ two types ➞ type confusion

Client thinks:

```
void service(TypeA a);
```

Server thinks:

```
void service(TypeB a);
```

Whether a type confusion (introduced this or any other way) can be exploited is another subject [CARDIS 2008].

# AID Exploit

The scenario:

- Certain kind of a type attack has to be possible: direct object access and reference switching

```
public class AID {
  private byte[] aidBytes;
  ...
```

# AID Exploit

The scenario:

- Certain kind of a type attack has to be possible: direct object access and reference switching

```
public class AID {
  private byte[] aidBytes;
  ...
```

The result:

- Malicious applet can change these **aidBytes** references, and hence change the applet AID registry in any way!
- In turn real impersonation of an applet possible

# AID Exploit

The scenario:

- Certain kind of a type attack has to be possible: direct object access and reference switching

```
public class AID {
  private byte[] aidBytes;
  ...
```

The result:

- Malicious applet can change these **aidBytes** references, and hence change the applet AID registry in any way!
- In turn real impersonation of an applet possible

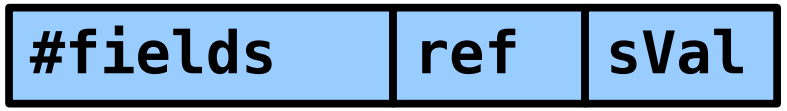
In reality subject to: BCV, code signing, runtime type checking, etc. But, it was possible on two open cards!

Similar exploit allows to bypass firewall, but has limitations.

# Direct Reference Manipulation Details

Confuse an object with an array:

- An object

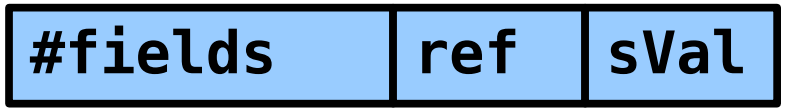



```
public class TestClass {
    Object ref = new Object();
    short sVal = 10;
}
```

# Direct Reference Manipulation Details

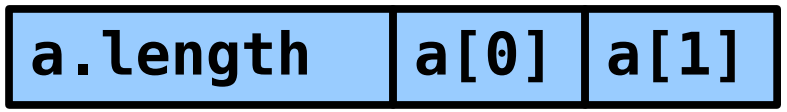
Confuse an object with an array:

- An object




```
public class TestClass {
    Object ref = new Object();
    short sVal = 10;
}
```

- An array

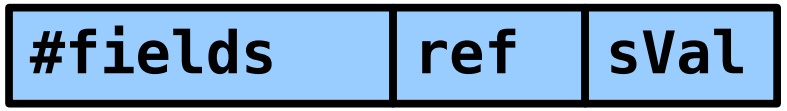



```
a.length: 2
a[0]:     0x09E0 // ref
a[1]:     0x000A // sVal
```

# Direct Reference Manipulation Details

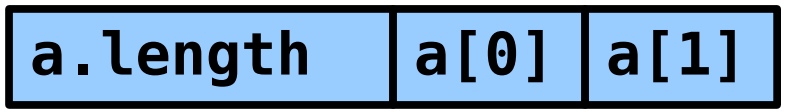
Confuse an object with an array:

- An object




```
public class TestClass {
    Object ref = new Object();
    short sVal = 10;
}
```

- An array




```
a.length: 2
a[0]:     0x09E0 // ref
a[1]:     0x000A // sVal
```

- All reference values readable and writable directly, public access

# Discussion and Conclusion

- Specifications are not followed to the letter: implementations still safe, but non-compliances question platform interoperability (what is TCK for?)
- Specifications (although correct) still leave a little bit to be desired, Java Card 3.0 does not change the picture
- Restrictive on-card BCV non-compliant?
- The tricks and exploits are possible because of
  - insufficient protection mechanisms against malicious byte code
  - weak firewall design
- Out of 8 cards tested:
  - 4 are non-compliant (one vulnerable to AID exploit)
  - 3 not fully tested (BCV forbids SIOs)
  - 1 fully compliant, but vulnerable to AID exploit

# The End

Questions?