JavaOne℠

Sun's Worldwide Java Developer Conference

Cryptography and Security

Benjamin Renaud
Marianne Mueller

Overview

- Security for the Java Platform
- Security Futures
  - Digitally Signed Code
  - Security API
- Secure Java Apps
  - Understanding Java Security
  - Example: Jeeves, a Java HTTP Server

Secure Java Platform

- Java as a complete platform
  - for the Internet
  - for your applications
- Key design principles
  - Open
  - Simple
  - Complete

Openness

- Public Specifications
- Source Code Available
- Interoperability and Standards

Simplicity

- Clean design
- Ease of development for
  - custom security
    - Security Manager
  - complex security
    - Key Management
    - Digital Signatures
    - Encryption

Completeness

- Built-in language security
  - Safe language
  - Extensive security information available
  - Flexible security models
- Security API
  - Comprehensive library for security-related functionality

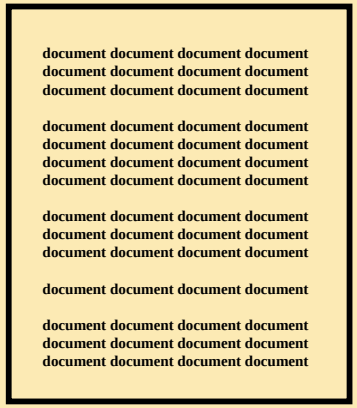
Security Futures

- Digitally signed code
  - Trust and partial trust
- Security API
  - To write secure applications

Digital Signatures

Private Key

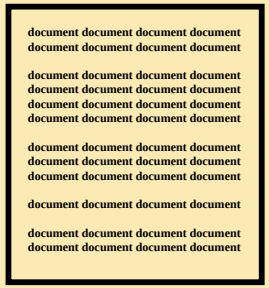
+



=

Public Key

+



=

OK?

The Meaning of Signatures

"The cardholder aknowledges receipt of the goods and/or services in the amount of the Total shown hereon and agrees to perform the obligations set forth in the Cardholder's agreement with the issuer."

The Meaning of Trust

- What does trust mean?
  - Complete freedom, supervised freedom
- How to decide trust?
  - Authorship, endorsement, rating etc.
- Digital signatures express assertions
  - Labeling systems refine assertions

The Simple Example

- “I fully trust software signed (published) by DoomSoft, Inc.”
- The code itself is signed by AcmeSoft, Inc.
- The code is allowed to do anything
- This is the shrink-wrap model of trust

Assertions

- “This code is published by DoomSoft, Inc. It comes with no guarantees.”
- “This code has been found to be free of viruses by UL.”
- “This code was rated five stars by PCWeek of 1/1/96.”
- “James thinks this code is good.”

Capabilities

- Can read all files in /opt/doomsoft
- Can connect to all IP addresses except 128.152.*.*
- Can use my spare CPU cycles
- Can do anything

Policies

Policy = Assertions + Capabilities

- Code published by DoomSoft can read files in /opt/doomsoft/
- Code certified by UL can connect to all IP addresses except 128.152.*.*
- Code that James thinks is good can do anything

Nuts and Bolts

- Simple model at first, but...
- Signing Java Archives (JAR)
  - Signatures of classes, images, sounds, etc.
- Support for
  - Multiple Schemes (DSA, RSA,…)
  - Multiple Signatures

Nuts and Bolts…

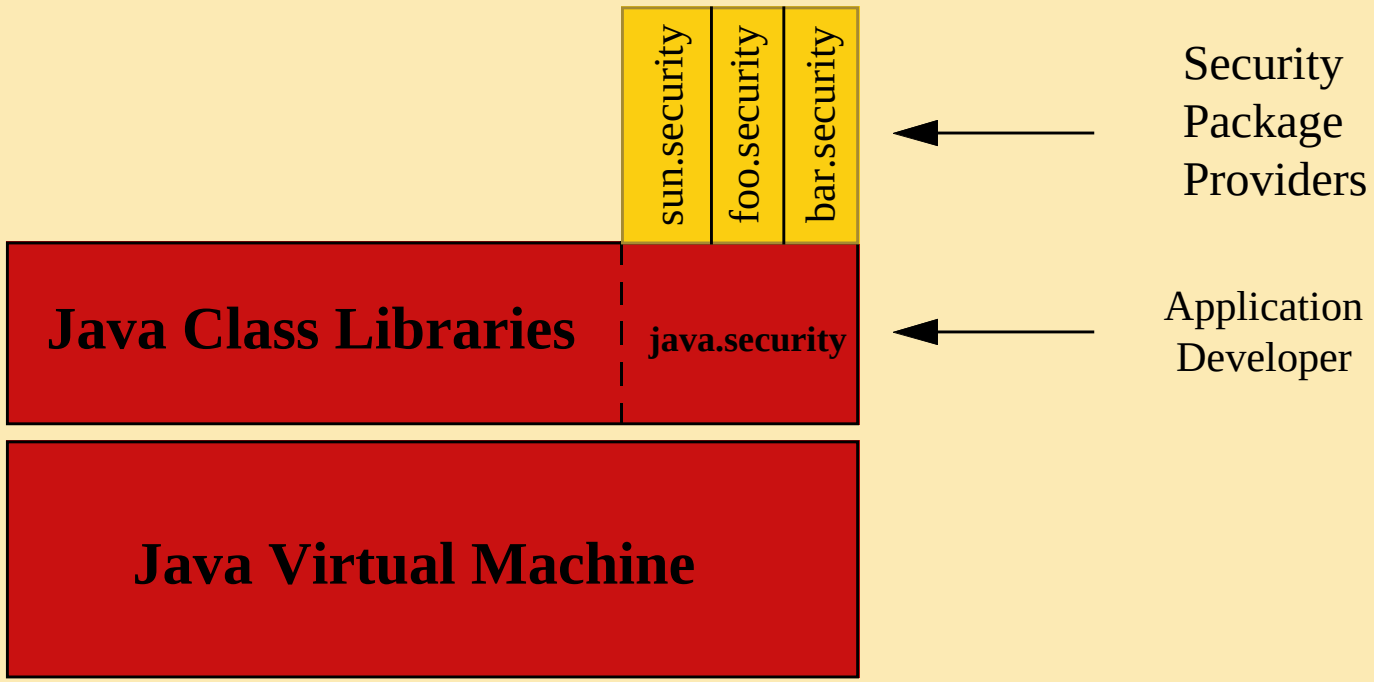
- `javakey` to handle signing and key management
- Based on Security API
  - Key management
  - Signature code

Security API

An API for Network-Centric Security

- Uniform interface to security services
  - Digital Signature
  - Encryption
  - Key Exchange
  - Utilities (Hash, PRNG, Bignums, etc.)
- System support for critical functions (e.g. key management)

Security API Architecture



Security Package Providers

- Developers do not call into Security Packages directly
- Flexibility and pluggable security
  - adding better implementations
  - adding better algorithms
- Security Packages must be signed

Key Management

- System key management
  - Secure storage
  - Extensible key information
- Public and private keys
  - Indexed by entity and algorithm
  - Not handled by applications directly
- Session keys
  - Can be persistent

Digital Signature

```
Signature dsa = new Signature("dsa");
// joe is an Entity object from keydb
dsa.initialize(joe);
byte[] sigBytes = dsa.sign(document);
```

or

```
boolean valid = dsa.verify(document,
  sigBytes);
```

`document` and `sigBytes` are `byte[]`.

Encryption

```
SymmetricCipher des = new
  SymmetricCipher("DES");
// say we generate a random session key
byte[] sessionKey = des.initialize(new
  CryptoRandom());
byte[] ciphertext =
  des.encrypt(document);
```

or

```
byte[] cleartext =
  des.decrypt(document);
```

document, cyphertext and cleartext are byte[].

Export Issues

- Security Packages must be signed
- Policy for signing is public and open
- Exportable API
- Exportable applications

Release and Schedule

- Key management, digital signatures and encryption first
- Secure channels and key exchange to follow

For more information

http://www.javasoft.com/

security-api@javasoft.com

JavaOne℠

Sun's Worldwide Java Developer Conference

Playing in the Sandbox

Marianne Mueller, JavaSoft

Overview

- Sandbox model
- Sandbox implementation

- Writing a security manager
- Extending the sandbox

Sandbox model

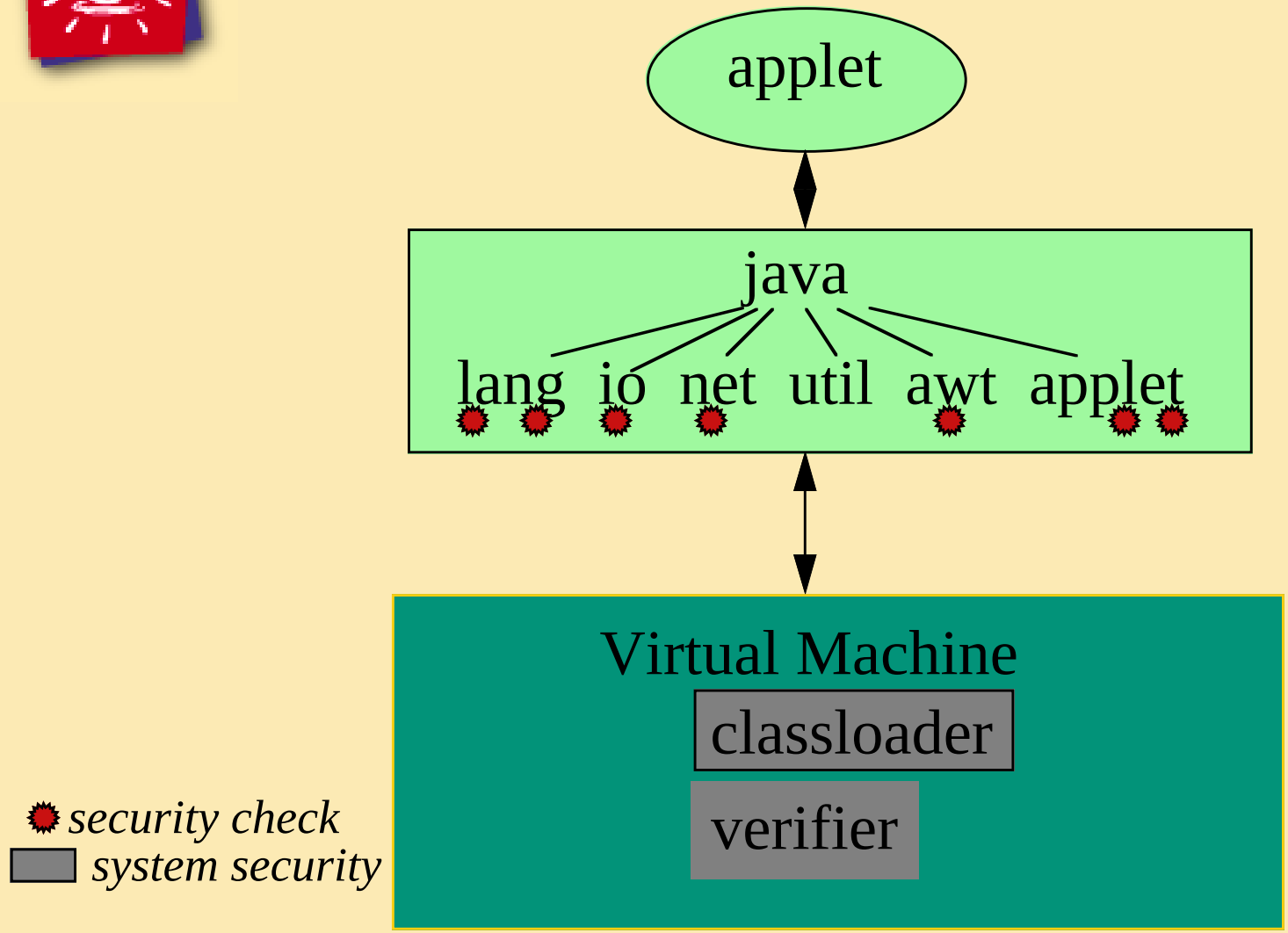
Application policy defines sandbox borders

Application implements border checks

Sandbox model: HotJava

- Applets barred from client file system
- Applets can only phone home
- Applets cannot load libraries
- Applets cannot exec processes
- Applets cannot examine properties

Sandbox implementation



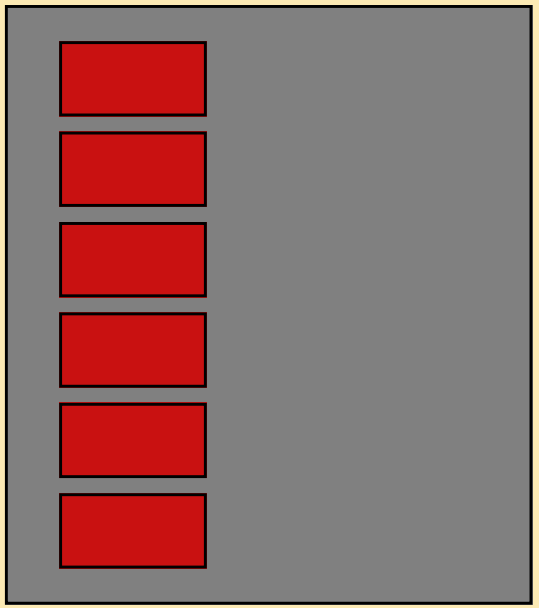
Sandbox implementation

- Security checks
- AppletSecurity.java

```
SecurityManager s;
s = System.getSecurityManager();
if (s != null) {
  s.checkConnect(host, port);
}
```

```
s.checkRead(filename);
```

```
s.checkExec(command);
```



~50 ✹ ‘s

~15 ▭ ‘s

Sandbox implementation: language features

- Classes declare and implement types
- Strong typing
- Access modifiers
- Memory management
- Misc: arrays, strings, no preprocessor, no `#define private public`, no `goto`

Sandbox implementation: classloader

- Classloader enforces namespaces
- Policy: applets can’t create classloader
- Classloader invokes verifier

Sandbox implementation: verifier

- Invoked on downloaded classes
- `java -verify`
- Verifier has 4 passes
  - Classfile verification
  - Type system verification
  - Bytecode verification
  - (runtime) Type and access checking

Sandbox implementation: fixes

- DNS name resolution
- Verifier (/absolute/path)
- Classloader (exception, private)

- Hostile applets - working on hooks to monitor and kill wayward applets

Getting beyond penetrate & patch

- Model VM, language, policy
- Verify against specification
- Security assessment
  - External software integrity review
  - Internet community's scrutiny of source code
  - Internal scrutiny
- Security Compatibility Test Suite
- Implementor's Guidelines

Writing a security manager

- Decide policy
- Minimize code for enforcing policy
- Insert checks
- Subclass java.lang.SecurityManager
- Publish policy and tests

Jeeves: Servlet Security

- Jeeves servlets
- Local servlets
- Network servlet sandbox
- Signed servlets *and the tools that love them*
  - List of trusted signatures
  - Allow/don’t allow unsigned servlets
  - Allow/don’t allow signed servlets
  - Parameterized attributes

Jeeves: Security Manager

- Servlet sandbox
  - Identify origin of servlet ==> Policy
  - Network unsigned servlets
  - Network signed servlets

Jeeves: unsigned servlets

- HTTP requests and responses
- Server's file system
- Server properties files
- Server dynamic configuration
- Servlet dynamic management
- Inter-servlet communication

Jeeves: signed servlets

- Open Policy
  - Treat signed servlets as file servlets
  - Full access
- Configurable Policy
  - Define short list of parameterized attributes
  - Allow administrator to grant/deny servlet requests for additional capability

Jeeves: signed servlets

- Configurable Policy

attribute	parameter	check
read(filename)	String	* Strings immutable * Access authorization
socket(host,port)	int, int	* DNS name resolution
any		

Try this at home

- Verifier model
- Type system model
- Capabilities system
- Experimental application policies

Where we are today

- JDK 1.0.2 (May 96) - fixes classloader, DNS, verifier bugs
- Jeeves alpha (summer 96) - signed servlets only, signed and unsigned in beta timeframe
- JDK 1.1 - signed applets, signed servlets