Aplikationssicherheit

Jan Fässler & Ege Kaba

5. Semester (HS 2013)

If you use this documentation for a exam, you should offer a beer to the authors!

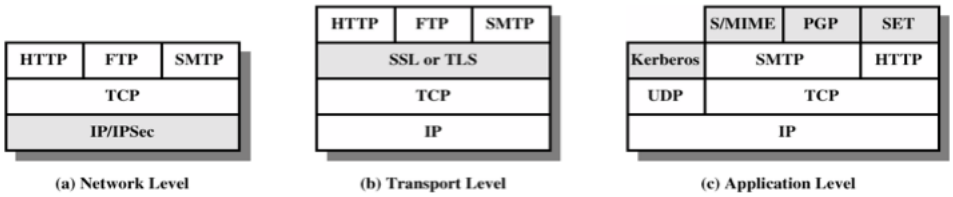
Inhaltsverzeichnis

1 SSL

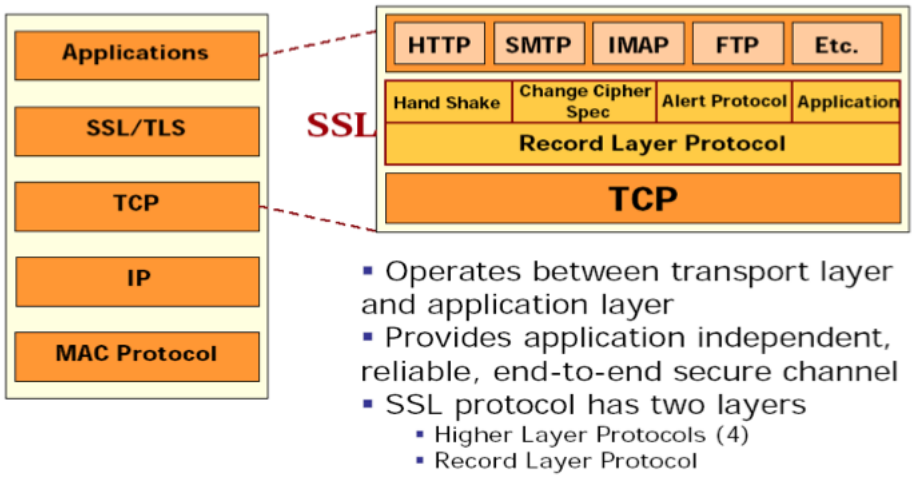
1.1 The secure channel

- A, B key-exchange protocol that establishes an **Authenticated, Secret Session Key** between A and B
- This **session key** is used together with a **symetric key** cryptographic function to protect the **integrity** and/or **secrecy** of the transmitted data.

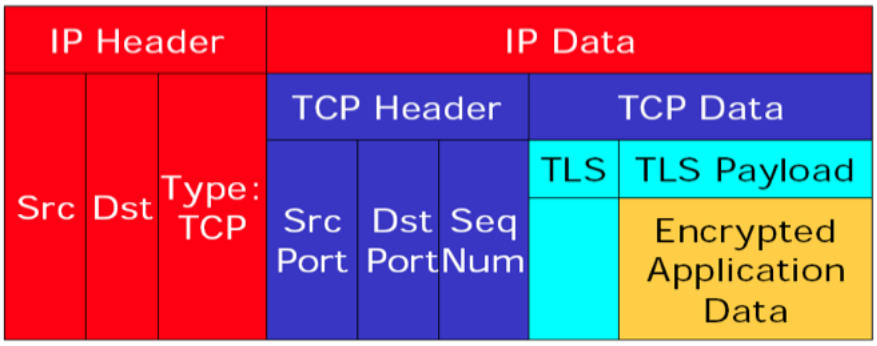
1.2 security in the TCP/IP stack



1.3 diffrent parts of the SSL protocoll



1.4 SSL(TLS) over TCP/IP



1.5 SSL characteristics

- Netscape Secure Socket Layer

- Layered between the application and TCP
- Server authenticated with public keys (X509)
- Can authenticate client (rare)
- Privacy enforced by encryption
- Integrity enforced by MACs
- Works with any TCP application

1.6 SSL protocols

SSL provides end to end security, based on a communication protocol. The different protocols implemented by SSL:

- Handshake
- Record (specifies how the data are encrypted)
- Change cipher specifications
- Alert services

1.7 Handshake protocol

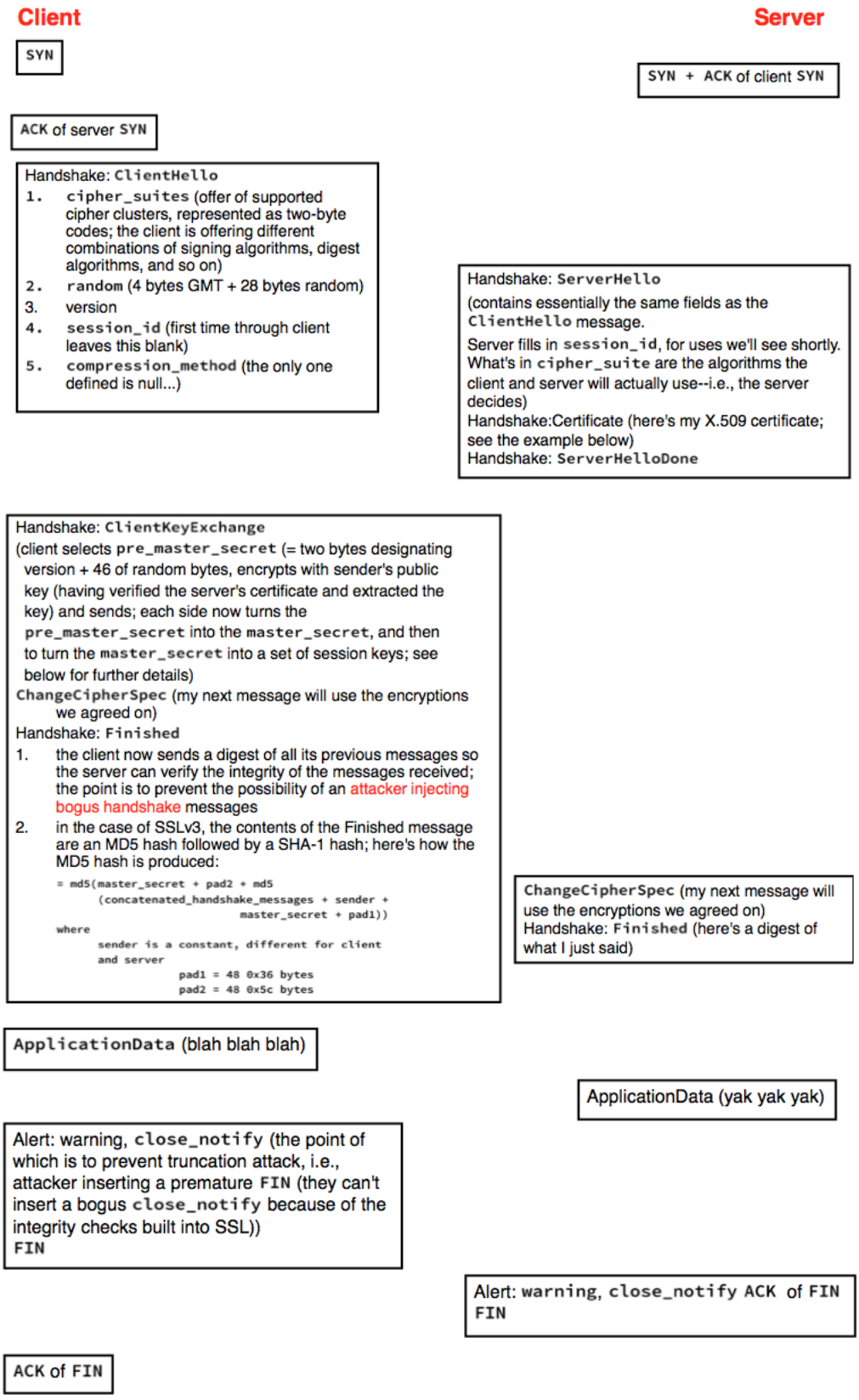
The ost complex and unsecure part of SSL are:

- Allows the server and client (**optional**) to athenticate each other
- Negotiate encryption, message authentication code algorithm and cryptographic keys
- Used before any application data are transmitted

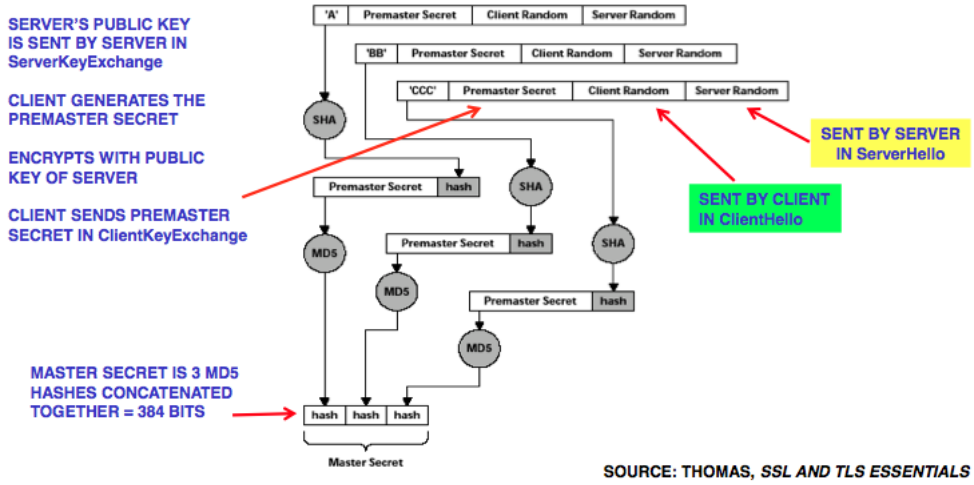
1.7.1 SSL protocol basic handshake features

1. The client initiates the connection to the server and tells the server which SSL cipher suites the client supports.
2. The server responds with the cipher suites that it supports.
3. The server sends the client a certificate that should authenticate it.
4. The server initiate a session key exchange algorithm, based in part on the information contained in the certificate it has just sent, and sends the necessary key exchange information to the client.
5. The client completes the key exchange algorithm and sends the necessary key exchange information to the server. Along the way, it verifies the certificate.
6. Based on the type of key exchange algorithm (that in turn is based on the type of key in the server's certificate), the client selects an appropriate cipher suite and tells the server which suite it wishes to use.
7. The server makes a final decision as to which cipher suite to use.

1.7.2 protocol gritty details

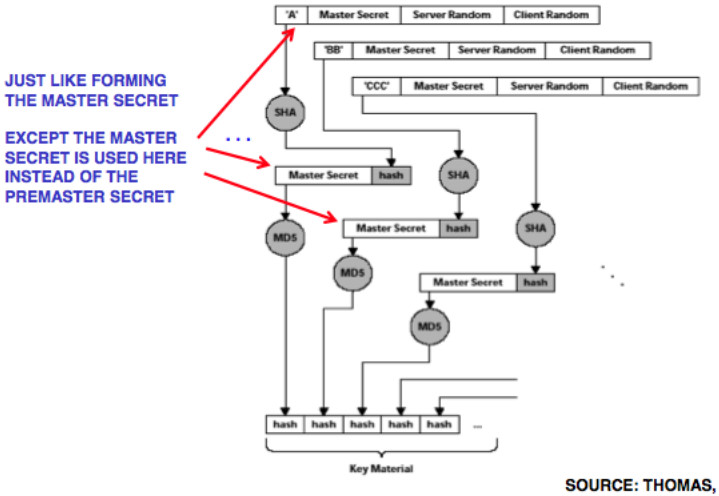


1.7.3 generating Master Secret



SOURCE: THOMAS, *SSL AND TLS ESSENTIALS*

1.7.4 generating key material



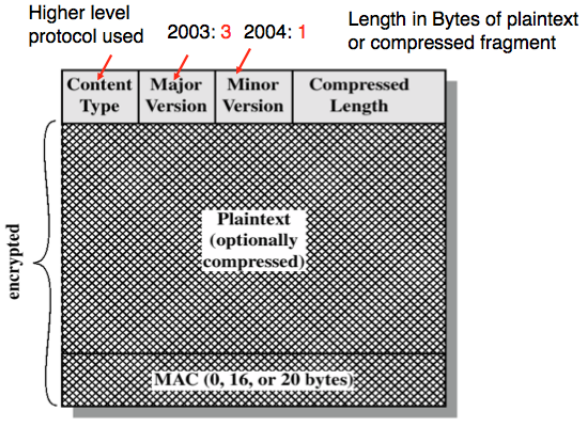
SOURCE: THOMAS, *SSL AND TLS ESSENTIALS*

1.8 Record protocol

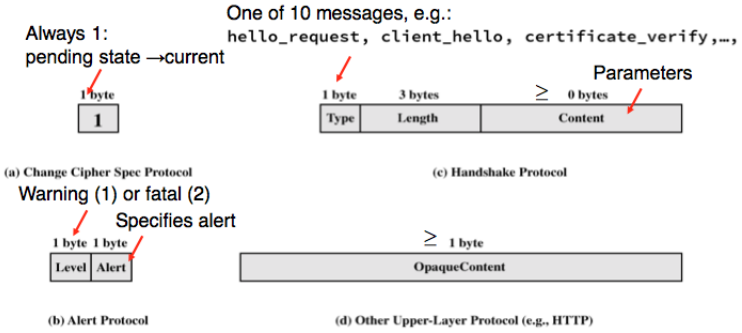
1.8.1 Ablauf

1. Provides confidentiality and integrity.
2. Fragments message (214 bytes).
3. Optional compression (default: none).
4. Calculates MAC (Message Authentication Code = integrity). MAC traditionally is based on symmetric block cipher, i.e. MAC = CK(M) where M is the message, MAC a fixed length authenticator, K a secret symmetric key shared between sender and receiver and C(.) a function.
5. Modified H(Hash)MAC to include sequence number (prevent replay attacks).
6. Encrypts both message and MAC (confidentiality+integrity).
7. Propend header.

1.8.2 Format



1.9 SSL remaining protocols



1.10 TLS enhancements

Differences in the:

- version number
- message authentication code
- pseudorandomfunction(PRF)
- alert codes
- cipher suites
- client certificate types
- certificate_verify and finished message
- cryptographic computations
- padding

Otherwise similar to SSL v3.1. Important: the same record format as the SSL record.

1.11 Words to the wise about applied cryptography

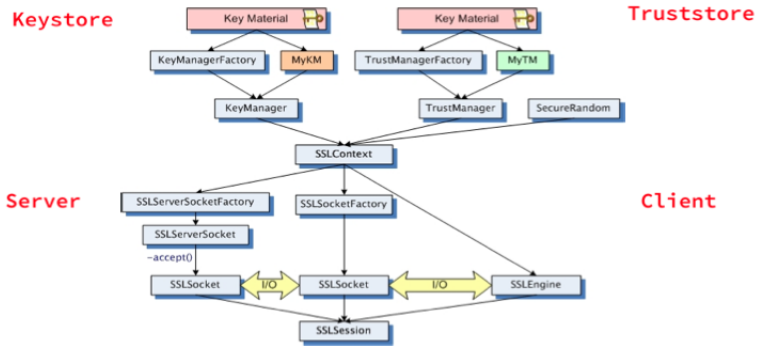
1. Hashing and symmetric key ciphers are fast: use them for session encryption.
2. RSA public key is slow, but it is useful for key exchange and user/server authentication

3. Informal web of trust of PGP (you are in charge) versus CA based (you do not know, who controls it: in fact the NSA controls it (note added in 2013))
4. A good protocol should:
   (a) negotiate **cryptographic** parameters
   (b) establish **shared** secret
   (c) authenticate **endpoints**
5. Use ssh, pgp liberally as the crypto part of an application.
6. SSL: a transport interface, but you still needs to modify the application
7. IPsec places cryptography where it belongs: at the network layer.

1.12 SSL and Java applications

1.12.1 Java SSL software (JSSE)

Set of API to construct SSL-Client and SSL-Server sockets.
Two important new concepts: **trust store** and **key store**. Both are databases that hold certificates. Key store are used to provide credentials; trust store to verify them.



1.12.2 X.509 certificate

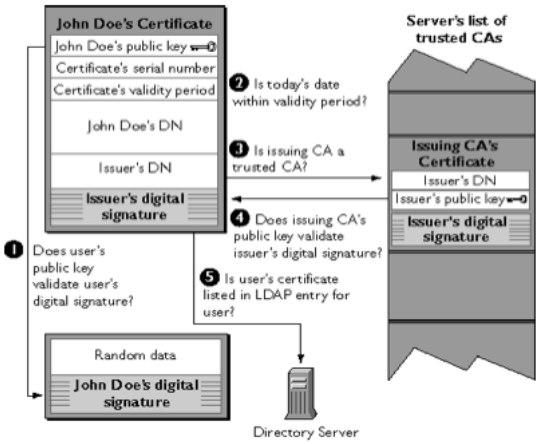
Every X.509 certificate consists of two sections: (i) data and (ii) signature.
The data section includes the following information:

1. The **version number** of the X.509 standard supported by the certificate.
2. The certificate's **serial number**. Every certificate issued by a CA has a serial number that is unique to the certificates issued by that CA.
3. Information about the user's **public key**, including the algorithm used and a representation of the key itself.
4. The **DN** of the CA that issued the certificate.
5. The period during which the certificate is **valid** (for example, between 1:00 p.m. on January 1, 2000 and 1:00 p.m. December 31, 2000).
6. The DN of the certificate **subject** (for example, in a client SSL certificate this would be the user's DN), also called the subject name.
7. Optional **certificate extensions**, which may provide additional data used by the client or server. For example, the certificate type extension indicates the type of certificate - that is, whether it is a client SSL certificate, a server SSL certificate, a certificate for signing email, and so on. Certificate extensions can also be used for a variety of other purposes.

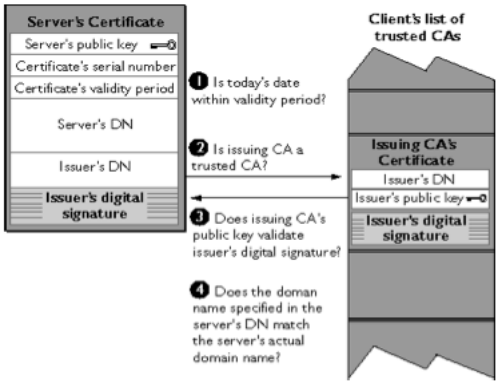
The signature section includes:

1. The **cryptographic algorithm**, or cipher, used by the issuing **certificate authority (CA)** to create its own digital signature.
2. The CA's **digital signature**, obtained by hashing all of the data in the certificate together and encrypting it with the CA's private key.

1.12.3 Webserver verification of client's certificate



1.12.4 Client verification of server's certificate



1.12.5 Certification path

The chain, or path, begins with the certificate of that entity, and each certificate in the chain is signed by the entity identified by the next certificate in the chain. The chain terminates with a root CA certificate. The root CA certificate is always signed by the CA itself.

1.12.6 A secure Webserver in Java

Listing 1: secure webserver

```
1 public class HTTPSSErver {
    public static void main(String[] args) {
      SSLServerSocketFactory ssf = (SSLServerSocketFactory)SSLServerSocketFactory.getDefault
          ();
      SSLServerSocket ss = ssf.createServerSocket(8443);
```

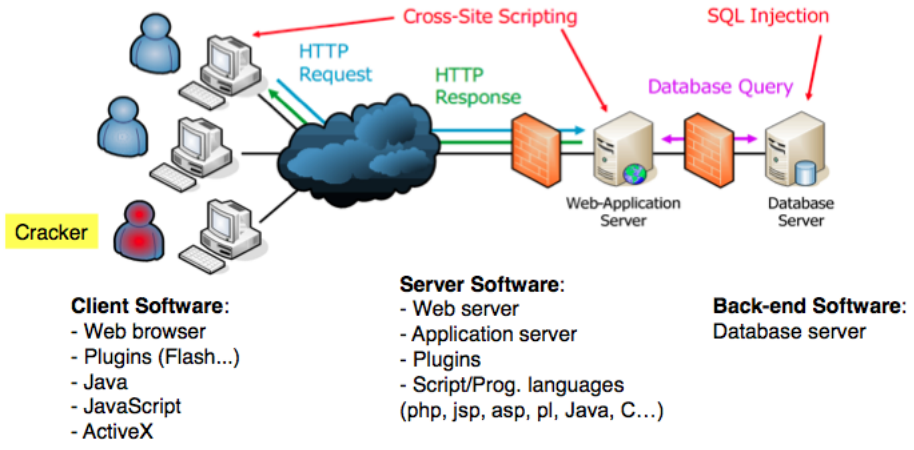
```
6       System.out.println("SSLSocketServer is listening...");

        while (true) {
          try {
            Socket socket = ss.accept();
11          OutputStream out = socket.getOutputStream();
            BufferedReader in = new BufferedReader(new InputStreamReader(socket.getInputStream
                ()));
            String line = null;
            while(((line = in.readLine()) != null) && (!("".equals(line)))) {
              System.out.println(line);
16          }

            StringBuffer buffer = new StringBuffer();
            buffer.append("<html>\n");
            buffer.append("<head><title>HTTPS Server</title></head>\n");
21          buffer.append("<body>");
            // more noise here ...
            buffer.append("</body>\n</html>");

            byte[] data = buffer.toString().getBytes();
26          out.write("HTTP/1.0 200 OK\n".getBytes());
            out.write(new String("Content-Length: "+data.length+"\n").getBytes());
            out.write(data);
            out.flush();
            out.close();
31          in.close();
            s.close();
          } catch(Exception e) { e.printStackTrace(); }
        }
      }
36  }
```

2 Validation

2.1 Fundamental principles

1. **Least privilege**: Both users and applications should have only the minimum set of privileges necessary to run.
2. **Economy of mechanisms/Simplicity**: "Techniques such as line by line inspection of software and physical examination of hardware that implements protection mechanisms are necessary. For such techniques to be successful, a small and simple design is essential."
3. **Open design**: The protection mechanism must not depend on the attacker's ignorance. Security by obfuscation is not only stupid, but dangerous too.
4. **Complete mediation**: Every access attempt must be checked.
5. **Fail-safe defaults**: The default should always be denial of access. The protection scheme should clearly identify under which conditions access is permitted.
6. **Separation of privilege**: Access to objects should depend on more than one condition, so that defeating one protection's layer doesn't grant complete access. (Compartmentalisation.)
7. **Least common set of mechanism**: Minimize the amount and use of shared mechanisms.
8. **Easy to use**: If the protection is not intuitively simple to learn, the users will ignore it.

2.2 Points of attack



2.3 SQL injection

2.3.1 login attack

The cracker (always aptly named Oscar) logs in:

- He tries to manipulate the SQL query in such a way that it always returns at least one row.
- With logins, this often works with 'or' '='
- GET/path/login.pl?user='or"='&pwd='or"='HTTP/1.0
- Resulting SQL query: SELECT * FROM Users WHERE users=" or "=" AND pwd=" or "="
- Since the WHERE clause is always true, the query returns all rows
- The attacker is allowed to "enter the system" and gets the identity of the first entry in the table Users

2.3.2 some more tricks

Suppose users can select car models via a drop-list:

- Selected entry is sent as a number to the web server
- GET/path/showcars.pl?brand_id=17; DROP TABLE Users HTTP/1.0
- Resulting SQL query: SELECT * FROM Cars WHERE brandID=17; DROP TABLE Users

SQL injection is powerful, but it is not always easy to carry out:

- One must first find out a vulnerable web application
- The internal structure of the database is normally unknown to the attacker
- The type of database the application uses, is mostly unknown
- Requires a good knowledge of SQL and of different database products
- Resulting SQL queries must be syntactically correct

Good strategy:
Insert a '(single quote) in web form fields: This usually breaks the SQL statements and (if we are lucky) gives information about the SQL via an error message;

2.3.3 Defenses against SQL injection

- On the server, **validate**, **constrain** and **sanitize** all data provided by the client (length restriction, disallow critical characters (single quote etc.))
- Do not use client parameters directly in SQL queries, but use prepared statements (type-checking, parameters are escaped by the DBMS, only one query is executed)
- A similar effect can be achieved by parameterized stored procedures
- Avoid disclosing detailed database error information to the client
- Access the database with **minimal** privileges

2.3.4 SQL statements' sanitation

Listing 2: PreparedStatement Pseudocode

```
updateSales = con.prepareStatement("UDATE Cars SET Name = ? WHERE id = ?");
updateSales.setString(1, e.getKey());
updateSales.setInt(2, e.getValue().intValue());
```

2.4 Cross-Site Scripting

2.4.1 XSS

Dynamically generated web pages send often data entered by the user in web forms back to the user. With XSS, an attacker exploits formulars:

- The attacker submits a JavaScript in a web form
- The dynamically generated web page contains the script
- When displayed in a browser, the script is executed
- Unlike SQL injection, the attack targets another user, not the server!
- Effective attack because (i) most browsers have JavaScript enabled and (ii) it is platform independent

If an attacker manages to execute JavaScript code in the browser of a victim, the following is possible:

- He can fetch arbitrary additional JavaScript code from a server
- He can retrieve the currently used session-id, which allows to take over the session by session hijacking (e-Commerce, e-Banking...)
- He can dynamically adapt the web page during the loading/rendering process in the victim's browser, e.g. to replace the login dialogue with an own version and to steal the victim's credentials

Difficulties for the attacker: To execute a JavaScript in the browser of another user, that user must send the JavaScript to the vulnerable web server

2.4.2 Defenses against XSS

Secure communication protocols, packet-filtering firewalls are useless. But: using HTTPS helps against Session Hijacking.
Server side: do the following:

- Validate all data provided by client
- Sanitize all client-provided data before sending them back to the browser
- Especially watch for <script>, </script>, javascript:
- Replace < and > around script of client-provided data with and browser interprets this as string literals and not as scripts

Client side: disable JavaScript

2.5 Regular Expressions

2.5.1 Cheet Sheet

Literal Characters	
\f	Form feed
\n	Newline
\r	Carriage return
\t	Tab
\v	Vertical tab
\a	Alarm (beep)
\e	Escape
\xHH	The ASCII character specified by the two digit hexadecimal code. For octal use \OOO except JS
\x{HHHH}	PHP: ASCII character represented by a four digit hexadecimal code. Javascript uses \uHHHH
\cX	The control character ^X. For example, \cI is equivalent to \t and \cJ is equivalent to \n

Character Classes							
[...]	Any one character between the brackets.						
[^...]	Any one character not between the brackets.						
.	Any character except newline. Equivalent to [^\n]						
\w	Any word character. Equivalent to [a-zA-Z0-9_] and [[:alnum:]_]						
\W	Any non-word character. Equivalent to [^a-zA-Z0-9_] and [^[:alnum:]_]						
\s	Any whitespace character. Equivalent to [\t\n\r\f\v] and [[:space:]]						
\S	Any non-whitespace. Equivalent to [^ \t\n\r\f\v] and [^[:space:]] Note: \w != \S						
\d	Any digit. Equivalent to [0-9] and [[:digit:]]						
\D	Any character other than a digit. Equivalent to [^0-9] and [^[:digit:]]						
[\b]	A literal backspace (special case)						
[[:class:]]	alnum	alpha	ascii	blank	cntrl	digit	graph
	lower	print	punct	space	upper	xdigit	

Replacement	
\	Turn off the special meaning of the following character.
\n	Restore the text matched by the nth pattern previously saved by \(and \). n is a number from 1 to 9, with 1 starting on the left.
&	Reuse the text matched by the search pattern as part of the replacement pattern.
~	Reuse the previous replacement pattern in the current replacement pattern. Must be the only character in the replacement pattern. (ex and vi).
%	Reuse the previous replacement pattern in the current replacement pattern. Must be the only character in the replacement pattern. (ed).
\u	Convert first character of replacement pattern to uppercase.
\U	Convert entire replacement pattern to uppercase.
\l	Convert first character of replacement pattern to lowercase.
\L	Convert entire replacement pattern to lowercase.

Repetition	
`{n,m}`	Match the previous item at least n times but no more than m times.
`{n,}`	Match the previous item n or more times.
`{n}`	Match exactly n occurrences of the previous item.
`?`	Match zero or one occurrences of the previous item. Equivalent to {0,1}
`+`	Match one or more occurrences of the previous item. Equivalent to {1,}
`*`	Match zero or more occurrences of the previous item. Equivalent to {0,}
`{}?`	Non-greedy match - will not include the following group/match characters.
`??`	Non-greedy match - will not include the following group/match characters.
`+?`	Non-greedy match - will not include the following group/match characters.
`*?`	Non-greedy match. E.g. `^(.*?)\s*$` the grouped expression will not include trailing spaces.

Options	
`g`	Perform a global match. That is, find all matches rather than stopping after the first match.
`i`	Do case-insensitive pattern matching.
`m`	Treat string as multiple lines: ^ and $ match internal \n
`s`	Treat string as single line: ^ and $ ignore \n, but . matches \n
`x`	Extend your pattern's legibility with whitespace and comments.

Extended Regular Expression	
`(?#...)`	Comment, "..." is ignored.
`(?:...)`	Matches but doesn't return "..."
`(?=...)`	Matches if expression would match "..." next
`(?!...)`	Matches if expression wouldn't match "..." next
`(?imsx)`	Change matching rules (see options) midway through an expression.

Grouping	
`(...)`	Grouping. Group several items into a single unit that can be used with *, +, ?, |, and so on, and remember the characters that match this group for use with later references.
`|`	Alternation. Match either the subexpressions to the left or the subexpression to the right.
`\n`	Match the same characters that were matched when group number n was first matched. Groups are subexpressions within (possibly nested) parentheses.

Anchors	
`^`	Match the beginning of the string, and, in multiline searches (/m), the beginning of a line. PHP: Use \A to match beginning of string in all line matching modes.
`$`	Match the end of the string, and, in multiline searches (/m), the end of a line. PHP: Use \z and \Z to match the end of a string or end of text respectively.
`\b`	Match a word boundary. That is, match the position between a \w character and a \W character. (Note, however, that [\b] matches backspace.)
`\B`	Match a position that is not a word boundary.

2.5.2 Example

Listing 3: email checker

```
  public class Email AddressChecker {
2   public static void main(String[] args) {
```

```
      // Checks for email addresses starting with
      // inappropriate symbols like dots or @ signs.
      Pattern p = Pattern.compile("^\\.|^\\@");
      Matcher m = p.matcher(args[0]);
      if (m.find()) System.err.println("Email addresses don't start with dorts or @ signs.");

      // Cekcs for email addresses that start with
      // www. and prints a message if it does.
      p = Pattern.compile("^www\\.");
      m = p.matcher(args[0]);
      if (m.find()) System.err.println("Email addresses don't start with www, only web pages"
          );

      p = Pattern.compile("[\\@] [\\.]+");
      m = p.matcher(args[0]);
      if (!m.find()) System.err.println("Emails must contain at most a @ and at least a .");

      p = Pattern.compile("^([A-Za-z0-9]+[-._+&])*[0-9a-zA-Z]+@([-0-9a-zA-Z]+[.])+[a-zA-Z
          ]{2,6}$");
      m = p.matcher(args[0]);
      Stringbuffer sb = new StringBuffer();
      boolean result = m.find();
      boolean deletedIllegalChars = false;
      while (result) {
        deletedIllegalChars = true;
        m.appendReplacement(sb, "");
        results = m.find();
      }

      // Add the last segment of input to the new String
      m.appendTail(sb);
      String input = sb.toString();
      System.out.println(input);
      if (deletedIllegalChars) {
        System.out.println("It contained incorrect characters, such as spaces or commas.");
      }
    }
  }
```

3 Robust Programming

3.1 Einleitung

Writing programs is not a trivial task. Most (large) programs that are written contain errors: that means that the program doesn't do what it's meant to do.

There are many sources of error:

- **Syntactic errors** (good compilers take care of that);
- **Semantic errors** (our brains are not perfect programming machines ...)
- Programs are "correct"but **input/output fail** and resource are not ready ! exception handling.
- **Languages' ambiguous specifications** (or their implementations) break the program (example: Java memory model and concurrency).

A good definition of robustness is therefore: **Graceful behaviour in presence of errors**
This means that if the program fails, it falls into a well known state and at least logs why it has failed.

3.2 Specific tools enhance robustness

The most useful techniques are:

- Judicious use of types and visibility attributes;
- Annotations (compile time);
- Correct use of the exception handling mechnisms (run time);
- Assertions (run time).

3.3 Types and visibility in Java

Rule 1 All fields are **private**

Rule 2 Fields, once initialized by the class constructor, are **never** updated.

Rule 1 + Rule 2 $\rightarrow$ the class is **immutable**!

3.4 Code inheritance breaks encapsulation

Against code inheritance:

1. Strong relationship between base and derived class
2. The extension of a class with sub classing, requires an in-depth knowledge about the implementation of the base class
3. Specialization interface means:
    - Protected methods: Disrupts the private/public only model.
    - Specialization semantics

Inheritance breaks encapsulation: Support for inheritance implies that (some) implementation details have to be published!

3.4.1 Instead inheritance use composition

The advantages of composition:

- Only the client interface is used
- No call-backs (in contrast to the inheritance based solutions)
- No dependency on the implementation

3.4.2 Instead inheritance use Template/Hooks

Template methods:

- Template method pattern provides a robust method to allow for extensions
- Base class provides extension points

3.5 Annotations

3.5.1 General remarks

Annotations provide data about a program that is not part of the program itself. They have no direct effect on the operation of the code they annotate.
Main usages:

- **Information for the compiler**: Annotations can be used by the compiler to detect errors or suppress warnings.
- **Compiler-time and deployment-time processing**: Software tools can process annotation information to generate code, XML files, and so forth.
- **Runtime processing**: Some annotations are available to be examined at runtime.

3.5.2 Why are annotations so important?

1. Annotations act mostly at compile-time: and this is a very good thing. We want to catch all of our subtle and not so subtle errors at compile time and not at run- time in front of an important customer ...
2. Annotations can be extracted by external tools. Instead of looking for methods with a particular name or signature, retrieve all methods with a specific annotation.
3. Annotations are like classes. They have a specific type. They can contain fields to store details.
4. Meta-specifications for annotations include:
   (a) Where they can appear on (e.g. only on classes, or only on methods)
   (b) A retention policy: when and where they are made available:

3.5.3 Overview

Meta
Im Paket java.lang.annotation - diese werden nur für die Definition von Annotationen gebraucht.

Annotation	Beschreibung
@Target	Dieser Annotationstyp wird bei der Programmierung einer Annotation angewandt. Damit wird festgelegt, auf welche Elemente eines Programms sie angewendet werden darf. Die möglichen Werte für eine Annotation dieses Typs sind in der Enumeration ElementType aufgeführt: TYPE, FIELD, METHOD, PARAMETER, CONSTRUCTOR, LOCAL_VARIABLE, ANNOTATION_TYPE, PACKAGE.
@Retention	Enumeration in RetentionPolicy. Specifies wheter the annotation is only available for: SOURCE: at compile time, discarded by compiler CLASS: stored in class file, discarded by VM RUNTIME: retained in class file and loaded by VM, accessible at runtime using reflection
@Inherited	The annotation get applied to subclasses as well (only CLASS annotation)
@Documented	Annotation should appear in the JavaDoc

Vordefinierte

Annotation	Beschreibung	Beispiel
@Deprecated	[RUNTIME] Generates compiler warnings	
@SupressWarnings	[SOURCE] Indicates that the named compiler warnings should be supressed	
@Override	[SOURCE] Indicates that a method is intended to override a method in a super class.	
@Resource	[RUNTIME] Any class or component that provides some useful functionality to an application can be thought of as a Resource and the same can be marked with @Resource annotation. The programmer specifies, how this resource must be accessed.	`@Resource(name = "MyQueue", type = javax.jms.Queue, shareable = false, authenticationType = Resource.AuthenticationType.CONTAINER, description = "A Test Queue" )` `private javax.jms.Queue myQueue;`
@PostConstruct	[RUNTIME] Used on a method that needs to be executed after dependency injection (Spring FW!) is done to perform any initialization. This method must be invoked before the class is put into service.	`@PostConstruct` `public void initIt() throws Exception { System.out.println("Init method after properties are set : " + message); }`
@PreDestroy	[RUNTIME] Used to release resources that it has been holding.	`@PreDestroy` `public void cleanUp() throws Exception { System.out.println("Spring Container is destroyed! Customer clean up"); } }`
@CheckReturnValue	Telling the compiler that it should issue a warning if the return value isn't used.	`@CheckReturnValue` `public String slimDownString() { return str.trim(); }`

@Nonnull	The annotated element must not be null. Annotated fields must not be null after construction has completed. Annotated methods must have non-null return values.	`public String myToUppercase(@NonNull String parameter) { return parameter.toUpperCase(Locale.getDefault()); }`
		`@NonNull`
		`public String getValue() { return this.value; }`
		`public void justUseGetValue() { int length = getValue().length(); System.out.println("Value length: " + length); }`

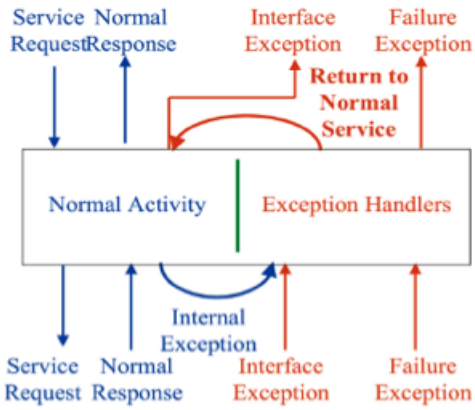
@CheckForNull	The annotated element might be null, and uses of the element should check for null. When this annotation is applied to a method it applies to the method return value.	`@CheckForNull`
		`public String value;`
		`public int getValueLength() { if (this.value == null) return 0; return this.value.length(); }`

Custom

Annotation	Beispiel
public @interface ToDo{public String value();}	@ToDo("improve quality")
public @interface Authors{public String[] names;}	@Authors(names={"Grz","Nc"})
public @interface RequestForEnhancements{ public int id(); public String assigned() default ""; public String date();}	@RequestForEnhancements(id=5, date="11.12.13")

3.6 exception handling

3.6.1 Ideal fault-tolerant software components



3.6.2 Requirements for exception handling

1. **Simplicity**: Should be simple to understand and use
2. **Unobtrusiveness**: Exception handling code should not obscure the understanding of the software: (1) and (2) are crucial for designing reliable systems!
3. **Efficiency**: Run-time overheads should be incurred only when handling an exception. This may be relaxed, e.g. if speed of recovery is not critical
4. **Uniformity**: Uniform treatment of exceptions detected both by the environment and by the program
5. **Recovery**: It should allow recovery actions

3.6.3 Classes of exceptions

Who detects the error?

- Environmental error detection
- Application error detection

When it is detected?

- **Synchronously**: as an immediate result of a process attempting an inappropriate operation
- **Asynchronously**: some time after the operation causing the error. May be raised either in the process which executed the operation or in another process. The latter is also called asynchronous notification or signal. Mostly an issue with concurrent programming.

3.7 Assertions

These three roles collectively support what is called the design-by-contract model of programming, a model that is well supported for example by the Eiffel programming language. Java doesn't have built-in support for the design-by-contract model of programming. But you can use the java.assertion package to enforce these rules.

3.7.1 Decleration

An assertion statement in Java takes one of the following two forms:

- assert condition:

- assert condition : error message;

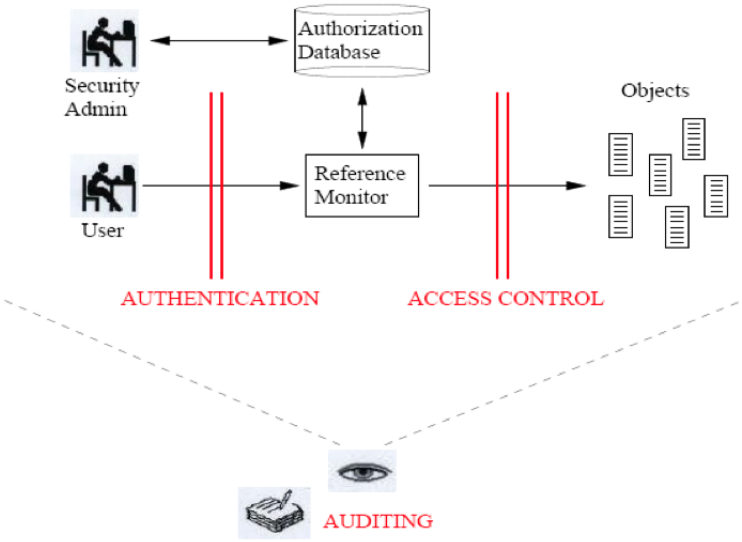
3.7.2 enable/disable

To enable assertions at various granularities, use the -enableassertions To disable assertions at various granularities, use the -disableassertions

4 Action Control

4.1 Goal

Specify the limits within which a legitimate user can work with the system.



4.2 AC policies

Subjects detain privileges on objects according to AC policies (models).
Policy: specifies how accesses are controlled and access decisions determined.

Mechanism (structure): implements or enforces a policy.

- Access matrix.
- AC list (ACL).
- Capability list.

4.2.1 Discretionary AC

This policy restricts access to system objects based on the identity of the users and/or the groups to which they belong.
Two types of DAC policies are used in practice:

- **Closed DAC**: authorization must be explicitly specified, since the default decision is always denial.
- **Open DAC**: denials must be explicitly specified since the default decision is always access.

Problems:

- Danger of users' mistakes or negligence;
- The dissemination of information is not controlled: for example a user who is able to read data can pass it to other users not authorized to read it without the actual owner knowing it.

4.2.2 Mandatory AC

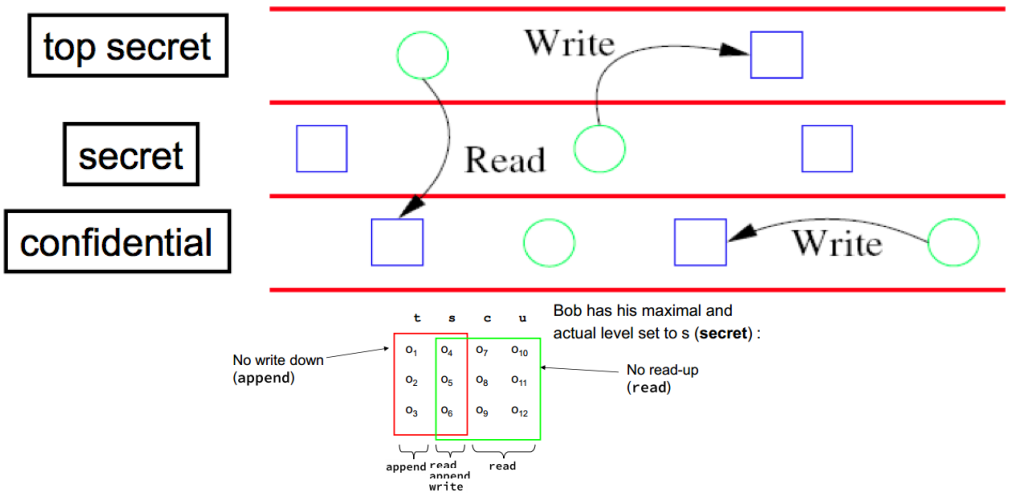
Idea: each subject and each object is assigned a security level.

1. Subject's label (clearance) specifies the level of trust associated with that subject.
2. Object's label specifies the level of trust that a subject must have to be able to access that file.

Pro: Reduction of possible damage
Contra: Loss of flexibility.

4.3 Bell-LaPadula (BLP) policy model

- The Simple Security Policy: no process may read data at a higher level. **No Read Up (NRU)** or: a subject can only read an object of less or equal security level.
- The *-property: no process may write data to a lower level. **No Write Down (NWD)** or: a subject can only write into an object of greater or equal security level.



4.4 Other policy concepts

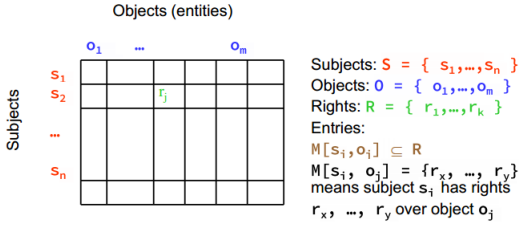
- Seperation of Duty: Check is only valid if signed y two authorized people
- Chinese Wall Policy

4.5 Role-based access control (RBAC)

Permissions are associated with:

- roles (= set of actions and responsibilities, associated with particular working activities)
- users/subjects

4.6 ACM/ACL



Right to copy: Two rights in stead of one Read (r / rc). Attenuation of privilege: You can't give away rights you do not possess. This principle is usually ignored for the owner of an object. **Summary**

- Access control matrix is the simplest abstraction mechanism for representing a protection state within a system.
- Transitions alter the protection state.
- Six primitive operations alter the AC matrix: Transitions can be expressed as commands composed of these operations and, possibly , conditions.

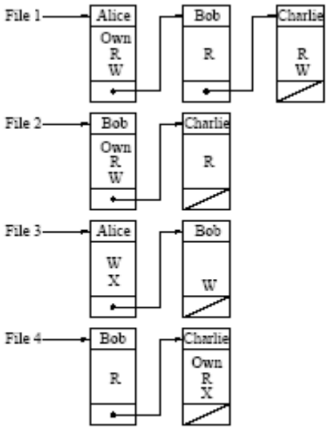
4.7 AC list (ACL)

In large systems the matrix will be big and sparse. Therefore in practice, the matrix is implemented as a list.

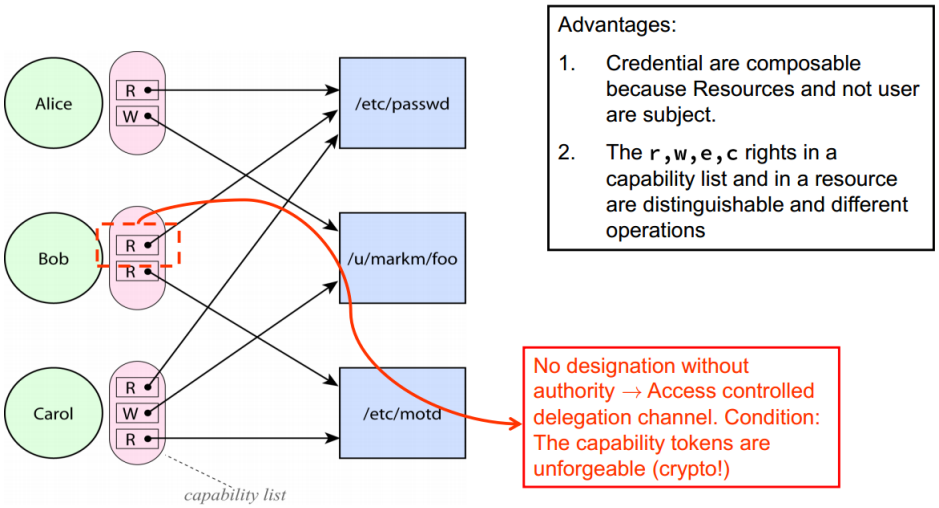
AC matrix

	File 1	File 2	File 3	File 4	Account 1	Account 2
Alice	Own R W		W X		Inquiry Credit	
Bob	R	Own R W	W	R	Inquiry Debit	Inquiry Credit
Charlie	R W	R		Own R X		Inquiry Debit

AC list (ACL)



4.8 Capability list



Delegation: Process can pass capability at run time **Revocation:**

- Only if OS knows which data is associated with a capability. If capability is used for multiple resources, one has to revoke all or none.
- Indirection: capability points to pointer to resource (C=¿P=¿R, set P=0 to revoke)

4.9 Android

4.9.1 Basic Components of an Application

- Activity: User Interface
- Service: Java daemon
- Content provider: Store and share data
- Broadcast receiver: For messages from other applications
- Intents: asynchronous mesaging system

4.9.2 Security steps

- Signing with certificates
- Application sandbox: runs with its UID and own Dalvik virtual machine
- Linux and ACL
  - Each application executes with its own user identity as Linux process
  - Android middleware has a reference monitor that mediates the establishment of inter-component communication (ICC)

4.9.3 Android manifest file

Focus on Inter Component Communication (ICC) whose security policy is defined in the Android manifest file. Allows developers to specify a high-level ACL to access the components:

- Each component can be assigned an access permission label
- Each application requests a list of permission labels

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
      ...
<application android:icon="@drawable/icon“android:label="@string/app_name">
  <activity android:name=".BatteryCheck"android:label="@string/app_name">
    <intent-filter>
      <action android:name="android.intent.action.MAIN" />
      <category android:name="android.intent.category.LAUNCHER" />
    </intent-filter>
    <intent-filter>
      <action android:name="ch.fhnw.android.sms.receiver.BATTERY_SMS" />
      <category android:name="android.intent.category.DEFAULT" />
    </intent-filter>
  </activity> </application>
<uses-permission android:name="android.permission.BATTERY_STATS" />
<uses-permission android:name="android.permission.SEND_SMS"></uses-permission>
<uses-permission android:name="android.permission.RECEIVE_SMS"></uses-permission>
</manifest>
```

- Components can be public or private (exported attribute). It specifies whether the activity can be launched by components of other applications. Default value is not reliable, set it always explicitly.
- If the manifest does not specify an access permission, any component in any application can access it. Components without access permissions should be exceptional cases, and possible inputs must be carefully analysed (consider splitting components).
- If no permission label is set on a broadcast, any unprivileged application can read it. Always specify an access permission on Intent broadcasts (unless you specify the destination explicitly).

Listing 4: Intent broadcast permissions

```
Intent resultIntent = new Intent();
String action = "ch.fhnw.android.battery.BATTERY_SMS";
resultIntent.setAction(action);
resultIntent.putExtra("Sending Activity", "Battery Check");
resultIntent.putExtra("Battery Info", batteryRcv.batteryInfo);
sendBroadcast(resultIntent);
```

Android content provider permissions

- seperate ”read and ”write”
- URI permissions to specify which data subsets of the parent content provider permission can be granted for
- Always define separate read and write permissions. Use URI permissions to delegate right to other components.

```
<provider android:authorities="friends"
        android:name="FriendProvider"
        android:readPermission="“ch.fhnw.apsi.permission.READ_FRIENDS"
        android:writePermission="ch.fhnw.apsi.permission.WRITE_FRIENDS">
</provider>
```

- The application developer can add reference monitor hooks
- Use checkCallingPermission() to mediate ädministrativeöperations. Alternatively, create separate Services.

```
private final IFriendTracker.Stub mBinder = new IFriendTracker.Stub() {
  public boolean isTracking() {return mTracking;}
  public boolean addNickname(String nick, int contactId) {
   if (FriendTracker.this.checkCallingPermission(PERMISSION_FRIEND_SERVICE_ADD)
                                    != PackageManager.PERMISSION_GRANTED) {
    throw new SecurityException("Requires " + PERMISSION_FRIEND_SERVICE_ADD);
  } ... }
};
```

Permission categories: normal, dangerous, signature, signatureOrSystem

- Use signature permissions for dangerous permissions. Otherwise and always include informative descriptions

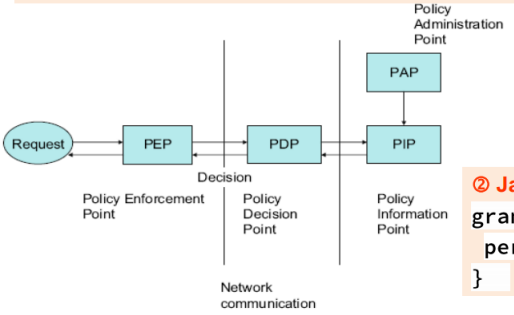
```
<permission android:name="ch.fhnw.apsi.permission.FRIEND_NEAR“
        android:label="@string/permlab_friendNear"
        android:description="@string/permdesc_friendNear“
        android:protectionLevel="dangerous">
</permission>
```

Defined in string.xml

5 JavaIdiocies

5.1 Java Security Model

④ Java PEP:
`SecurityManager, Permission, AccessController, AccessControlContext, ProtectionDomain` management.



① Java PAP → File: `java.security`

② Java PIP → File: `java.policy`

```
grant codeBase …{
 permission.java.security.AllPermissions;
}
```

③ Java PDP:

```
if (sm != null){
 context = sm.getSecurityContext();
 Filepermission p = new Fileprmission(filename, “read”);
 sm.checkPermission(p,context);
}
```

5.2 Visibility modifiers

(default, private, protected, public)

5.2.1 Purpose

- Information hiding
- Compartmentalization of program's components (things you should know (public) and things you can safely ignore(private))

5.2.2 Attention

- Java allows increasing the visibility of a member in subclasses
- protected is unsafe: subclassing and a class can claim to have the same package.

5.3 Protecting packages

5.3.1 Sealed JAR archives

Add header in the manifest. All classes in a package must be found in the same JAR file.

5.3.2 Signing JAR archives

Sending point: jarsigner -keystore x -signedJar sCount.jar Count.jar signFiles
keytool -export -keystore susanstore -alias signFiles -file SusanJones.cer
Receiving end:

```
grant CodeBase "file:./..." SignedBy "Susan" {
  permission java.io.FilePermission "C:\\TestData\\*", "read";
}
VM-Argumente:
java -Djava.security.manager -Djava.security.policy=raypolicy -cp sCount.jar Count C:\
    TestData\data
```

5.3.3 Join packages via security policy PAP

Can insert in the java.security file:
package.definition=com.ibneco.securepackage
Attention: no class loaders from Sun support this at present

5.4 Inner classes

Inner classes are not understood by JVM:

```
class A {
  private int m;
  private class B {
    private int x;
    void f() {
      x = m;
    }
  }
  public void g() {
    B ob = new B();
    ob.f();
  }
}
```

After compilation:

```
class A {
  private int m;
  public void g() {
    A$B ob = new A$B();
    ob.f();
  }
  int access$0(){
    return m;
  }
}
```

1. access$0() of class A has package level visibility.
2. The class A$B also has package level visibility

```
class A$B {
  A this$0;
  private int x;
  void f() {
    x = this$0.access$0();
  }
}
```

Other classes belonging to the same package can call the access function and tamper the private data member.

5.5 Common Java Antipatterns

5.5.1 Assuming objects are immutable

Object[] signers;
public Object[] getSigners()return signers;
Problem: We quickly forget that mutable input and output objects can be modified by the caller application.
Solution: Make a copy of mutable I/O parameters.

5.6 Basing security checks on untrusted sources

Signature (final java.io.File f) ... f.getPath() ...
Problem: Do not forget that security checks can be always fooled if they are based on information that attackers can control!

- You can not trust libraries
- non-final classes/methods can be sub classed
- mutable types can be modified

Solution: Make copys of inputs or make a copy of the source of the input. Never invoke doPrivileged() with caller-provided inputs.

5.7 False inheritance relationship

Problem: class Stack¡E¿ extends Vector¡E¿. Stack is not a Vector but all Methods that can be applied to a Vector can now applied to the Stack.

5.8 Ignoring changes to super classes

Problem: Changes to a super class may affect its sub classes which leads to unwanted behaviour or misuse of inherited methods.
Solution: Subclass only when the inheritance model is well-specified and well-understood. Monitor changes to super classes.

5.9 Neglecting to validate inputs

Problems: Invalidated inputs can be out-of-bounds values, escape characters, can affect processing of requests or delegating them to sub-components (SQL). Additional issues when calling native methods.
Solution: Validate all inputs

- Check for escape characters
- Check for out-of-bounds values
- Check for malformed requests
- Regex API can help validate String inputs
- Wrap native methods in Java language wrapper to validate inputs
- Make all natives method private

5.10 Misusing public static variables

Problem: Static variables can be modified or misused as a communication channel globally across a Java runtime environment.
Solution: Reduce the scope of static fields and define acces methods (with securityManagerCheck();). Treat public static fields primarly as constants (final).

Listing 5: Possible implementation of securityManagerCheck()

```
private void securityManagerCheck() {
  SecurityManager sm = System.getSecurityManager();
  if (sm != null) {
    sm.checkPermission(...);
  }
}
```

5.11 Believing a constructor exception destroys the object

Problem: If the constructor throws an exception, a reference on it still exists on the uninitialized object in the heap. Java 1.7 destroys automatically objects whose constructor has thrown an exception.
Solution: Make the class final if possible. If the finalize method can be overridden, ensure that partially initialized instances are unusable (initialized-flag).

5.12 TOC2TOU - Time Of Check To Time Of Use

Example: Check for a file-permission is done at the opening but never ever checked again, even if privileged actions are done. That is typical for Unix systems.
Solution: Always check the user's permission while executing a privileged operation.

5.13 Twelve (very conservative) guidelines for writing safer Java

- Do not depend on implicit initialization
- Limit access to entities
- Make everything final
- Do not depend on package scope
- Do not use inner classes
- Avoid signing your code. Code that isn't signed will run without special privileges i.e. less likely to do damage!
- If you must sign, put all signed code in one jar archive
- Make classes uncloneable. Java's object cloning mechanism allows an attacker to build new instances of the classes you define, without using any of your constructors.
- Make classes unserializeable
- Make classes undeserializeable
- Do not compare classes by name
- Do not store secrets