6.857 Computer and Network Security November 20, 2001

Lecture Notes 21 : Java Security and Biometrics

Lecturer: Ron Rivest *Scribe: Bevilacqua/Itsara/Kochman/Reinstein*

1 Preliminaries

1.1 Topics

- Java Security
- Biometrics

1.2 Announcements

- The practice talks for the term projects start November 30. These should be professional, finished talks. Treat them as if you were delivering them at a conference. Remember to make slides in large fonts. A good rule of thumb is to assume 2 minutes per slide—so, for the 10 minute presentation, you will probably want about 5 slides.
- John Deutch will be talking at 4pm in 26-100. The topic is "Combating Terrorism."
- Drop date is tomorrow.
- There are 2 lectures left. If there are any topics you would like to see covered next week, email Professor Rivest or the course staff.

2 Java Security and its Holes

Last week we looked at the Java security model. Now we will look at a number of bugs which have been discovered in Java. The lesson to take from these is that security is in the details: "a small detail done wrong creates a security hole you can drive a truck through." A good book on this subject is *Securing Java*, by McGraw and Felten.

2.1 DNS Bug

The DNS bug allows an applet to attack other machines even when they are behind a firewall. An applet is allowed to communicate with its host machine. However, this bug tricks Java into allowing it to communicate with the target computer.

The key to this attack is that the attacker is running his own DNS server (e.g., attacker.org), which can return multiple IP addresses for a particular DNS name. The attack works as follows:

1. The applet downloads from www.attacker.org to the victim's machine.
2. The applet reads bogus.attacker.org. This might be the same machine as www.attacker.org, so Java performs a check.
3. It does a DNS lookup (on the attacker's DNS server) for bogus.attacker.org, and the DNS server returns two addresses: the first address is inside the victim's domain, and the second is the same as www.attacker.org.
4. The JVM checks that *any* of the returned IP addresses matches the host machine. Since one of them does, it allows the connection.
5. Then, in order to decide which IP to use for the connection, the JVM picks the *first* IP address off the list of those returned. Note: this is the bug.
6. Now, the attacker applet has made a connection to the target machine inside the victim's domain.

This is a very simple error, and was easily fixed, but it caused a lot of problems when it was discovered.

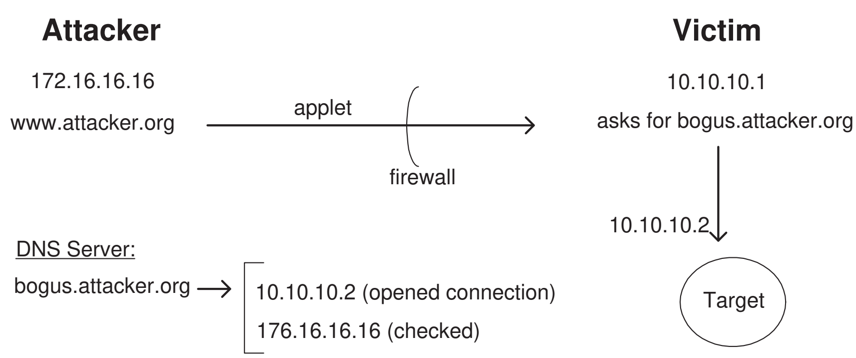

Figure 1: The DNS Bug.

Question: Is the DNS-related bug that tricks the computer into accessing another computer within its local network fixed now?
Answer: Yes, it is. The way to fix this is to only allow new connections made by the applet to be to a machine of the same IP address.

2.2 Steal This IP Address

This bug allowed a server to discover a client's real IP address, even when the client was using an anonymizer to hide it. The applet simply called "InetAddress.getLocalHost()", and the real IP of the machine it was running on was returned to the server. This was fixed by making it return the loopback address, 127.0.0.1.

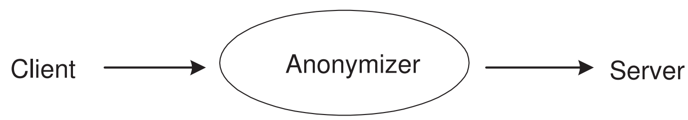

Figure 2: Anonymizer data flow.

2.3 Look Over There

This attack used the URL redirect feature. The applet would load a file of the form URL = getDocumentBase() + "cgi-bin/redirect?where=target.com/foo". The server then returned the HTTP redirection to any location on the Internet. Once the applet was in, this worked even behind a firewall.

Question: Does the "look over there" bug allow an attacker to cause the user to open a non HTTP socket to a site?
Answer: Not that I know of, although it could do something like e-mail.

2.4 Cache Cramming

This attack exploited the fact that early browsers stored files in the cache in a predictable way. The attack went like this:

1. Get a class file into the browser cache.
2. Within an applet, reference the cached file using a file:// URL. In other words, load it like a local file, even though it's actually a class file from the attacker's server.
3. Now, the class's "home" is the local client machine. The applet can communicate with its home using TCP, so it can do port scans, etc. on the client machine. This allows it to find vulnerabilities or open any kind of connection.

2.5 Slash and Burn

A Java class a.b.c is stored in the directory a\b\c. However, there was a bug in the early implementations that allowed a class name to start with a backslash. If an attacker named a class "\program.sun.foo", it would be stored in \program\sun\foo, thus allowing arbitrary access to the local file system. In JDK 1.0.2, local code was always trusted, so the attacker could do nearly anything with this exploit.

This bug demonstrates that Java security is a very hard problem to solve, and that thorough code reviews are important.

2.6 Applets Running Wild

This vulnerability allows an attacker's applet to instantiate a class loader of the attacker's choice. Applets should not be able to instantiate their own class loader. Java enforces this rule by requiring that an applet class loader be a subclass of the generic class loader. However, a method of preventing the call to the generic class loader constructor was found. In this case, the security manager is never loaded, since it is contained in the generic class loader. Without a security manager, the attacker can use type confusion to make any change he wishes, including changes to the security manager. This leads to the following metatheorem:

Metatheorem 1 *Any type confusion implies a full security break.*

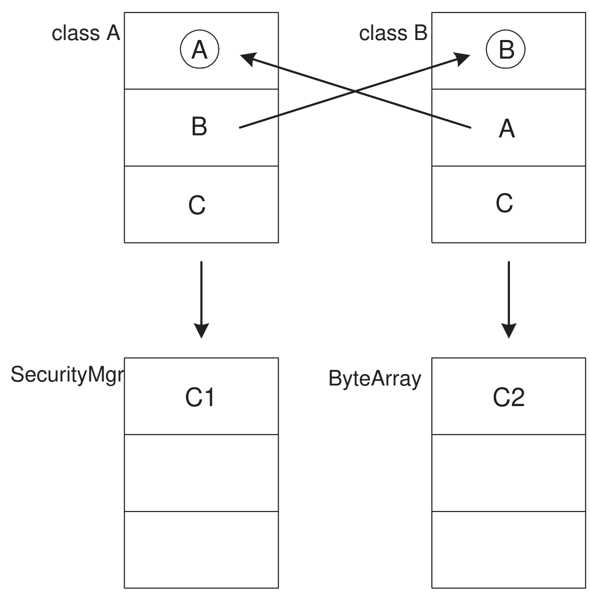

Figure 3: An example of how type confusion can be used to change data. If a method in class A passes an object of type C to a method in class B, and if class A interprets C to mean C1, and class B interprets C to mean C2, then modifications to the passed object by the B method may result in unexpected or malicious changes to the object as interpreted by the methods in class A.

Question: How does the int-to-pointer conversion bug convert an int to a pointer?
Answer: It isn't a pointer; it's a reference to an object.

Question: Would C++ allow casting attacks?
Answer: Yes, C++ has weaker casting rules than Java.

2.7 Verifying the Verifier

The Kimera project at the University of Washington involved implementing a bytecode verifier. To test it, they compared it to existing bytecode verifiers on millions of randomly generated variations of some standard code. They found 24 flaws in Sun's verifier, and 17 in that of IE. One such error allowed integer to pointer conversion.

2.8 Lessons

What can we learn from all of this?

- Security is hard! Don't underestimate it.
- The devil is in the details: even a small hole can turn into a dangerous exploit.
- Open source coding can help find bugs.
- Automated testing can really help.

3 Biometrics

Biometrics seeks to authenticate users based on who the person *is*, rather than something they know or have (e.g., a password). It is important to realize that it is not a cure for all user authentication issues.

3.1 Current techniques

Currently, the most prominent biometric techniques are the following:

- Fingerprints
- Iris scanning
- Hand geometry (width and length of fingers, for example)
- Face recognition
- Voice recognition
- Retina scanning (less user-friendly than iris scanning)
- Signature recognition
- Keystroke latencies (how a person types various words)

Question: Aren't keystroke latencies inaccurate?
Answer: It depends on your goals. It is less accurate when dealing with a large number of people, but good enough for some applications.

Statement: I heard keystroke latencies could be used to tell if a person is typing under duress.
Response: That is possible.

3.2 Authentication Process

The user first "enrolls" in the system, a process which connects a particular user with the biometric information. This information connected to a user is known as the "template." To authenticate a user, the system takes a biometric reading and compares it to the template. Based on a sensitivity threshold, the system then either authenticates or rejects the user.

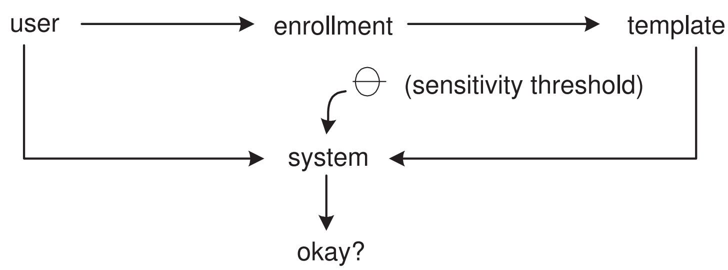

Figure 4: Biometric authentication process.

A big issue, therefore, is where to set the sensitivity threshold. Such a system can produce the following two errors:

- False positive - The system authenticates someone who should not be authenticated.
- False negative - The system incorrectly rejects a valid user.

Neither error is inherently worse. Furthermore, the number of false positives decreases and the number of false negatives increases as the sensitivity threshold increases. Therefore, deciding where to set the sensitivity threshold depends on the requirements of the system. One common measurement is the "crossover error rate." This is the point at which the number of false positives equals the number of false negatives. In most cases, it is difficult to get a crossover error rate less than 1%.

3.3 Issues with Biometrics

There are many issues and difficulties with biometrics, some of which are discussed below.

- Real-world complications such as dirt and sensor noise can increase the error rates and cause user frustration.

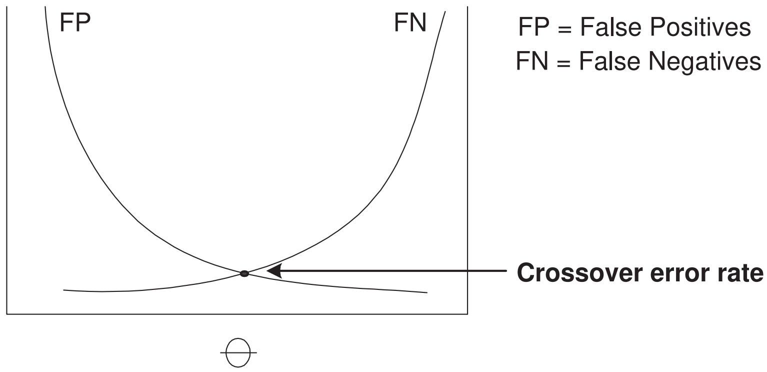

Figure 5: Error rates versus sensitivity threshold.

- An individual's biometric information can change. For example, a man shaving his beard could prevent him from being authenticated by a face-recognition-based system.
- Templates cannot be stored in a hashed form like passwords. The nature of hashing functions means that two nearly identical sets of biometric data have totally different hash values.
- Users may not be willing to accept certain biometric measurements. For example, a user may not want a light shined in his eye, and taking a fingerprint may make him feel like a criminal.
- Depending on the level of accuracy, biometric templates can get quite large.
- Some biometric methods have computationally-intensive authentication processes.
- Certain biometric equipment is bulky and expensive, though these sizes and costs are dropping rapidly.

3.4 Remote Biometrics

The main issue in a client/server biometric authentication scheme is where to compare the input data with the template. Doing a server-side comparison requires a secure path from the client, since it would otherwise be vulnerable to replay attacks. On the other hand, a comparison on the client might be vulnerable to template forgery.

Question: What can be done if someone's biometric information is stolen?
Answer: Not much. It might be better to use biometrics as an additional level of authentication, rather than a lone method.

Question: Can a server use your biometric information to duplicate logins at other places?
Answer: Good question, since hashing cannot be used.

Finally, with biometrics becoming smaller and less expensive, we could see such devices everywhere in the near future.

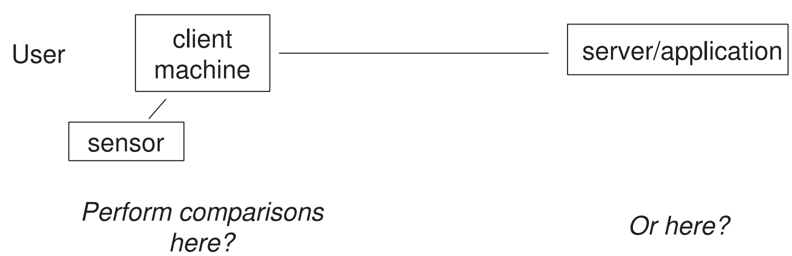

Figure 6: Remote biometrics layout.