Mobile code security

- **General:**
- Java, introduced by Sun Microsystems 1995, is a very widely used programming language. We will not look at Java as a programming language as that is covered by other courses.
- Java has fairly good security and the mechanisms used there are relevant for all ways of implementing mobile code.
- Java is mobile code in the sense that a Java applet is often fetched to the computer via WWW from the Internet and executes in the computer.
- There are more advanced forms of mobile code, like mobile software agents, which can execute in a computer and move from one computer to another through the network.
- Security issues for advanced forms of mobile code are still open, Java solutions have restricted applicability.

Mobile code security

- The problem of securing mobile code looks at first sight impossible, yet the solutions are fairly good.
- The idea is that a user can fetch code from the Internet, often automatically without any way of knowing that the code is fetched.
- The code is run in the user's computer. How can this be secured?
- Naturally there are hackers and other writers of malicious or incorrect code. It is clear, that one cannot let untrusted mobile code to be native machine code running with all rights to the computer.
- There are two ways: restrict the access for the mobile code - but then it cannot do very many things; or add credentials so that mobile code carrying credentials can be trusted.

Mobile code security

- Security methods in mobile code use the following basic methods:
- **Sandbox:** Mobile code is executed in an environment, where its access to any resources is restricted to activities which are considered safe.
- A sandbox model can be implemented in many ways, in Unix one can limit access by chrooting, access controls, process privileges. Java sandbox model is one way of making a sandbox. It has a bytecode verifier, class loader, security manager. Additionally there is an access control mechanism, which relies on permissions for signed applets.
- In the sandbox solution all mobile code is untrusted, the sandbox environment is trusted. There have been bugs in sandboxes.

Mobile code security

- **Signed code:**
- In this solution a hash value is produced from the code and it is encrypted by a private key of a public key cryptosystem.
- When code signed in this way is received, the signature is checked and access rights to the mobile code are granted according to some policy, typically signed mobile code will have large access rights.
- In this method the trust model is that mobile code producers of code signed with a trusted key are trusted.
- This trust model assumes that a trusted site will never be corrupted, so that the public key cryptosystem is broken or that the code is malicious even though it has a valid signature.

Mobile code security

- **Firewalling**
- This method means looking at the mobile code and executing it on a playground machine, which is dispensable in case the code is malicious. There is no strict difference between this and the sandbox approach, but some methods of protecting against mobile code state using firewalling.
- **PCC (Proof Carrying Code):**
- This is an experimental technique, proposed for mobile agents. There are ways to proof that some kind of code will not do malicious actions, like run over buffers etc.
- In the PCC method you add a proof to the code. These methods work so far only to limited types of code and the method is not in practical use yet.

Java security

- **General:**
- Java is portable interpreted language, current version is Java 2. A Java program is compiled to Java bytecode, which is interpreted by Java Virtual Machine to the native machine code of the processor.
- Java bytecode does not have jump addresses inserted, but the symbolic references are inserted on runtime.
- Java is object oriented, so all symbolic references to other routines are references to classes.
- These classes can be local classes or they can be fetched on runtime from an URL by the Java Class Loader.
- Java is multi-threaded. One thread handles automatic garbage collection.
- Java is rather high performance and rather secure.

Java security

- **Java sandbox model**
- The basic Java security model is the sandbox model. In that model received Java applet is:
- Loaded by Class Loader, it sets the namespace of a remote applet in a way that the applet cannot access directly other memory.
- Investigated by the Bytecode Verifier. Most types of efforts to access other than allowed resources are stopped here.
- If it is a remote applet, still investigated by the security manager module. This can use access control policy.
- If the code is allowed to be run, it cannot be harmful.
- The main problem may be that security assumes that the sandbox works fine. Any errors there and security can fail.
- Java Run-Time Environment is a possible place for errors.

Java security

- **Limitations of the sandbox model**
- Sandbox limits mobile code to perform only allowed actions.
- This works, but it limits Java applets to do only a limited range of actions.
- If all you want to do is to, say show graphical animations on the screen, then the limited range of actions is easy to achieve.
- Consider an antivirus applet: it is downloaded from a antivirus software provider, comes to check the memory, disc and programs.
- Clearly a useful applet, but cannot be made with the sandbox.

Java security

- **Digital signatures**
- To extend applets to tasks which demand more access rights, SUN Microsystems introduced 1997 signed Java applets in JDK 1.1. (Java Development Kit)
- Signed applets are delivered in Java Archive format.
- JDK 1.1 and higher release security solution is a hybrid solution: there is the sandbox and additionally signatures.
- The trust model is that all code is untrusted except for code from a trusted supplier who has signed the code. The trusted supplier is assumed to be incorruptible.
- If should be mentioned that revocation of public keys is in general a problem in public key systems, that is, if a private key is compromised, the public key is cancelled by a revocation. How to spread the revocations to everybody?

Java security

Java avoids many software problems C or C++ has. Memory management (malloc, free and their varieties) is one cause of problem. Java has an automatic garbage collection running as a separate thread.

Java as a language has no pointers like C or C++.

Therefore no correctly working Java compiler will make bytecode with pointers pointing to something else than a valid memory place.

However, as a hacker you might write Java bytecode in some other way, like you can write it by hand and make a bad pointer. Such is called a forced pointer.

You can also make a bad data conversion and in this way try to create problems. The bytecode verifier ir to stop this, it generates immediately an interrupt.

Java security

- **Bytecode verifier**
- The most serious threat is that Java applet could jump through an invalid pointer to some memory area outside the sandbox.
- Java bytecode coming from outside is not trusted in the sandbox model. All code must pass investigation by the bytecode verifier.
- Bytecode verifier checks that there are
- No forced pointers
- No access restriction violations
- No object mismatching
- No operand stack over- or underflows
- Parameters for bytecodes are all correct
- No illegal data conversion

Java security

- The bytecode verifier has simple checks as a first stage, but checking for forged pointers and more advanced things relies on automatic correctness proofs made by the verifier.
- Better checks for a malicious applet code have been proposed. Finjan is one third party applet checker code supplier.
- There is no clear view if such products can do anything more than the standard bytecode verifier. After all, the halting problem already shows that from arbitrary code it is impossible to say with any finite algorithm what it does by looking at the code.
- It seems relatively easy to create malicious applets that pass Finjan's SurfShield, Smartgate or other products.
- See Mark LaDue’s page at
- www.rstcorp.com/hostile-applets/index.html

Java security

- **Java class loader**
- Java bytecode is not linked in the way a linker works, i.e., by inserting jumps to the correct memory places.
- Java is dynamically linked, so the exact memory places can change.
- When a class loader brings in a class it places it to own namespace. This way the potentially malicious bytecode cannot do much harm.
- Local classes are in one address space and can directly address each others memory.
- There is no way for the Class Loader of knowing if somebody has created a new tampered class which has the same name as some old existing class. Signatures, bytecode verification etc. must be used to do this.

Java security

- **Security manager module**
- With local and remove classes the Java sandbox model first loads them with Class Loader which sets the name spaces etc.
- Then the bytecode is passed through the bytecode verifier.
- Remote classes are passed to a second verification by security manager. Example shows how security manager is called.

```
Public boolean XXX(Type arg1) {
    SecurityManager security =
    System.getSecurityManager();
    if(security != null) {
        security.checkXXX(arg1);
    }
}
```

Java security

- The security manager makes run-time verification of methods that request access to any resources, like file IO, and network access.
- It is also the security manager’s responsibility to stop an attacker from defining a new class loader.
- It manages socket operations, personal data, guards access to files, access to operating systems programs and processes, maintains integrity and controls access to Java.
- Therefore it is probably the most important security feature of the sandbox model.
- It implements the security policy, which can be customized by the user.

Java security

- **Access controls**
- If an Java applet is executed in the sandbox, it will not be able to access most of the resources of the computer.
- In JDK 1.2 the class loading mechanism is improved and allows permission-based extensible access controls.
- There is a table of permissions set for each piece of code. User can specify his security policy in the table.
- Digital signature of the code is checked and if verified, the table access permissions apply, else the code is treated by the sandbox mechanism.
- Access controls apply both to Java applications and to Java applets.
- Interfaces supporting access controls are in Java security package.

Java security

- **Java virtual machine (JVM)**
- Java is an interpreted language, but avoids the slowness of an interpreted language by compiling the source first into Java bytecode.
- JVM is a virtual processor, which runs programs written in Java bytecode. JVM is implemented by translating the bytecode to the true machine code.
- The mapping is not one-to-one. In DISC-processors, like Intel’s processors there is a large instruction set and many routines are made in microcode, i.e., as small software programs in the processor.
- In RISC processors there is only a reduced set of instructions implemented directly in silicon. Therefore RISC-processor machine code is longer but runs very fast.

Java security

- **Opcodes, operands**
- Opcodes are the instructions in Java bytecode.
- Operands are the parameters to the instructions.
- Opcodes are 8-bits long and have variable length operand field. It is not practical to keep a 4-byte opcode alignment as in RISC because that would waste memory.
- Consequently, some of RISC speed is lost when interpreting Java bytecode.
- The number of opcodes is therefore limited to 256. Currently there are 160 valid opcodes.
- Java uses big endian representation for numbers (i.e., most significant bits are in the first bits). It is native in Motorola and in RISC, Intel uses little endian representation.

Java security

- **Registers**
- JVM has only the following registers:
- pc = program counter
- optop = pointer to top of the operand stack
- frame = pointer to current execution environment
- vars = pointer to the first (0th) local variable of the current execution environment

  This is a very small set of registers. Those familiar with assembly languages may be wondering how it is possible to implement very fast code without accumulator and memory pointer registers.

  The answer is probably that it is not possible to implement super fast code with these registers and we should take evaluations of Java performance with some consideration to this.

Java security

- **Memory model**
- The memory model in JVM contains three parts:
- **Java stack** is the main storage method for the JVM. More of Java stack on the next page.
- **Garbage Collected Heap** is the store in memory from which class instances are allocated. So, this is the are which corresponds to memory you can allocate by calling malloc in C. There is a thread on the background releasing memory pointer out by handles which are no longer in use. This is the automatic garbage collection.
- **The Memory Area**
  - The method area : this is the place where the bytecode for the Java methods is.
  - The constant pool area : class names, method and filed names, string constants are stored here.

Java security

- Java stack has also three areas:
- Local variables pointer out by the **vars** register

  Local variables is an array of 32-bit variables used to store the local variables of methods.
- Execution environment pointed out by the **frame** register

  The execution environment contains information of:
  - previous method invoked
  - pointer to the local variables
  - pointers to the top and bottom of the operand stack
- Operand stack pointed out by **optop**. FIFO where the operands of bytecode instructions are and results or the instructions are stored. It corresponds to the stack in normal assembly languages.

Java security threats

- Java is rather safe, there are however some concerns.
- **Bugs**
- Attack applets have been demonstrated using of bugs in JVM in HotJava implementation of JDK 1.0.
- **Java dead-lock, Denial-of-Service**
- As Java is multi-threaded, it is possible to use the threads to create a deadlock by several threads trying to access the same resources and locking each other. This can be made for instance in JDK 1.0.
- Locking is one kind of denial-of-service, other DoS attacks exist. In Java these are called Malicious applets.
- **Email forgery**
- It has been demonstrated in a lab, that applets can send email.

Java security threats

- Java applets can make four type of attacks:
- Modify the system
- Invade user’s privacy
- Denial-of-Service by taking some resources
- Antagonize a user.
- The first type of attack is possible but so far only done in a laboratory.
- Other three types have been made. A large collection of malicious applets was placed July 1998 on a WWW-site and the applets were later used to launch an attack.
- In general, WWW and applets provide a new distribution channel for malicious code and it is not necessary any more for the code to be a virus or worm to spread.

Java security threats

- The collection of hostile applets is probably Mark LeDue's
- www.cigital.com/hostile-applets/index.html
- which has a link to the original hostile applets page with the source of these applets.
- There are for instance the applets:
- bear that insists on marching on the beat of a drummer
- applet which can bring down Netscape 3.0
- applet that causes Netscape 3.0 to hang
- applet which tries to take over the workstation
- applet which steals your password and login name and attacks if you try to quit
- an applet killing other applets
- an applet which will be factoring a large number on background, that is stealing your computer resources etc.

Java security threats

- It is interesting to see how Dr. LeDue made these applets. He developed a class disassembler, disassembled Netscape Java classes and found bugs.
- The hostile applets can destroy files, run native machine code and do most nasty things.
- The message is that there are bugs, but an attacker should learn assembly and reverse engineer code to find new bugs.
- There are a number of bugs for Java in bug lists, like
- Java Security Hole in Navigator (Netscape 4.0x)
- Sohr Java Vulnerability
- There are several old bugs for JDK 1.0 and Netscape Navigator 2.02.
- Secure Internet Programming team at Princeton University is very active in locating bugs for Java.

Java security

- **Java Cryptography Extension**

  JDK has the Java Cryptography Extension which provided plug-in cryptographic libraries where you can insert own algorithms. An example of the usage of this is an implementation of the German second world war Enigma algorithm in software for Java. It is only for fun as Enigma is long ago broken. (Doctor Dobbs J. March 1999, article by Paul Tremblett.)

- **Java Secure Sockets Extension**

  SSL (Secure Socket Layer= Transport Level Security) extension can be obtained to sockets opened by Java applets. DDJ Feb 2001.

- **Storing a Java applet in a self-executing encrypted file.**

  Yes, this is possible, consider if there are benefits for a hacker - does it make stealthing possible. Usage of it to protect your code from illegal copies is in DDJ Feb. 1999 p. 115).

Security methods in mobile code

- **MicroSoft Active X**
- Active X uses only digital signatures in the form Microsoft Authenticode.
- Security problems can be coming from a trusted Active X component being able to access users files and open access to any other Active X components.
- Security problems of Active X have been demonstrated, e.g. on television by the Hamburg hacker group Chaos Computer Club. They could transfer money from an account.
- A general opinion is that signatures alone are not a sufficient method.
- It is a common opinion, but not shared by all experts (like Gary McGraw, an expert on Java security), that Java is more secure than ActiveX.

Security methods in mobile code

- **JavaScript**
- JavaScript if a scripting language developed by Netscape. Not to be mixed up with Java.
- JScript is MicroSoft's clone of JavaScript.
- Like Java applets, JavaScript code is inside a HTML-page.
- JavaScript is a loosely typed, dynamically linked script language. It has object based properties, but for instance no inheritance, so it is not fully object oriented.
- Source code for Mozella browser from Netscape implementing JavaScript is freely available.
- There is no real security model for JavaScript. There is a proposal for such a model in IEEE Internet Computing December 1998. It proposes access controls with different levels of access and access lists.

Security methods in mobile code

- **JavaScript**
- There is a hierarchy of objects:
- navigator object
- location object
- document object
- history object etc.
- Security is based on an interpreter, which tries to limit access. The hierarchy tree allows giving access differently to different branches.
- The interpreter uses a security policy assigned by a user.
- Code signing is included in JavaScript 1.2.
- There are many known flaws in JavaScript.
- Other script language alternatives exist: VBScript (MicroSoft)

Security methods in mobile code

- **Security awareness of mobile code users**
- A questionnaire of Fortune’2000 from IEEE Spectrum August 1999 p. 41.
- 92% are concerned of Java and ActiveX security.
- 72% allow Java or ActiveX to enter their network
- 56% do not scan Java or ActiveX code
- 31% scan the code in a firewall
- 8% scan it on desktop computer
- 6% scan it on application server

- What does this mean? People are concerned but do nothing.