The Java Development Kit (JDK)

Java Code
2M Lines

Why?
Approach
Challenges
Results
Moving Forward

Robusta:
Taming the Native Beast of the JVM
The Java Development Kit (JDK)

Robusta:
Taming the Native Beast of the JVM

Why?
Approach
Challenges
Results
Moving Forward

Java Code
2M Lines

Native (C/C++) Code
800,000 lines (1.6)
The Java Development Kit (JDK)

Java Code
2M Lines

Native (C/C++) Code
800,000 lines (1.6)

59 Security critical bugs
(38,000 lines)

*Tan and Croft, 2008
Mobile Code

Internet

GUI Implementation (Java)

Image Processing Logic (Legacy C++)

NAPPLET

Why?
Approach
Challenges
Results
Moving Forward

Robusta:
Taming the Native Beast of the JVM
Design Approach

- Threat Model
- Direct Access
- Abusive JNI Calls
- System Calls
- Robusta
  - Sandboxing
  - JNI Checking
  - JSM Integration
Threat #1: Direct Access

Java Virtual Machine (JVM)

Why?

Approach

Challenges

Results

Moving Forward

Robusta:
Taming the Native Beast of the JVM
Threat #1: Direct Access

- Corrupt Objects, Overwrite Code, Etc...
- Disable security checks, Crash the JVM, Etc...

Native Area

Java Area

JVM State

JVM

Robusta: Taming the Native Beast of the JVM
Threat #2: Abusive JNI Calls

Java Native Interface

Native Area

Java Area

JVM State

JVM

Why?
Approach
Challenges
Results
Moving Forward

Robusta:
Taming the Native Beast of the JVM
Threat #2: Abusive JNI Calls

Confidentiality:

Java

public class A {
    private int a;
}

C++

int a = getIntField(A, field); //violation
Threat #2: Abusive JNI Calls

Integrity:

Java

```java
public class A {
    public B b;
}
```

C++

```c++
setObjectField(A, field, c_ref); //type confusion
```
Threat #3: System Calls

Java Native Interface

Native Area

Java Area

JVM State

System Calls

JVM

Why?

Approach

Challenges

Results

Moving Forward

Robusta:

Taming the Native Beast of the JVM
Threat #3: System Calls

Java Native Interface

Native Area

Java Area

JVM State

System Calls

Write to file system
Access network
Etc...

Robusta:
Taming the Native Beast of the JVM
Design Approach

- Threat Model
- Direct Access
- Abusive JNI Calls
- System Calls

Robusta
- Sandboxing
- JNI Checking
- JSM Integration
Preventing Direct Access

Robusta: Taming the Native Beast of the JVM
Sandboxing (via NaCl) in Robusta

Software Fault Isolation (SFI)

- Sandbox
- Trampolines
- Data Region (Not executable)
- Code Region (Not writeable)

- Tool Chain (naclgcc, ... etc)
- Code Verifier

Robusta:
Taming the Native Beast of the JVM

Why?
Approach
Challenges
Results
Moving Forward
Sandboxing (via NaCl) in Robusta

- Trampolines
- Data Region (Not executable)
- Code Region (Not writeable)

All exits to known points
All R/W remain in DR
All jumps remain in CR

Google code nativeclient
Native code for web apps

Robusta: Taming the Native Beast of the JVM
Regulating JNI Calls

Java Native Interface

Native Area

Robusta

Java Area

JVM State

JVM

Robusta:
Taming the Native Beast of the JVM
Regulating JNI Calls

Trampolines

SetObjectField()
GetObjectField()
(...)

Data Region

Code Region

NaCl Sandbox

JNI Handlers

SetObjectField()
GetObjectField()
(...)

JNI Functions

Safety Checking!
(a la SafeJNI and JINN)

Robusta:
Taming the Native Beast of the JVM
Confidentiality:

Java

```java
public class A {
    private int a;
}
```

C++

```cpp
int a = getIntField(A, field); //violation
```
Regulating JNI Calls

Integrity:

Java

```java
public class A {
    public B b;
}
```

C++

```c++
setObjectField(A, field, c_ref); //type confusion
```
Regulating System Calls

Robusta: Taming the Native Beast of the JVM

Why?
Approach
Challenges
Results
Moving Forward

Java Native Interface
Native Area
Security Manager (JSA)
System Calls
Java Area
JVM State
JVM

Why?
Approach
Challenges
Results
Moving Forward

Robusta: Taming the Native Beast of the JVM

22
Regulating System Calls

NaCl Sandbox

Trampolines
- Open()
- Close()
- (...)

Data Region

Code Region

Syscall Handlers
- Open()
- Close()
- (...)

System Calls

Policy Checks!

Robusta:
Taming the Native Beast of the JVM

Why?
Approach
Challenges
Results
Moving Forward
Regulating System Calls

Enabling the JSM:
```
java -Djava.security.manager \n    -Djava.security.policy=some_policy_file HelloWorldApp
```

File Access Configuration:
```
grant {
    permission java.io.FilePermission "/path/to/file/A", "write";
    permission java.io.FilePermission "/path/to/file/B", "read, write";
    permission java.io.FilePermission "/path/to/file/C", "read, write, delete";
}
```
Challenges

- Dynamic Linking (DL)
  - New Address Space
  - Security Risks of DL
  - Robusta’s Secure DL Strategy
Dynamic Linking (DL)

Not supported by NaCl!

A ace = new A();
    System.loadLibrary();
...
...
C car = new C();
    System.loadLibrary();
...

Java Package

Native Libraries

Robusta:
Taming the Native Beast of the JVM
Sandbox Address Space for DL*

<table>
<thead>
<tr>
<th>Code Region (CS)</th>
<th>Data Region (DS)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Unmapped Page (64k)</td>
<td>Unmapped (64k)</td>
</tr>
<tr>
<td>Trampolines (64k)</td>
<td>Unmapped (64K)</td>
</tr>
<tr>
<td>ld.so code (LC)</td>
<td>Unmapped (LC)</td>
</tr>
<tr>
<td>Unmapped (LD)</td>
<td>ld.so data (LD)</td>
</tr>
<tr>
<td>Library Code (C0)</td>
<td>Unmapped (C0)</td>
</tr>
<tr>
<td>Unmapped (D0)</td>
<td>Library Data (D0)</td>
</tr>
<tr>
<td>Library Code (C1)</td>
<td>Unmapped (C1)</td>
</tr>
<tr>
<td>Unmapped (D1)</td>
<td>Library Data (D1)</td>
</tr>
<tr>
<td>Unmapped</td>
<td>Unmapped</td>
</tr>
</tbody>
</table>

Why?  Approach  Challenges  Results  Moving Forward

Robusta: Taming the Native Beast of the JVM

*Mark Seaborn
Safety Concerns for DL

Preventing Execution During Loading

Robusta:
Taming the Native Beast of the JVM
Safety Concerns for DL

Preventing Execution During Loading

Ensuring Verification Before Execution
Safety Concerns for DL

- Preventing Execution During Loading
- Ensuring Verification Before Execution
- Blocking Changes After Verification
Robusta’s Secure DL Strategy

(Trusted) JVM: dlopen

(Sandbox) ld.so: mmap

Novel NX-bit Solution!

Native Library (.so)

Sandbox

Code Region

Read  Write  NX

Page Permissions

Why?  Approach  Challenges  Results  Moving Forward
Robusta’s Secure DL Strategy

(Trusted) JVM: dlopen

(Sandbox) ld.so: mmap

(Trusted) mmap handler

Native Library (.so)

Sandbox

Page Permissions

Why? Approach Challenges Results Moving Forward

Robusta:
Taming the Native Beast of the JVM
Robusta’s Secure DL Strategy

Why?

Approach

Challenges

Results

Moving Forward

Robusta:
Taming the Native Beast of the JVM

(Trusted) JVM: dlopen

(Sandbox) ld.so: mmap

(Trusted) mmap handler

Copy into R

Native Library (.so)

Sandbox

Code Region

Page Permissions

Read
Write
NX

Read
Write
NX

Robusta:
Taming the Native Beast of the JVM

33
Robusta’s Secure DL Strategy

(Trusted) JVM: dlopen

(Sandbox) ld.so: mmap

(Trusted) mmap handler

Verify R

Copy into R

Native Library (.so)

Sandbox

Code Region

Page Permissions

Read  Write  NX

Why?  Approach  Challenges  Results  Moving Forward

Robusta:
Taming the Native Beast of the JVM
Robusta’s Secure DL Strategy

(Trusted) JVM: dlopen

(Sandbox) ld.so: mmap

(Trusted) mmap handler

R Exec Ready

Verify R

Copy into R

Native Library (.so)

Sandbox

Code Region

Page Permissions

Read  Write  NX

Why?  Approach  Challenges  Results  Moving Forward
Results

- Runtime Cost Overview
- Measurement of Overhead
- Analysis of Sample Programs
What is the cost of using Robusta?

- **Context Switch Cost**
  - Passing Params, Spilling Registers
  - Running JNI Checks
  - Consulting Security Manager

- **SFI Cost**
  - Block Alignment
  - Jump Masking
Example Programs

- java.util.zip
- Elliptic Curve Keygen
- BMP to JPEG Conversion
- PDF Creation
- StrictMath

Robusta:
Taming the Native Beast of the JVM
Overhead Measurement

\( X = \text{Runtime on OpenJDK 1.7.0 + Robusta 1.0} \)

\( Y = \text{Runtime on OpenJDK 1.7.0} \)

\[ \% \text{ increase} = 100 \times \frac{(X - Y)}{Y} \]
Runtime Overhead % for JPEG, Elliptic Curve (EC), ZIP

Context Switches Per Millisecond

Robusta:
Taming the Native Beast of the JVM
Runtime Overhead % for JPEG, Elliptic Curve (EC), ZIP

Context Switches Per Millisecond

NaCl Estimated Cost

Additional Robusta Cost

Robusta: Taming the Native Beast of the JVM
Runtime Overhead % for PDF and StrictMath

Context Switches Per Millisecond

StrictMath

JPEG
EC
ZIP
PDF

OpenJDK 1.7.0, Robusta 1.0, Ubuntu 8.10

Robusta:
Taming the Native Beast of the JVM
Runtime Overhead % for PDF and StrictMath

Context Switches Per Millisecond

Runtime Overhead %

Robusta:
Taming the Native Beast of the JVM
Moving Forward

- Related Work
- Future Work
- Conclusion
- Questions
Related Work

- Klinkoff *et al.*, Extending .NET security to unmanaged code
- Tan *et al.*, Safe JNI; Lee *et al.*, Jinn
- Hirzel and Grimm, Jeannie
- Jim *et al.*, Cyclone
- Provos, Improving host security with system call policies
- Mettler *et al.*, Joe-E: A security-oriented subset of Java

AND MANY MORE...
Future Work

- Portability
- Context switch optimizations
- Sandboxing more of the JDK native libraries
- Library rewriting tool for NaCl
Conclusion

-Native code is the *dark corner* of Java security

-Robusta uses:
  - SFI from NaCl to isolate native code
  - JNI checks similar to SafeJNI and Jinn
  - JSM to regulate system calls from native code
  - a new dynamic linking architecture

- Working Robusta prototype in OpenJDK 1.7.0
- Few context switches = good performance
Questions?

Joseph Siefers  
Lehigh University  
joseph.siefers@gmail.com

Gang Tan  
Lehigh University  
gtan@cse.lehigh.edu

Greg Morrisett  
Harvard University  
greg@eecs.harvard.edu