Practical Implications of Java/JVM/JRE

Li Gong
lgong@mozilla.com

Security Seminar Series
Computer Lab, Cambridge, UK
May 04, 2011

Disclaimers via Old Quotes

- Theorem -- “Any problem in computer science can be solved with another level of indirection” [David Wheeler/Butler Lampson]
- Corollary -- “There is nothing new in computer science after 1970s” (all just rehash of old problems in new settings) [Lampson?]
- Nevertheless, old tricks applied in different environments can have new practical impacts

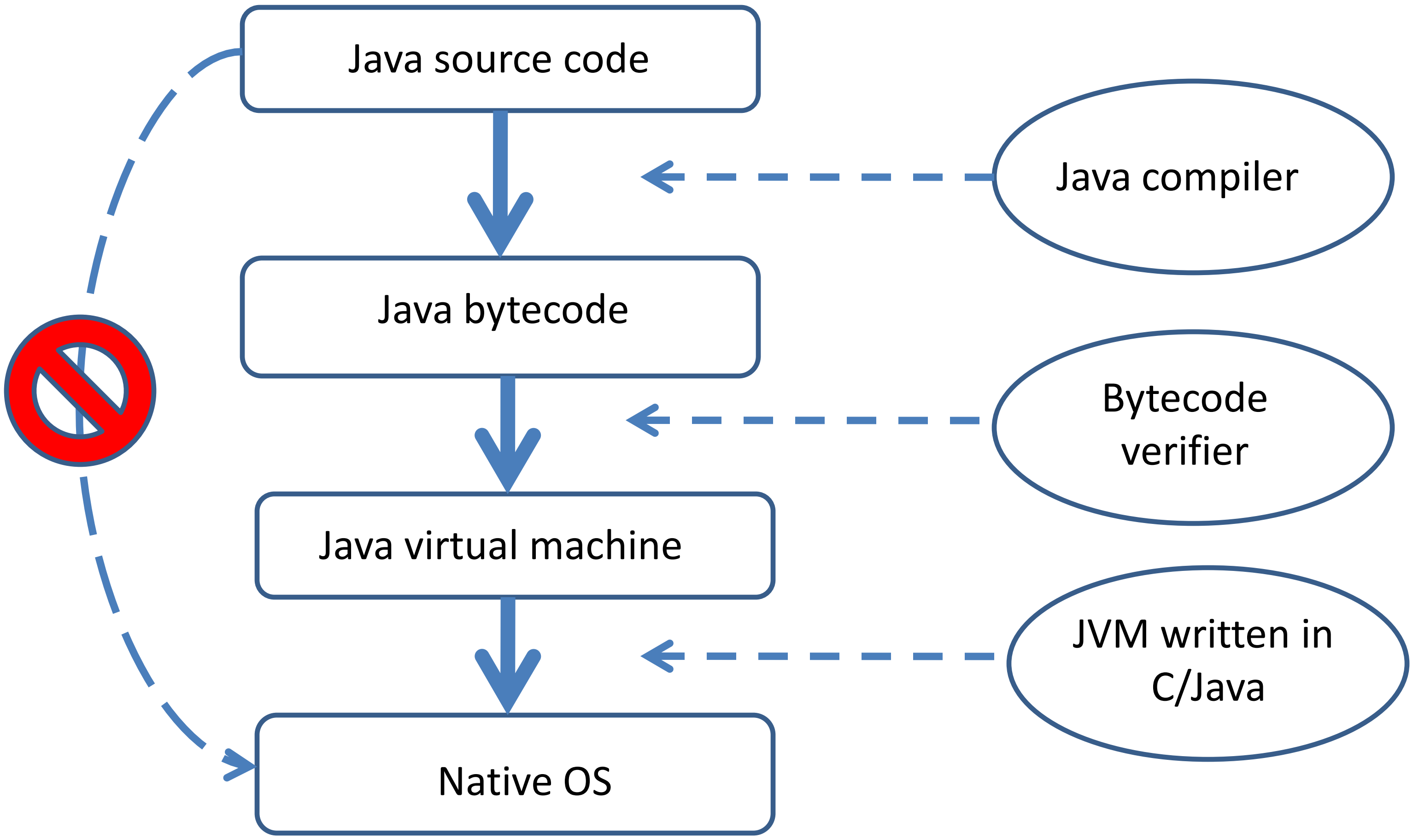
Who Do We (Secure Systems Builders) Work For?

- Programmers/application developers
  - “Users” do not directly use the OS
- So the key objective is to help the developer get what is intended with his/her code
  - Make the most common cases the easiest to write
  - Reduce risks of badly written code
- Major assumption
  - The system “we” produce has correct behaviors

Four Major Concerns for JDK 1.2 (as written in late 1996)

- Usability
  - Suitable for a wide variety of applications
- Simplicity
  - Easy to understand and analyze
- Adequacy
  - Enough features before the next release
- Adaptability
  - Do not over prescribe
  - Can evolve with ease

How Java Code Is Run/Executed



Java source code

Java compiler

Java bytecode

Bytecode verifier

Java virtual machine

JVM written in C/Java

Native OS

How Java Code Is Run/Executed

- Java source code is compiled into Java bytecode
- Bytecode is fed into and interpreted by JVM/JRE
- Design objectives
  - Only valid bytecode is run
  - Only intended consequences occur
    - Good intended behaviors are ensured by usual testing
    - Bad unintended behaviors must be prevented
- JVM/JRE itself written in part in Java

How Java Code Is Run/Executed

- Java source code is compiled into Java bytecode
  - How do we know the source is valid Java code?
  - A correct compiler accepts valid Java source code and produces valid Java bytecode
  - Can we trust the compiler someone else uses? No?
- Bytecode is fed into and interpreted by JVM/JRE
  - How do we ensure that we accept only valid bytecode?

It’s an Input Validation Problem

- F(n), n = 0, 1, 2, 3, 4, 5, 6, 7, 8, 9
  - N is completely/well structured
  - N has a small space
  - Input validation is trivial
- Java bytecode
  - Not completely/well structured
  - Has a large space
  - Consider an extreme example H(x) where x is 128 bit arbitrary number and H() is a one-way hash function that produces 256 bit hash values. Given any y, is y valid hash?
- Java is dynamically extensible
  - Type safety problem (think of buffer overflows)

Ensuring Bytecode Validality

- Static bytecode verifier
- Runtime type checking
- Have we covered all cases?
  - UW bytecode basher by Brian Bershad, Gun Sirer
- Can we type check sufficiently fast during run time?
  - Acceptable in the absolute
  - Acceptable in the relative

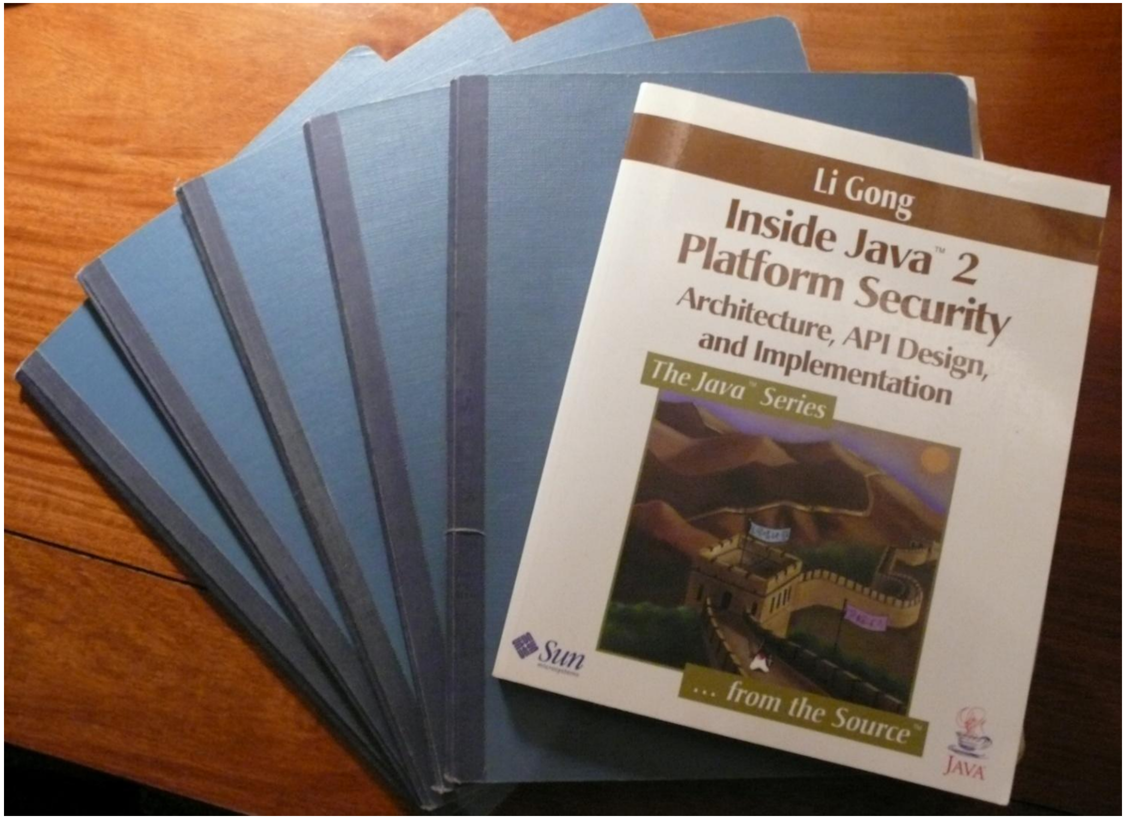
Preventing Bad Unintended Consequences

- Least-privilege principle
  - Associate objects with protection domains, each with its own set of privileges
  - Calculate dynamically “active privileges” (or if a specific privilege is active)
- Internal representation of privileges
  - java.security.permission classes, generic, extensible
  - The “implies” method
- External declarations of privileges
  - Policy specification (not intended as the only solution)
- Note: no requirement for MLS, info flow, etc.

Critical Issues of Least Privilege

- Can privilege calculations be done sufficiently fast?
  - Typical environments have simple permissions
  - Can be punted away – write your own algorithm
- Protection domains retrofitted into JVM/JRE
  - JIT cannot combine frames from different domains and other complications
  - Protection domains related to class/type extensions*
- Special privileged operations
  - Programmers must declare these explicitly

Get the Book and/or Read the Docs


Li Gong
Inside Java™ 2 Platform Security
Architecture, API Design, and Implementation
The Java™ Series
Sun
... from the Source™
JAVA

JDK 1.2 Security Feature List (12/11/1996)

- Project code named Gibraltar
- Features
  - Authentication
  - Delegation
  - Fine-grained access control
  - Policy management
  - Audit
  - Secret sharing
  - Key generation
  - Storage of private keys (e.g., passwords)
- Alpha (05/1997), FCS (09/1997)

Other Considerations Circca 1996/7

- Export control of crypto packages
  - Key escrow/key recovery, RSA/Bsafe/Cylink/others, CDSA, MS CAPI
  - “Church of Cryptology”
- Where is Java security headed
  - Is it just a component of the browser? More specifically the Netscape browser?

Other Considerations (Cont.)

- Protect against decompilation of Java bytecode
  - Code obfuscation
  - Encrypted bytecode
- Control of resource consumption by applets
- Java on a smartcard
- Java as e-commerce platform (Java Wallet)
- JavaOS (Java Station)
  - Security needs for a standalone OS?
- Sun company wide security architecture and strategy?

So Where Is the Drama?

- The whole project is equally a social (and political) process, not just a tech project
- Stressful – 1000~ meetings in 30 months, 300 pages of meeting notes
- Fast moving -- be ready to take the single available shot
- Constant onslaught of security bugs
  - The Friday fire drills
  - Microsoft was a Java licensee; but was it a good partner?
- There were people who wanted to “kill” it
  - Sun internal (delete our workspace, override security code, resist changes to the VM, resist security audit)
  - Fringes inside IBM (and other places)
  - Netscape fight (more later)

Technical Lessons Learned

- Systematic is better and easier than ad hoc
  - Implementing least privilege in JDK 1.2 turned out to be easier and more robust than a “bolted-on” binary sandbox model in JDK 1.0/1.1
- Do not use NULL
  - you cannot later change the behavior of a NULL (Null ClassLoader, Null SecurityManager)
- Do not overload functions
  - finding a class (which should be easily extensible) and defining it (which should be tightly controlled)

Is Java Fail Safe?

- Java cannot guarantee sequential execution, due to exceptions handling, even with Catch and Finally
- What happens when machine run out of memory? What’s the defined behavior then?

Alternative Ideas

- Erlingsson and Schneider, Inlined Reference Monitor (IRM)
  - Why interesting: support for arbitrary enforceable policy
  - Why not in: too late in the JDK 1.2 cycle to be fully evaluated
- Balfanz and Gong, multi-processing
  - Why: support for different security policies and properties for different processes
  - Why not in: too radical departure from JDK, too disruptive to existing code, not backward compatible

GuardedObject

- An object containing a resource (e.g., a file) and a specific guard (a permission)
  - The resource is accessible only if the permission is allowed
- Access permission is checked at the point of resource consumption, ensuring the right check is done in the right context
  - Can pass objects (references) around freely
  - Can prepare resources before actual requests
  - developers do not need to know about security managers or access control checks
- This is “slipped” into JDK, but not used internally, because it is alien to the familiar usage of invoking SecurityManager

Observations – The Good (the practical impacts)

- Java security has matured
  - From “what it is” to “how to utilize the features”
  - Did too little, too much, or just right?
- Raised the bar for everyone else
  - Anyone designing a new language/platform must consider type safety, systems security, least privilege, etc.
- Impacted thousands of programmers on their security awareness

Observations – The Bad

- Those companies who can afford the time and effort to improve security do not feel incented to spend the/adequate resources
- Those who want to differentiate from the dominate players cannot afford the time and effort
- When rarely a good/better security platform emerges, competition would not allow it to be adopted across the industry

Observations – The Bad (cont.)

- Many/any extensible systems (e.g., browser add-ons, iPhone apps) need the same sort of protection/security infrastructure, but they tend to be built on different technology platforms, so reuse is difficult or impossible

Observations – The Ugly

- A new thing (a toy widget, scripting language, etc.) starts nice and small, with limited usage scope and no security considerations
- It gains good traction
- The feature set keeps expanding and the toy becomes a widely adopted
- Soon the “small toy” resembles a full system or programming platform, except without adequate security support

12-Month Battle with Netscape

- The three battles
  - JFC vs Netscape’s IFC (combined into Swing)
  - Hotspot vs Netscape’s proposed Java VM
  - Java security vs Netscape Java security extensions
- IBM as arbitrator
  - Don Neal overall IBM taskforce lead (Bob Blakely took over the lead 3 months later)
  - Arbitration resolution meeting 10/15/2007

"Never Forget Class Struggle!"

- Email me at lgong@mozilla.com