Programming Securely II

Computer Security Lectures 16 & 17

David Aspinall

School of Informatics
University of Edinburgh

2nd, 6th March 2006

Outline

Outline

Web security: client-side threats

- Risky treatment of **MIME-types**: e.g., shell-escapes in `troff`. By design downloaded **active content** (e.g., Java, ActiveX controls) should run in a restricted environment. Problems come when restrictions fail, or aren't tight enough.

Web security: client-side threats

- Risky treatment of **MIME-types**: e.g., shell-escapes in `troff`. By design downloaded **active content** (e.g., Java, ActiveX controls) should run in a restricted environment. Problems come when restrictions fail, or aren't tight enough.
- SSL issues: revoked certificates, spoofed site names, mixed encrypted/unencrypted pages.

Web security: client-side threats

- Risky treatment of **MIME-types**: e.g., shell-escapes in `troff`. By design downloaded **active content** (e.g., Java, ActiveX controls) should run in a restricted environment. Problems come when restrictions fail, or aren't tight enough.
- SSL issues: revoked certificates, spoofed site names, mixed encrypted/unencrypted pages.
- Browsers store **cookies** which have confidentiality implications. Even without cookies, web browsing is much less anonymous than it feels: information is stored in browser's history and document cache, various firewall and proxy logs, and each of the remote sites visited. This is even before any snitchware is present. (This is all fantastic for market researchers).

Web security: client-side threats

- Risky treatment of **MIME-types**: e.g., shell-escapes in `troff`. By design downloaded **active content** (e.g., Java, ActiveX controls) should run in a restricted environment. Problems come when restrictions fail, or aren't tight enough.
- SSL issues: revoked certificates, spoofed site names, mixed encrypted/unencrypted pages.
- Browsers store **cookies** which have confidentiality implications. Even without cookies, web browsing is much less anonymous than it feels: information is stored in browser's history and document cache, various firewall and proxy logs, and each of the remote sites visited. This is even before any snitchware is present. (This is all fantastic for market researchers).
- Untrained users may unwittingly make bad security decisions.

Web security: client-side threats

- Risky treatment of **MIME-types**: e.g., shell-escapes in `troff`. By design downloaded **active content** (e.g., Java, ActiveX controls) should run in a restricted environment. Problems come when restrictions fail, or aren't tight enough.
- SSL issues: revoked certificates, spoofed site names, mixed encrypted/unencrypted pages.
- Browsers store **cookies** which have confidentiality implications. Even without cookies, web browsing is much less anonymous than it feels: information is stored in browser's history and document cache, various firewall and proxy logs, and each of the remote sites visited. This is even before any snitchware is present. (This is all fantastic for market researchers).
- Untrained users may unwittingly make bad security decisions.
- Buggy browsers: buffer overflows, crypto bugs, etc.

Web security: server-side threats

- Access control: preventing certain files being served.

Web security: server-side threats

- Access control: preventing certain files being served.
- Complex or malicious URLs

Web security: server-side threats

- Access control: preventing certain files being served.
- Complex or malicious URLs
- Denial of service attacks

Web security: server-side threats

- Access control: preventing certain files being served.
- Complex or malicious URLs
- Denial of service attacks
- Remote authoring and administration tools

Web security: server-side threats

- Access control: preventing certain files being served.
- Complex or malicious URLs
- Denial of service attacks
- Remote authoring and administration tools
- Buggy servers, with attendant security risks

Web security: server-side threats

- Access control: preventing certain files being served.
- Complex or malicious URLs
- Denial of service attacks
- Remote authoring and administration tools
- Buggy servers, with attendant security risks
- Server-side **scripting languages**: C or shell CGI, PHP, ASP, JSP, Python, Ruby, all have serious security implications in configuration and execution. File systems and permissions have to be carefully designed. That's before any implemented *web application* is even considered...

Web programming: application security

Many issues (only some of which were introduced in the practical).

Web programming: application security

Many issues (only some of which were introduced in the practical).

- **Input validation**: to prevent SQL injection, command injection, other confidentiality attacks. AJAX: *beware client-side validation!*. Understand metacharacters at every point. Use labels/indexes for hidden values, not values themselves.

Web programming: application security

Many issues (only some of which were introduced in the practical).

- **Input validation**: to prevent SQL injection, command injection, other confidentiality attacks. AJAX: *beware client-side validation!*. Understand metacharacters at every point. Use labels/indexes for hidden values, not values themselves.
- **Output filtering**: cross-site scripting (XSS), when attacker-generated HTML appears on site: used for session hijacking, phishing attacks. Beware passing informative error messages.

Web programming: application security

Many issues (only some of which were introduced in the practical).

- **Input validation**: to prevent SQL injection, command injection, other confidentiality attacks. AJAX: *beware client-side validation!*. Understand metacharacters at every point. Use labels/indexes for hidden values, not values themselves.
- **Output filtering**: cross-site scripting (XSS), when attacker-generated HTML appears on site: used for session hijacking, phishing attacks. Beware passing informative error messages.
- **Careful cryptography**: encryption/hashing to *protect server state in client*, use of *appropriate authentication* mechanisms for web accounts (never Referer header).

Web programming: application security

Many issues (only some of which were introduced in the practical).

- **Input validation**: to prevent SQL injection, command injection, other confidentiality attacks. AJAX: *beware client-side validation!*. Understand metacharacters at every point. Use labels/indexes for hidden values, not values themselves.
- **Output filtering**: cross-site scripting (XSS), when attacker-generated HTML appears on site: used for session hijacking, phishing attacks. Beware passing informative error messages.
- **Careful cryptography**: encryption/hashing to *protect server state in client*, use of *appropriate authentication* mechanisms for web accounts (never Referer header).

Web programming: application security

Many issues (only some of which were introduced in the practical).

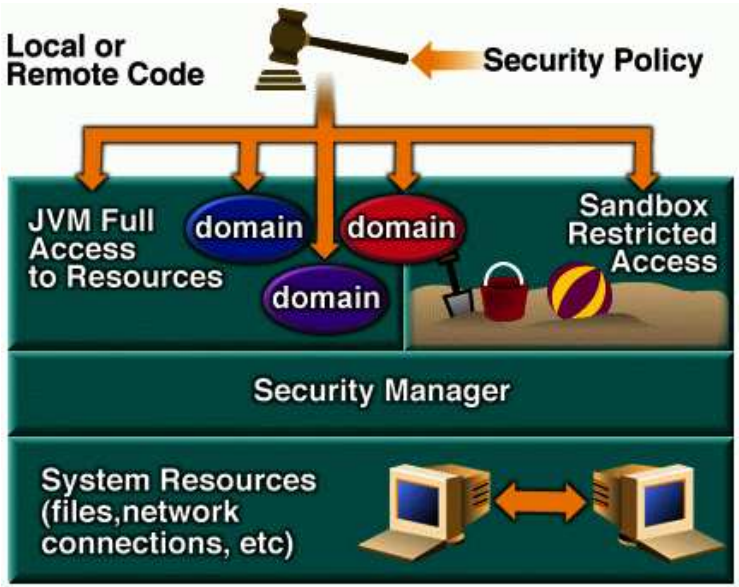
- **Input validation**: to prevent SQL injection, command injection, other confidentiality attacks. AJAX: *beware client-side validation!*. Understand metacharacters at every point. Use labels/indexes for hidden values, not values themselves.
- **Output filtering**: cross-site scripting (XSS), when attacker-generated HTML appears on site: used for session hijacking, phishing attacks. Beware passing informative error messages.
- **Careful cryptography**: encryption/hashing to *protect server state in client*, use of *appropriate authentication* mechanisms for web accounts (never Referer header).

Huseby's book *Innocent code — a security wake-up call for web programmers* is recommended.

Outline

Security in Java

Java 1.0 had a **sandbox** security model, where downloaded Java applets ran in a restricted environment with no access to local files, etc: often too restrictive. Java 2 has a more flexible, fine-grained level of control:

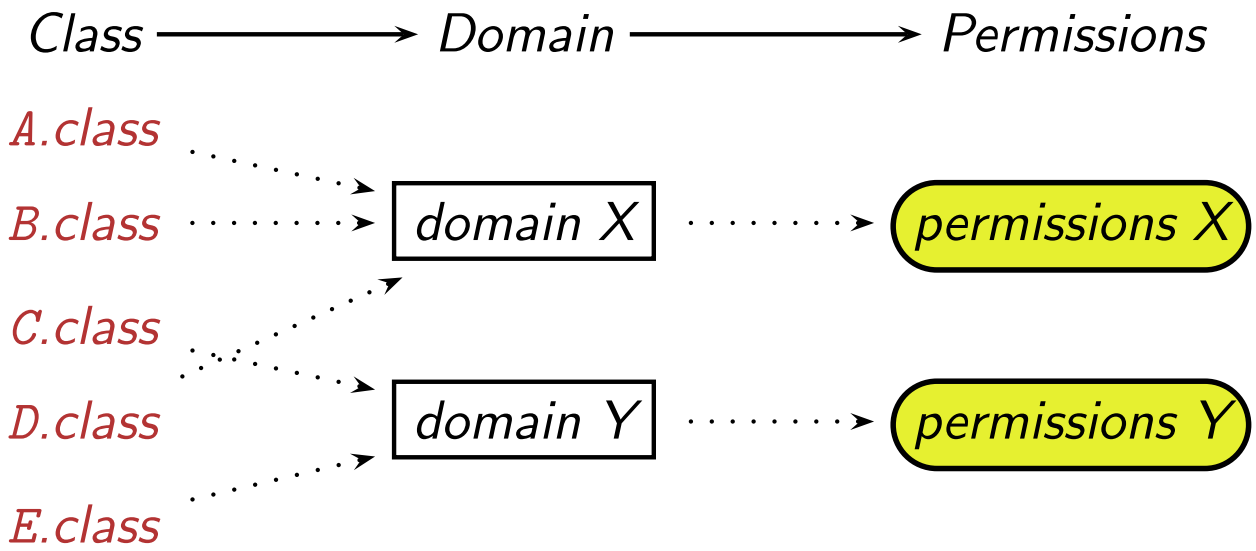


Applications and applets are subject to a **security policy** which specifies **protection domains** based on **location** of code, whether it is **signed** by a trusted entity, and the user **identity**. Each domain specifies a set of **permissions** for accessing resources.

This picture is reproduced from the Java Security Tutorial (c) Sun

Java security architecture

- A `SecurityManager` is installed by web browsers for Java applets; an application must either itself install the security manager, or be invoked with the option `-Djava.security.manager`. If the security manager's checks fail, a `java.lang.SecurityException` is raised.
- Access control in Java is based on **protection domains** which group together the set of objects which are currently accessible by a principal.



Java access control permissions

- Domains are associated with sets of **permissions**, such as:

Java access control permissions

- Domains are associated with sets of **permissions**, such as:

Java access control permissions

- Domains are associated with sets of **permissions**, such as:

java.security.AllPermission	every resource
java.io.FilePermission	file system access
java.net.SocketPermission	accept/connect based on host/IP & port
java.awt.AWTPermission	window-system permissions
java.lang.RuntimePermission	JVM configuring; printing; thread control
java.security.SecurityPermission	accessing security policy, key store

- Permissions may be associated with a **target** and some **actions**:

Java access control permissions

- Domains are associated with sets of **permissions**, such as:

java.security.AllPermission	every resource
java.io.FilePermission	file system access
java.net.SocketPermission	accept/connect based on host/IP & port
java.awt.AWTPermission	window-system permissions
java.lang.RuntimePermission	JVM configuring; printing; thread control
java.security.SecurityPermission	accessing security policy, key store

- Permissions may be associated with a **target** and some **actions**:

```
import java.io.FilePermission;
FilePermission p1 = new
FilePermission("/tmp/myfile", "read");
FilePermission p2 = new
FilePermission("/tmp/*", "read");
```

Java access control permissions

- Domains are associated with sets of **permissions**, such as:

java.security.AllPermission	every resource
java.io.FilePermission	file system access
java.net.SocketPermission	accept/connect based on host/IP & port
java.awt.AWTPermission	window-system permissions
java.lang.RuntimePermission	JVM configuring; printing; thread control
java.security.SecurityPermission	accessing security policy, key store

- Permissions may be associated with a **target** and some **actions**:

```
import java.io.FilePermission;
FilePermission p1 = new
FilePermission("/tmp/myfile", "read");
FilePermission p2 = new
FilePermission("/tmp/*", "read");
```

Java access control permissions

- Domains are associated with sets of **permissions**, such as:

java.security.AllPermission	every resource
java.io.FilePermission	file system access
java.net.SocketPermission	accept/connect based on host/IP & port
java.awt.AWTPermission	window-system permissions
java.lang.RuntimePermission	JVM configuring; printing; thread control
java.security.SecurityPermission	accessing security policy, key store

- Permissions may be associated with a **target** and some **actions**:

```
import java.io.FilePermission;
FilePermission p1 = new
FilePermission("/tmp/myfile", "read");
FilePermission p2 = new
FilePermission("/tmp/*", "read");
```

Permissions implement an **implies** method used to make access control decisions. Here `p2.implies(p1)`.

Java security policies

- The system security policy for a Java application environment specifies permissions available for code from various sources, represented by a `Policy` object. Only one in effect at a time.
- A `Policy` object evaluates the global policy using the `ProtectionDomain` for a class, and returns an appropriate `Permissions` object.
- Java supplies a GUI **policytool** utility for editing ASCII format policy files, with entries like this, specifying a key store and zero or more "grant" entries:

```
keystore ".keystore", "JKS";
grant principal com.sun.security.auth.UnixPrincipal "da" {
  permission java.util.PropertyPermission "java.home", "read";
  permission java.io.FilePermission "/tmp/foo", "read,write";
};
```

Default, system policy is in *javahome*/lib/security/java.policy. User policy is in *userhome*.java.policy.

Java security extensions

- The Java security extensions add additional APIs for programming security features. Previously add-ons; integrated since J2SDK v1.4.

Java security extensions

- The Java security extensions add additional APIs for programming security features. Previously add-ons; integrated since J2SDK v1.4.
- **Java Cryptography Extension** (JCE)
  A Java framework for cryptographic functionality, including message digests, encryption, signing, and X.509 certificates.

Java security extensions

- The Java security extensions add additional APIs for programming security features. Previously add-ons; integrated since J2SDK v1.4.
- **Java Cryptography Extension** (JCE)
  A Java framework for cryptographic functionality, including message digests, encryption, signing, and X.509 certificates.
- **Java Secure Socket Extension** (JSSE).

Java security extensions

- The Java security extensions add additional APIs for programming security features. Previously add-ons; integrated since J2SDK v1.4.
- **Java Cryptography Extension** (JCE)
  A Java framework for cryptographic functionality, including message digests, encryption, signing, and X.509 certificates.
- **Java Secure Socket Extension** (JSSE).
- **Java Authentication and Authorization Service** (JAAS)
  Used for "reliable and secure" authentication of users, to determine who is currently executing Java code; and for authorization of users to ensure they have the permissions necessary for desired actions.

Java security extensions

- The Java security extensions add additional APIs for programming security features. Previously add-ons; integrated since J2SDK v1.4.
- **Java Cryptography Extension** (JCE)
  A Java framework for cryptographic functionality, including message digests, encryption, signing, and X.509 certificates.
- **Java Secure Socket Extension** (JSSE).
- **Java Authentication and Authorization Service** (JAAS)
  Used for "reliable and secure" authentication of users, to determine who is currently executing Java code; and for authorization of users to ensure they have the permissions necessary for desired actions.
- **Java GSS-API**.
  Bindings for Generic Security Service API (RFC2853). Used for securely exchanging messages between communicating applications, using various underlying mechanisms (e.g., Kerberos).

Java Cryptography Extension (JCE)

- Crypto framework. A *provider* plug-in architecture allows multiple simultaneous implementations. Was split out of SDK because of export restrictions; since v1.4, JCE included but limited (import restrictions).

Java Cryptography Extension (JCE)

- Crypto framework. A *provider* plug-in architecture allows multiple simultaneous implementations. Was split out of SDK because of export restrictions; since v1.4, JCE included but limited (import restrictions).
- Has *algorithm independence*, clients don't need to understand algorithms; abstract "engine" classes provide different services.

Java Cryptography Extension (JCE)

- Crypto framework. A *provider* plug-in architecture allows multiple simultaneous implementations. Was split out of SDK because of export restrictions; since v1.4, JCE included but limited (import restrictions).
- Has *algorithm independence*, clients don't need to understand algorithms; abstract "engine" classes provide different services.
- *Service provider interfaces* (SPIs) added statically or dynamically; clients query installed providers to find out supported services. JVM and clients specify preference orders.

Java Cryptography Extension (JCE)

- Crypto framework. A *provider* plug-in architecture allows multiple simultaneous implementations. Was split out of SDK because of export restrictions; since v1.4, JCE included but limited (import restrictions).
- Has *algorithm independence*, clients don't need to understand algorithms; abstract "engine" classes provide different services.
- *Service provider interfaces* (SPIs) added statically or dynamically; clients query installed providers to find out supported services. JVM and clients specify preference orders.
- Key management is through a "keystore" database. Different providers may implement different keystore formats.

Java Cryptography Extension (JCE)

- Crypto framework. A *provider* plug-in architecture allows multiple simultaneous implementations. Was split out of SDK because of export restrictions; since v1.4, JCE included but limited (import restrictions).
- Has *algorithm independence*, clients don't need to understand algorithms; abstract "engine" classes provide different services.
- *Service provider interfaces* (SPIs) added statically or dynamically; clients query installed providers to find out supported services. JVM and clients specify preference orders.
- Key management is through a "keystore" database. Different providers may implement different keystore formats.
- SUN provider: DSA, MD5, SHA-1, SHA1PRNG, X.509 certificates, keystore implementation for proprietary keystore type JKS. Another: **Cryptix** `http://www.cryptix.org`.

Java Cryptography Extension (JCE)

- Crypto framework. A *provider* plug-in architecture allows multiple simultaneous implementations. Was split out of SDK because of export restrictions; since v1.4, JCE included but limited (import restrictions).
- Has *algorithm independence*, clients don't need to understand algorithms; abstract "engine" classes provide different services.
- *Service provider interfaces* (SPIs) added statically or dynamically; clients query installed providers to find out supported services. JVM and clients specify preference orders.
- Key management is through a "keystore" database. Different providers may implement different keystore formats.
- SUN provider: DSA, MD5, SHA-1, SHA1PRNG, X.509 certificates, keystore implementation for proprietary keystore type JKS. Another: **Cryptix** `http://www.cryptix.org`.
- See: `javax.crypto`, `javax.crypto.interfaces`, `javax.crypto.spec`.

JCE cryptography services

- A cryptography service is associated with a particular algorithm or type, and manipulates or generates data, keys, algorithm parameters, keystores, or certificates.

JCE cryptography services

- A cryptography service is associated with a particular algorithm or type, and manipulates or generates data, keys, algorithm parameters, keystores, or certificates.
- Engine classes include:

MessageDigest	generate message digests (MDCs)
Signature	sign data and verify digital signatures.
KeyPairGenerator	generate public-private key-pair.
CertificateFactory	create certificates and CRLs.
KeyStore	create and manage key databases.
AlgorithmParameters	manage parameters for an algorithm.
SecureRandom	random or pseudo-random numbers.

JCE cryptography services

- A cryptography service is associated with a particular algorithm or type, and manipulates or generates data, keys, algorithm parameters, keystores, or certificates.
- Engine classes include:

MessageDigest	generate message digests (MDCs)
Signature	sign data and verify digital signatures.
KeyPairGenerator	generate public-private key-pair.
CertificateFactory	create certificates and CRLs.
KeyStore	create and manage key databases.
AlgorithmParameters	manage parameters for an algorithm.
SecureRandom	random or pseudo-random numbers.

- Factory methods in engine classes are used to return instances of the class, e.g. `Signature.getInstance("SHA1withDSA")`.

Java Secure Socket Extension (JSSE)

- The JSSE is also based on a provider plug-in architecture.
- Has a simple structure. Main use is with SSL client sockets, SSL server sockets, and SSL session handles. Sample classes:

SSLSocket	socket for SSL/TLS/WTLS protocols
SSLSocketFactory	factory for SSLSocket objects
SSLServerSocket	sever socket for SSL/TLS/WTLS
$\cdots$ Factory	factory for SSLServerSockets
SSLSession	encapsulation of SSL session (an interface)

- Creating SSL client or server sockets is as easy as creating ordinary Java TCP/IP sockets: each SSL class extends the corresponding ordinary TCP socket class, and provides a few extra hooks for setting security parameters.
- See `javax.net.ssl`, also `javax.net` and `javax.security.cert`.

Authentication and Authorization (JAAS)

- JAAS has a pluggable architecture, so applications are independent of underlying authentication methods: implementation of authentication method is decided at runtime, in a **login configuration file**.

Authentication and Authorization (JAAS)

- JAAS has a pluggable architecture, so applications are independent of underlying authentication methods: implementation of authentication method is decided at runtime, in a **login configuration file**.
- A `Subject` may have multiple identities, each of which is a `Principal` (name). Subjects also own security-related public and private **credentials** (e.g., key material).

Authentication and Authorization (JAAS)

- JAAS has a pluggable architecture, so applications are independent of underlying authentication methods: implementation of authentication method is decided at runtime, in a **login configuration file**.
- A `Subject` may have multiple identities, each of which is a `Principal` (name). Subjects also own security-related public and private **credentials** (e.g., key material).
- To authenticate, a `LoginContext` object is created, which then consults a configuration to load the require `LoginModule`s. The `login` method is invoked for each module, to try to authenticate a subject.

Authentication and Authorization (JAAS)

- JAAS has a pluggable architecture, so applications are independent of underlying authentication methods: implementation of authentication method is decided at runtime, in a **login configuration file**.
- A `Subject` may have multiple identities, each of which is a `Principal` (name). Subjects also own security-related public and private **credentials** (e.g., key material).
- To authenticate, a `LoginContext` object is created, which then consults a configuration to load the require `LoginModule`s. The `login` method is invoked for each module, to try to authenticate a subject.
- **Authorization** happens when a subject is associated with a thread's `AccessControlContext` using the **doAs** methods for performing actions (`run` methods of a `java.security.PrivilegedAction` object). Then principal-based entries in the current security policy are used.

Java security flaws

- Java was touted from the start as a secure mechanism for mobile code. But it has suffered from flaws in both design and implementation, surveyed in 1999 by McGraw and Felten in *Securing Java*, see `http://www.securingjava.com`.

Java security flaws

- Java was touted from the start as a secure mechanism for mobile code. But it has suffered from flaws in both design and implementation, surveyed in 1999 by McGraw and Felten in *Securing Java*, see `http://www.securingjava.com`.
- Most fundamental are problems in the **Byte Code Verifier**, which checks proper use of JVML (protecting against "malicious" or merely buggy compilers):

Java security flaws

- Java was touted from the start as a secure mechanism for mobile code. But it has suffered from flaws in both design and implementation, surveyed in 1999 by McGraw and Felten in *Securing Java*, see `http://www.securingjava.com`.
- Most fundamental are problems in the **Byte Code Verifier**, which checks proper use of JVML (protecting against "malicious" or merely buggy compilers):

Java security flaws

- Java was touted from the start as a secure mechanism for mobile code. But it has suffered from flaws in both design and implementation, surveyed in 1999 by McGraw and Felten in *Securing Java*, see `http://www.securingjava.com`.
- Most fundamental are problems in the **Byte Code Verifier**, which checks proper use of JVML (protecting against "malicious" or merely buggy compilers):
  - no operand stack overflow/underflow

Java security flaws

- Java was touted from the start as a secure mechanism for mobile code. But it has suffered from flaws in both design and implementation, surveyed in 1999 by McGraw and Felten in *Securing Java*, see `http://www.securingjava.com`.
- Most fundamental are problems in the **Byte Code Verifier**, which checks proper use of JVML (protecting against "malicious" or merely buggy compilers):
  - no operand stack overflow/underflow
  - correct types and conversions

Java security flaws

- Java was touted from the start as a secure mechanism for mobile code. But it has suffered from flaws in both design and implementation, surveyed in 1999 by McGraw and Felten in *Securing Java*, see `http://www.securingjava.com`.
- Most fundamental are problems in the **Byte Code Verifier**, which checks proper use of JVML (protecting against "malicious" or merely buggy compilers):
  - no operand stack overflow/underflow
  - correct types and conversions
  - field accesses obey visibility modifiers

Java security flaws

- Java was touted from the start as a secure mechanism for mobile code. But it has suffered from flaws in both design and implementation, surveyed in 1999 by McGraw and Felten in *Securing Java*, see `http://www.securingjava.com`.
- Most fundamental are problems in the **Byte Code Verifier**, which checks proper use of JVML (protecting against "malicious" or merely buggy compilers):
  - no operand stack overflow/underflow
  - correct types and conversions
  - field accesses obey visibility modifiers

Java security flaws

- Java was touted from the start as a secure mechanism for mobile code. But it has suffered from flaws in both design and implementation, surveyed in 1999 by McGraw and Felten in *Securing Java*, see `http://www.securingjava.com`.
- Most fundamental are problems in the **Byte Code Verifier**, which checks proper use of JVML (protecting against "malicious" or merely buggy compilers):
  - no operand stack overflow/underflow
  - correct types and conversions
  - field accesses obey visibility modifiers

  Type safety relies on byte code verification being correct. Unfortunately getting this right is complicated...

Java security flaws – continued

- The Java Language Specification is written in English. It suffers from usual problems of large language specifications: missing details, ambiguity, and other inaccuracies.

Java security flaws – continued

- The Java Language Specification is written in English. It suffers from usual problems of large language specifications: missing details, ambiguity, and other inaccuracies.
  - Sun BUG ID 6360463 (Dec 05): "offset item of the stack map frame" not defined in specification ... "renders most of discussion on type checking moot"

Java security flaws – continued

- The Java Language Specification is written in English. It suffers from usual problems of large language specifications: missing details, ambiguity, and other inaccuracies.
  - Sun BUG ID 6360463 (Dec 05): "offset item of the stack map frame" not defined in specification ... "renders most of discussion on type checking moot"
- Sun's implementations are usually taken as the reference behaviour. But these have had a series of type safety and access control failings (from 1.x SDKs to J2ME in mobile phone KVMs).

Java security flaws – continued

- The Java Language Specification is written in English. It suffers from usual problems of large language specifications: missing details, ambiguity, and other inaccuracies.
  - Sun BUG ID 6360463 (Dec 05): "offset item of the stack map frame" not defined in specification ... "renders most of discussion on type checking moot"
- Sun's implementations are usually taken as the reference behaviour. But these have had a series of type safety and access control failings (from 1.x SDKs to J2ME in mobile phone KVMs).
  - 8th Feb 2006, CVE-2006-0614,0615,0617: Sun fixes seven vulnerabilities in current JREs which allowed remote code to bypass sandbox using reflection.

Java security flaws – continued

- The Java Language Specification is written in English. It suffers from usual problems of large language specifications: missing details, ambiguity, and other inaccuracies.
  - Sun BUG ID 6360463 (Dec 05): "offset item of the stack map frame" not defined in specification ... "renders most of discussion on type checking moot"
- Sun's implementations are usually taken as the reference behaviour. But these have had a series of type safety and access control failings (from 1.x SDKs to J2ME in mobile phone KVMs).
  - 8th Feb 2006, CVE-2006-0614,0615,0617: Sun fixes seven vulnerabilities in current JREs which allowed remote code to bypass sandbox using reflection.
- This serves to point out the importance of *defence in depth*; even if you have a carefully configured Java security policy restricting what arbitrary downloaded code can do, you should still beware *untrusted* code.

Outline

Palladium/TCPA/NGSCB/Trustworthy Computing

- Strong crypto built into PCs via a security chip with master keys. PC boots, hashing BIOS, OS and application code. Builds a chain of trust.

Palladium/TCPA/NGSCB/Trustworthy Computing

- Strong crypto built into PCs via a security chip with master keys. PC boots, hashing BIOS, OS and application code. Builds a chain of trust.
- Protection domains in OS are extended into hardware (secure keyboard reading, sound channels). Idea to turn open system into a closed architecture (cf XBox).

Palladium/TCPA/NGSCB/Trustworthy Computing

- Strong crypto built into PCs via a security chip with master keys. PC boots, hashing BIOS, OS and application code. Builds a chain of trust.
- Protection domains in OS are extended into hardware (secure keyboard reading, sound channels). Idea to turn open system into a closed architecture (cf XBox).
- Allows certificates, e.g. "this document was created with version 1751 of Microsoft Word, running on NGSCB version of Windows Vista, on 27th August 2006, on Dell Megaplex ZZ5 serial no. 5091237896". Files are stored encrypted and cannot be decrypted on other machines.

Palladium/TCPA/NGSCB/Trustworthy Computing

- Strong crypto built into PCs via a security chip with master keys. PC boots, hashing BIOS, OS and application code. Builds a chain of trust.
- Protection domains in OS are extended into hardware (secure keyboard reading, sound channels). Idea to turn open system into a closed architecture (cf XBox).
- Allows certificates, e.g. "this document was created with version 1751 of Microsoft Word, running on NGSCB version of Windows Vista, on 27th August 2006, on Dell Megaplex ZZ5 serial no. 5091237896". Files are stored encrypted and cannot be decrypted on other machines.
- Many uses. Strong anti-privacy measures. Business clients: financial services, government, and healthcare. Home PC users: reduction in spyware, digital rights management (DRM). New uses: renting, lending, time-limited, etc. Considerable controversy (Stallman: "Treacherous Computing").

Code signing: worth the bits?

- So far, we're trusting code based on authentication of its source (or maybe some security-verification agent). I'll trust updates to IE only if they're signed by Microsoft.

Code signing: worth the bits?

- So far, we're trusting code based on authentication of its source (or maybe some security-verification agent). I'll trust updates to IE only if they're signed by Microsoft.
- Code signing is better than trusting unauthenticated code: a digital version of "shrink-wrapping" software with holographic certificates. However, this trust is fallible:

Code signing: worth the bits?

- So far, we're trusting code based on authentication of its source (or maybe some security-verification agent). I'll trust updates to IE only if they're signed by Microsoft.
- Code signing is better than trusting unauthenticated code: a digital version of "shrink-wrapping" software with holographic certificates. However, this trust is fallible:
  1. Microsoft's signing scheme may be compromised (this has actually happened, by a infamous social engineering attack on Verisign),

Code signing: worth the bits?

- So far, we're trusting code based on authentication of its source (or maybe some security-verification agent). I'll trust updates to IE only if they're signed by Microsoft.
- Code signing is better than trusting unauthenticated code: a digital version of "shrink-wrapping" software with holographic certificates. However, this trust is fallible:
  1. Microsoft's signing scheme may be compromised (this has actually happened, by a infamous social engineering attack on Verisign),
  2. More seriously, the code might not be secure anyway, if Microsoft fails to program securely, or virus/corruption occurs before signing (also happened: MS accidently distributed Nimda virus with VS .NET!)

Code signing: worth the bits?

- So far, we're trusting code based on authentication of its source (or maybe some security-verification agent). I'll trust updates to IE only if they're signed by Microsoft.
- Code signing is better than trusting unauthenticated code: a digital version of "shrink-wrapping" software with holographic certificates. However, this trust is fallible:
  1. Microsoft's signing scheme may be compromised (this has actually happened, by a infamous social engineering attack on Verisign),
  2. More seriously, the code might not be secure anyway, if Microsoft fails to program securely, or virus/corruption occurs before signing (also happened: MS accidently distributed Nimda virus with VS .NET!)
- The problem is that we delegate trust to somebody else rather than examining the code for ourselves. Examining the code ourselves to **prove** that it is secure for us is a possible solution, but hardly feasible. Or is it?

Future: Proof-carrying code?

- Ideally, we would certify code not to its origin, but with a **self-evident guarantee of security**, to capture exactly what we want. The client would check that the guarantee matches the code, and that the guarantee meets local security policy, so is safe to execute.

Future: Proof-carrying code?

- Ideally, we would certify code not to its origin, but with a **self-evident guarantee of security**, to capture exactly what we want. The client would check that the guarantee matches the code, and that the guarantee meets local security policy, so is safe to execute.
- This perhaps sounds far-fetched, but is the subject of current research into **proof-carrying code** (PCC).

Future: Proof-carrying code?

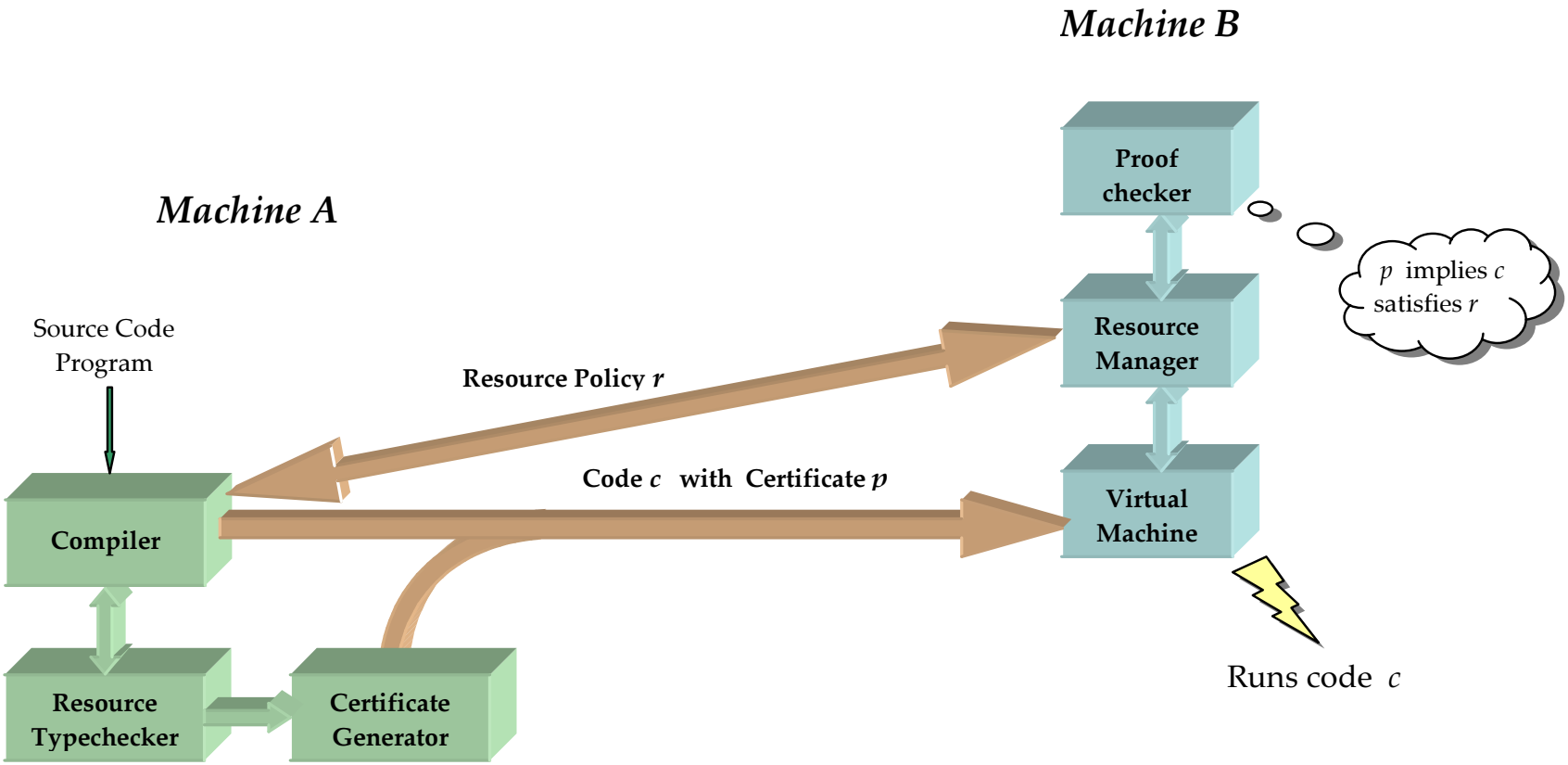
- Ideally, we would certify code not to its origin, but with a **self-evident guarantee of security**, to capture exactly what we want. The client would check that the guarantee matches the code, and that the guarantee meets local security policy, so is safe to execute.
- This perhaps sounds far-fetched, but is the subject of current research into **proof-carrying code** (PCC).
- Basic idea: give a **mechanized proof** that security properties are met. The compiler and/or programmer adds annotations to the code to express security-related invariants. These annotations become the **proof certificate** that the code is safe, and can be efficiently checked.

Future: Proof-carrying code?

- Ideally, we would certify code not to its origin, but with a **self-evident guarantee of security**, to capture exactly what we want. The client would check that the guarantee matches the code, and that the guarantee meets local security policy, so is safe to execute.
- This perhaps sounds far-fetched, but is the subject of current research into **proof-carrying code** (PCC).
- Basic idea: give a **mechanized proof** that security properties are met. The compiler and/or programmer adds annotations to the code to express security-related invariants. These annotations become the **proof certificate** that the code is safe, and can be efficiently checked.
- In practice, the PCC protocol allows for some negotiation to set security policy. It might also allow for the combination of cryptographic and proof certificates. Theoretical work of LFCS, Edinburgh from 80s–90s is being applied today in PCC.

Resource bounded code

The general framework of PCC is wide in scope: recent LFCS research investigated **mobile resource guarantees**, which are PCC certificates attesting that a program will work within particular constraints on resources (time, space, number of network connections, etc). Here is a picture of our scheme:



Outline

Language-based security

An active research area: applying programming language theory, designing new constructs and mechanisms.
Most work applies static analysis and extended type systems:

- Proof-carrying code (PCC), introduced above.

Language-based security

An active research area: applying programming language theory, designing new constructs and mechanisms.
Most work applies static analysis and extended type systems:

- Proof-carrying code (PCC), introduced above.
- Cyclone, Vault and others.
  Add richer, safer and more expressive typing and annotations to existing languages. [take *Advances in Programming Langauges* next year]

Language-based security

An active research area: applying programming language theory, designing new constructs and mechanisms.
Most work applies static analysis and extended type systems:

- Proof-carrying code (PCC), introduced above.
- Cyclone, Vault and others.
  Add richer, safer and more expressive typing and annotations to existing languages. [take *Advances in Programming Langauges* next year]
- Other security specialised typing includes:

Language-based security

An active research area: applying programming language theory, designing new constructs and mechanisms.
Most work applies static analysis and extended type systems:

- Proof-carrying code (PCC), introduced above.
- Cyclone, Vault and others.
  Add richer, safer and more expressive typing and annotations to existing languages. [take *Advances in Programming Langauges* next year]
- Other security specialised typing includes:
  - detecting and preventing illegal **information flows**

Language-based security

An active research area: applying programming language theory, designing new constructs and mechanisms.
Most work applies static analysis and extended type systems:

- Proof-carrying code (PCC), introduced above.
- Cyclone, Vault and others.
  Add richer, safer and more expressive typing and annotations to existing languages. [take *Advances in Programming Langauges* next year]
- Other security specialised typing includes:
  - detecting and preventing illegal **information flows**
  - **ensuring authentication** before authorisation

Language-based security

An active research area: applying programming language theory, designing new constructs and mechanisms.
Most work applies static analysis and extended type systems:

- Proof-carrying code (PCC), introduced above.
- Cyclone, Vault and others.
  Add richer, safer and more expressive typing and annotations to existing languages. [take *Advances in Programming Langauges* next year]
- Other security specialised typing includes:
  - detecting and preventing illegal **information flows**
  - **ensuring authentication** before authorisation
  - fixing patterns of **access control**, e.g. close file after opening.

References

Mark G. Graff and Kenneth R. van Wyk.

Secure Coding: Principles & Practices.

O'Reilly, 2003.

Sverre H. Huseby.

Innocent Code: a security wake-up call for web programmers.

Wiley.

Gary McGraw.

Securing Java.

John Wiley & Sons, 1999.

John Viega and Matt Messier.

Secure Programming Cookbook for C and C++.

O'Reilly, 2003.