6.857 Computer and Network Security **Fall Term, 1997**

Lecture 18 : November 4, 1997

Lecturer: Ron Rivest *Scribe: Jessica Wong*

1 Topics Covered

- Java
- Java Security Mechanisms
- Attack Applets
- Malicious Applets — *This topic was actually discussed in Lecture 19.*

2 Java

Java has been described as a "simple, object-oriented, distributed, interpreted, robust, secure, architecture-neutral, portable, high-performance, multi-threaded, dynamic language."

The basic principle of Java is that it runs on a *virtual machine.* This machine is a basically a particular well-defined architecture which will run the Java applet. An applet developer writes a program which converts Javacode into bytecode to be interpreted by a particular virtual machine.

2.1 Java features

Object-oriented: Java is object-oriented, meaning that it deals explicitly with objects which the programmer defines. This is implemented through *classes.*

Strongly-typed: The system knows what type of result to expect from each message. Types (e.g. int, float) are known at compile time.

Garbage-collection: When a new object is created, there is no need for storage allocation or deallocation. A garbage collection thread runs in the background and periodically frees up memory as appropriate. *(This is a big security feature, since many security holes are due to illegal memory accesses.)*

No pointers: There are no pointers- only *references.*

Exception Handling: Java declares a routine when an exception occurs.

Dynamic Linking: *Classes* can be brought in and linked to, as needed.

2.2 Applets vs. Applications

Applets are programs imbedded in webpages invoked without the user knowing, which therefore must follow a set of *restrictions.* Applications are stand-alone (like C programs), and have no security restrictions.

Applet restrictions:

An applet **can't**:

- read/write/delete/rename files on client
- create or list a directory
- check if a file exists or find file properties (size, type, mod time, creation date, etc.)
- connect over the network to anything except its own host
- listen for a connection
- create a top-level window without an "UNTRUSTED" banner (making it clear that the window is untrusted
- obtain a username or home directory
- run other programs
- exit the interpreter (killing all other applets)
- load dynamic libraries
- create or manipulate any thread other than those in its own Thread Group
- create a Class Loader (a program that loads a class onto a Virtual Machine)

- create a Security Manager
- define classes as part of a package (a collection of classes) on the local client system

3 Java Security

There are three parts to security:

1. Byte Code Verifier
2. Applet Class Loader
3. Java Security Manager

3.1 Byte Code Verifier

"Byte code," or machine code compiled for a virtual machine, must pass the tests of the Byte Code Verifier before it runs on the client machine. The Byte Code Verifier (written by the applet developer) checks the folllowing:

Static typing: All object types are appropriate. This is important since there are possible security breaches if an object has a different type than an applet expects (e.g. An applet thinks it's dealing with type *foo* when it's really dealing with the security manager, and then the applet changes some permissions.)

No stack overflow: At a return from a branch, the stack is the same height as it was before the branch.

Legal access to methods: There are no accesses to private methods outside of an object's own class.

3.2 Class Loader

Each applet has its own name space. A class is loaded onto a machine at the same time as an applet, which defines all associated classes and methods.

3.3 Java Security Manager

The Java Security Manager performs run-time checks to determine whether a request should be allowed. (It corresponds to a guardian or reference monitor.) It handles the following:

- no new Class Loaders created by applets
- protecting threads from each other
- file system access
- network access
- controlling access to packages

4 Attack Applets

Attack applets are applets which violate the policy, creating a security hole. Here are some examples of attack applets which have caused bugs (often an error of implementation, not of principle).

4.1 Jumping the firewall

Alice can download an applet that attacks a machine in the R&D department of her company, when the machine sits inside a firewall. How? [*The first implementation of Java used DNS addresses. It now uses IP addresses.*]

Alice downloads an attack applet from att.com onto her computer, which correctly passes through the firewall. att.com controls its domain name server, and a hacker writes his own DNS entries and assigns both **17.15.12.19** *and* **1.2.3.4** to b.att.com. Now b.att.com wants to run an applet on **1.2.3.4**. Alice's computer looks up the originator of the applet, and may find the IP address **1.2.3.4**. This is clearly wrong since the computer thinks that the applet is coming from inside the firewall, when in fact it's coming from outside the firewall.

4.2 Slash and Burn

If an applet tries to access class "foo.bar," the translation changes the "." to a "\" and looks up class foo\bar: a *relative* pathname. But if the applet tries to access "\foo.bar," then this translates into "\foo\bar," an *absolute* pathname! By allowing class names to start with the "\" character, an applet can access *any* file on the user's machine.

4.3 Type confusion

The bytecode declares its type to be "[... ." The client machine interprets the opening bracket as an *array* type, which can allow the applet to index parts of memory it shouldn't be allowed to see.

4.4 Linking

An "evil Class Loader" can create a situation where we get type confusion due to linking objects. One bug allowed the user to define a "subclass" of class Class Loader, and another allowed this subclass to instantiate without calling the instantiation routine for its superclass. As a result we get the following scenario: Class "A" references classes of types B and C. Class "B" references classes of types A and C'. Class C references a class of type C_1, and class C' references a class of type C_2. Now A makes a type C_1, and passes it to B, who thinks that it's of type C_2.

4.5 Interface casting

An interface casting allows an illegal call of a private method.

Suppose we had the commands listed below to declare interface function, f, a private function f and a public routine, f.

Then we define an array of length 2 with an instance of Dummy class and of Secure class. The loop calls the f function of Dummy, then of Secure. It should check each time whether the private function is allowed each time through the loop, but in fact the legality was only checked the first time.

```
interface Inter {void f();}
class Secure implements Inter {private void f();}
class Dummy implements Inter {public void f();}
static void attack() {
      Inter inter(2) = {new Dummy(), new Secure()} ;
      for (int j=0; j<2 ; ++j)
           inter[j].f();
      }
```

4.6 Package attack

Variables in Java are either	**private**: usable only by methods in its class
	protected: usable by class and subclasses
	public: usable by all classes
	undeclared: usable by all classes in the same *package*

The bug in Internet Explorer was that only the first component was used to compare package names. Therefore if a package name was "com.ms," a package named "com.foo" could access the variables in "com.ms" if they were undeclared.

5 Malicious Applets

Here is a list of a few "malicious applets" which are not necessarily destructive, but cause annoyance to the user.

Barking dog: An applet sets up a thread that is "immortal" ; it redefines its "stop" method to do nothing.

Business assassin: An applet kills all threads of competitors' applets.

Denial of service: An applet assigns itself as a high-priority CPU task and soaks up all the cycles.

Opening windows like crazy: rather annoying.

Forging email: An applet can take advantage of the fact that sendmail is often on port 25. Alice sends an email to Bob, and an applet telnets to port 25 of Bob's machine. The "Received from:" header contains Alice's IP address, since that's where the connection came from. The applet then changes the "Received from:" header of an email message from Alice's machine to Charlie's machine, thus forging the email to Bob.

Endnote on "Digitally Signed Applets:"

One approach to handling security of Java applets is for every applet to be signed by its developer. A browser can only run an applet if it is signed by an approved signer. The browser then stores a list of approved signers and a list of certifiers of signers. We can then have statements like "Applet is signed by author Y." and "Y is approved by X." to let us run the applet if X is on our certifier list but Y is not on our approved list.

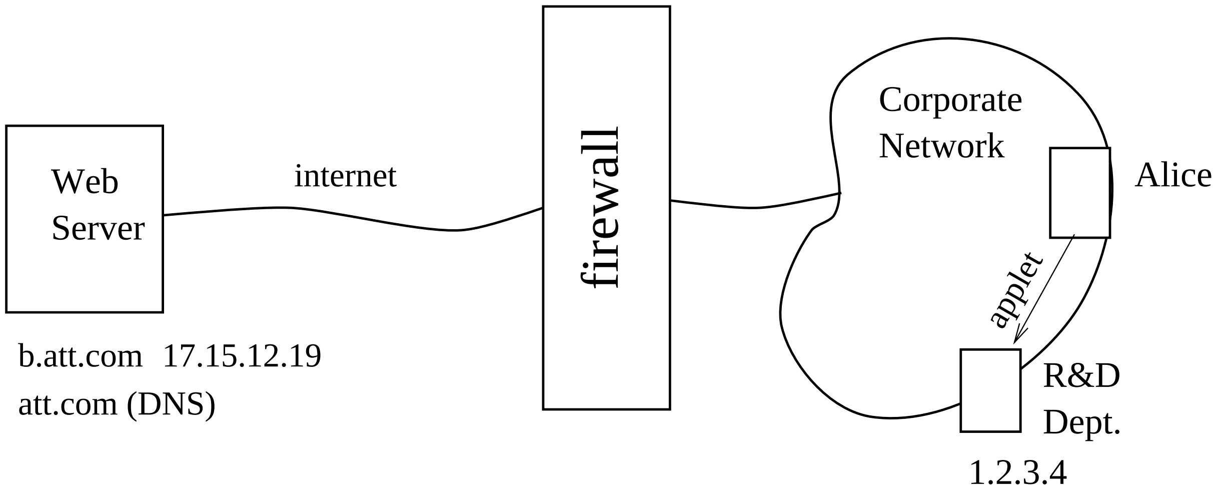

Figure 1: Jumping a firewall.

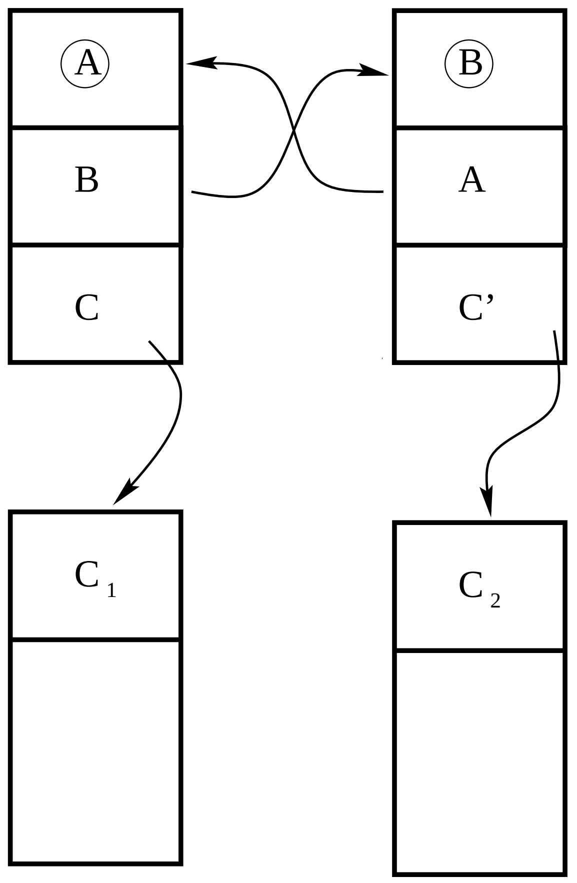

Figure 2: Linking causing type confusion.