NTNU
Institutt for datateknikk
og informasjonsvitenskap

Exam

TDT4237 – Software Security (Programvaresikkerhet)

Monday December 10th 2012, 15:00 - 19:00

Oppgaven er utarbeidet av faglærer Lillian Røstad og kvalitetssikrer er Torbjørn Skramstad.
Kontaktperson under eksamen er Lillian Røstad (mobil 994 00 628)

Språkform: Engelsk/Bokmål/Nynorsk

Tillatte hjelpemidler: D

Ingen trykte eller håndskrevne hjelpemidler tillatt. Bestemt, enkel kalkulator tillatt.
No printed or hand-written materials allowed. Approved calculator allowed.

Results available (sensurfrist): Thursday January 10th 2013

Read each task carefully. Identify what the task asks for.
If you find that information is missing in a task you are free to make the assumptions you consider necessary, but remember to explain and clearly mark your assumptions.

Les oppgaveteksten nøye. Finn ut hva det spørres etter i hver oppgave.
Dersom du mener at opplysninger mangler i en oppgaveformulering gjør kort rede for de antagelser og forutsetninger som du finner det nødvendig å gjøre.

ENGLISH

Task 1 (30%)

a) Compare XSS and CSRF. For both of these attacks, explain why they are possible (vulnerability) and how they are performed - using an example.

XSS – cross-site scripting is an attack type that exploits lack of output validation in an application. Typically, XSS-vulnerabilities may exist wherever it is possible for the user to store information in an application, common examples include user-reviews and comments. These are all examples of stored-XSS. The problem occurs if the user is allowed to store cod. This code will be executed when the next user visits the same page.
CSRF – cross-site request forgery – is an attack that exploits the fact that a user has a valid session in one application, while simultaneously visiting another (malicious/infected) application. The malicious application (or site) misuses the users valid session to perform illegitimate transactions. The most common example is that of an online bank: a user is logged in to the online bank, then visits a malicious site that site tricks the user into performing payment transactions.

b) Explain the Needham-Schroeder protocol.

There are actually two Needham-Schroeder protocols – symmetric and public key. The expected answer here is the one we focused on in class (symmetric), and that the book presents.

The purpose is to establish a shared secret, a key. Alice and Bob are the communication parties. Nounces are used instead of timestamps, and a trusted server (S) is used as TTP (trusted third party). The TTP creates the session key. Both Alice and Bob already have a shared secret key with the TTP-server (S). There are five steps to Needham-Schroeder:

Message 1 $A \rightarrow S : A, B, N_A$
Message 2 $S \rightarrow A : \{N_A, B, K_{AB}, \{K_{AB}, A\}_{K_{BS}}\}_{K_{AS}}$
Message 3 $A \rightarrow B : \{K_{AB}, A\}_{K_{BS}}$
Message 4 $B \rightarrow A : \{N_B\}_{K_{AB}}$
Message 5 $A \rightarrow B : \{N_B - 1\}_{K_{AB}}$

There is a subtle problem with this protocol: Bob has to assume that the key KAB he receives from Sam (via Alice) is fresh. This is not necessarily so: Alice could have waited a year between steps 2 and 3. In many applications this may not be important; it might even help Alice to cache keys against possible server failures. But if an opponent—say Charlie—ever got hold of Alice's key KAS he could use it to set up session keys with many other principals.

Revocation is a problem: Sam will probably have to keep complete logs of everything he has ever done, and these logs would grow in size forever unless the principals'

names expired at some fixed time in the future.

Over 20 years later, this example still generates controversy in the security protocols community. The simplistic view is that Needham and Schroeder just got it wrong; the view argued by Susan Pancho and Dieter Gollmann (for which I have much sympathy) is that this is one more example of a protocol failure brought on by shifting assumptions [345, 600]. 1978 was a kinder, gentler world; computer security then concerned itself with keeping the bad guys out, while nowadays we expect the bad guys to be users of the system. The Needham-Schroeder paper explicitly assumes that all principals behave themselves, and that attacks come only from outsiders [589]. Under these assumptions, the protocol remains sound.

c) Compare DAC (Discretionary Access Control) and MAC (Mandatory Access Control) – what are the main differences?

Mandatory Access Control – is system enforced, typically objects are classified into different levels, and subjects are assigned clearance levels. A subject needs clearance to a specific security level to be granted access to objects classified at that level. Typical example is military grade systems.
Discretionary Access Control – it is the owner (typically the creator) of an object that decides which subjects should be granted access to that object. The most common example is Unix-style file systems.

d) What is the simple security property?

The simple security property is one of the rules of the Bell-LaPadula model.

- *The Simple Security Property - a subject at a given security level may not read an object at a higher security level (**no read-up**).*

e) Compare the Biba and Bell-LaPadula policy models – what are the main differences?

Biba is a security policy model focused on integrity, while Bell-LaPadula focuses on confidentiality. They have similar sounding– but reverse in effect – security rules.

The Bell-LaPadula security rules:

- *The Simple Security Property - a subject at a given security level may not read an object at a higher security level (**no read-up**).*
- *The star-property - a subject at a given security level must not write to any object at a lower security level (**no write-down**).*

The Biba security rules:

- *A subject at a given level of integrity must not read an object at a lower integrity level (**no read down**).*
- *A subject at a given level of integrity must not write to any object at a higher level of integrity (**no write up**).*

f) List the seven touchpoints of software security, in order of effectiveness.

1. *Code review*
2. *Architectural risk analysis*
3. *Penetration testing*
4. *Risk-based security tests*
5. *Abuse cases*
6. *Security requirements*
7. *Security operations*

Task 2 (30%) – Code Quiz

On the next page is a set of Java-based code excerpts. For each case your task is to identify security vulnerabilities in the code.

For every vulnerability you identify, you shall:

1. Explain how it could be exploited, and
2. Explain how you would fix the code. This part of your answer should include the corrected code.

Case 1 :

```
import java.security.MessageDigest;

 public byte[] getHash(String password) throws NoSuchAlgorithmException {
       MessageDigest digest = MessageDigest.getInstance("SHA-1");
       digest.reset();
       byte[] input = digest.digest(password.getBytes("UTF-8"));
       return input;
 }
```

This is an example from OWASP of insecure use of hashing function. The problem is not adding a salt/key/random number – to prevent identical hashes for identical passwords. Not using a salt makes it easier to perform a brute force attack. Improved code:

```
import java.security.MessageDigest;


public byte[] getHash(String password, byte[] salt) throws
NoSuchAlgorithmException {
       MessageDigest digest = MessageDigest.getInstance("SHA-256");
       digest.reset();
       digest.update(salt);
       return digest.digest(password.getBytes("UTF-8"));
 }
```

More info:
https://www.owasp.org/index.php/Hashing_Java

Case 2:

```
public final Connection getConnection() throws SQLException {
  return DriverManager.getConnection(
      "jdbc:mysql://localhost/dbName",
      "username", "password");
}
```

Hard coding sensitive information, such as passwords, server IP addresses, and encryption keys can expose the information to attackers. Anyone who has access to the class files can decompile them and discover the sensitive information. Consequently, programs must not hard code sensitive information.

Hard coding sensitive information also increases the need to manage and accommodate changes to the code. For example, changing a hard-coded password in a deployed program may require distribution of a patch. This solution code reads the user name and password from a configuration file located in a secure directory.

```
public final Connection getConnection() throws SQLException {

  String username;
  String password;
  // Username and password are read at runtime from a secure config file
  return DriverManager.getConnection(
      "jdbc:mysql://localhost/dbName", username, password);
}
```

More info:

https://www.securecoding.cert.org/confluence/plugins/viewsource/viewpagesrc.action?pageId=25624745

Case 3:

```
import java.util.Random;

Random number = new Random(123L);
//...
for (int i = 0; i < 20; i++) {
  // Generate another random integer in the range [0, 20]
  int n = number.nextInt(21);
  System.out.println(n);
}
```

Pseudorandom number generators (PRNGs) use deterministic mathematical algorithms to produce a sequence of numbers with good statistical properties. However, the sequences of numbers produced fail to achieve true randomness. PRNGs usually start with an arithmetic seed value. The algorithm uses this seed to generate an output value and a new seed, which is used to generate the next value, and so on.

The Java API provides a PRNG, the `java.util.Random` class. This PRNG is portable and repeatable. Consequently, two instances of the `java.util.Random` class that are created using the same seed will generate identical sequences of numbers in all Java implementations. Seed values are often reused on application initialization or after every system reboot. In other cases, the seed is derived from the current time obtained from the system clock. An attacker can learn the value of the seed by performing some reconnaissance on the vulnerable target and can then build a lookup table for estimating future seed values.

Consequently, the `java.util.Random` class must not be used either for security-critical applications or for protecting sensitive data. Use a more secure random number generator, such as the `java.security.SecureRandom` class. Improved code:

```
import java.security.SecureRandom;
import java.security.NoSuchAlgorithmException;
// ...

public static void main (String args[]) {
   try {
     SecureRandom number = SecureRandom.getInstance("SHA1PRNG");
     // Generate 20 integers 0..20
     for (int i = 0; i < 20; i++) {
       System.out.println(number.nextInt(21));
     }
   } catch (NoSuchAlgorithmException nsae) {
     // Forward to handler
   }
}
```

More info:
https://www.securecoding.cert.org/confluence/plugins/viewsource/viewpagesrc.action?pageId=23724440

Case 4:

```
class SecurityIOException extends IOException {/* ... */};

try {
  FileInputStream fis =
      new FileInputStream(System.getenv("APPDATA") + args[0]);
} catch (FileNotFoundException e) {
  // Log the exception
  throw new SecurityIOException();
}
```

Failure to filter sensitive information when propagating exceptions often results in information leaks that can assist an attacker's efforts develop further exploits. An attacker may craft input arguments to expose internal structures and mechanisms of the application. Both the exception message text and the type of an exception can leak information. For example, the `FileNotFoundException` message reveals information about the file system layout, and the exception type reveals the absence of the requested file.

This noncompliant code example logs the exception and throws a custom exception that does not wrap the `FileNotFoundException`. While this exception is less likely to leak useful information, it still reveals that the specified file cannot be read. More specifically, the program reacts differently to nonexistent file paths than it does to valid ones, and an attacker can still infer sensitive information about the file system from this program's behavior. Failure to restrict user input leaves the system vulnerable to a brute force attack in which the attacker discovers valid file names by issuing queries that collectively cover the space of possible file names. File names that cause the program to return the sanitized exception indicate nonexistent files, while file names that do not return exceptions reveal existing files.

```
class ExceptionExample {
  public static void main(String[] args) {

    File file = null;
    try {
      file = new File(System.getenv("APPDATA") +
             args[0]).getCanonicalFile();
      if (!file.getPath().startsWith("c:\\homepath")) {
        System.out.println("Invalid file");
        return;
      }
    } catch (IOException x) {
      System.out.println("Invalid file");
      return;
    }

    try {
      FileInputStream fis = new FileInputStream(file);
    } catch (FileNotFoundException x) {
      System.out.println("Invalid file");
      return;
    }
  }
}
```

This compliant solution implements the policy that only files that live in `c:\homepath` may be opened by the user and that the user is not allowed to discover anything about files outside this directory. The solution issues a terse error message when the file cannot be opened or the file does not live in the proper directory. Any information about files outside `c:\homepath` is concealed.

More info:
https://www.securecoding.cert.org/confluence/plugins/viewsource/viewpagesrc.action?pageId=18186675

Case 5:

```
private void createXMLStream(BufferedOutputStream outStream,
                             String quantity) throws IOException {
  String xmlString;
  xmlString = "<item>\n<description>Widget</description>\n" +
              "<price>500.0</price>\n" +
              "<quantity>" + quantity + "</quantity></item>";
  outStream.write(xmlString.getBytes());
  outStream.flush();
}
```

Because of its platform independence, flexibility, and relative simplicity, the extensible markup language (XML) has found use in applications ranging from remote procedure calls to systematic storage, exchange, and retrieval of data. However, because of its versatility, XML is vulnerable to a wide spectrum of attacks. One such attack is called XML injection.

A user who has the ability to provide structured XML as input can override the contents of an XML document by injecting XML tags in data fields. These tags are interpreted and classified by an XML parser as executable content and, as a result, may cause certain data members to be overridden.

In this code example, a client method uses simple string concatenation to build an XML query to send to a server. XML injection is possible because the method performs no input validation.
Depending on the specific data and command interpreter or parser to which data is being sent, appropriate methods must be used to sanitize untrusted user input. This compliant solution uses whitelisting to sanitize the input.
In this solution, the method requires that the quantity field must be a number between 0 and 9.

```
private void createXMLStream(BufferedOutputStream outStream,
                             String quantity) throws IOException {
  // Write XML string if quantity contains numbers only.
  // Blacklisting of invalid characters can be performed
  // in conjunction.

  if (!Pattern.matches("[0-9]+", quantity)) {
    // Format violation
  }

  String xmlString = "<item>\n<description>Widget</description>\n" +
                     "<price>500</price>\n" +
                     "<quantity>" + quantity + "</quantity></item>";
  outStream.write(xmlString.getBytes());
  outStream.flush();
}
```

More info:
https://www.securecoding.cert.org/confluence/plugins/viewsource/viewpagesrc.action?pageId=34669118

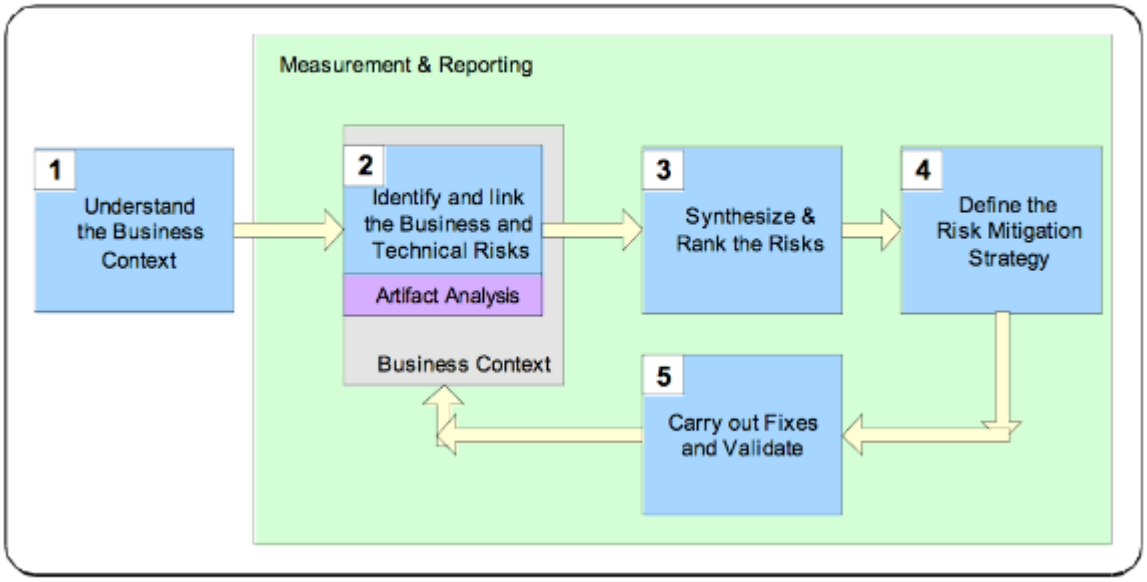
Task 3 (20%) – E-voting

Several countries around the world have in recent years explored and experimented with e-voting. The basic idea behind e-voting is to allow votes to be cast online in elections, and the motivation is of course that making voting easier and more accessible will help increase participation in elections.

Security concerns is the number one reason why e-voting is still not widespread. Your company still believes e-voting is the future, and is eager to develop a product for online voting. Your task is to figure out how this can be done, and what security requirements such a product needs to fulfill

Use the Risk Management Framework (RMF) and perform one iteration for your e-voting system. The result from this initial iteration of the RMF should be a set of security requirements.

Figure 1. The BSI risk management framework



Expected: completing steps 1-4 of the RMF (not expecting step 5). As the text states the outcome should be security requirements. We have discussed in class how in such an early stage of a project, when nothing has been created yet (no design, no code), the outcome of step 4 will be security requirements for the system. As always: it is important both to show knowledge of the method, but also be able to execute it and demonstrate knowledge of vulnerabilities, threats and attacks to create useful results when executing the RMF. Both method and content/results counts.

Task 4 (20%) – Futuristic: computer science exams on computers

Security concerns is also the number one reason why exams like the one you are working on right now is still performed using pen and paper. Suppose that NTNU have selected three potential systems to be used to provide exams such as this one on computers. As part of the evaluation process for the candidate systems, NTNU has created a hacking contest. Students are invited to create teams and they will have access to all three systems for one month. The systems are deployed online, and are available for testing to anyone with a valid NTNU-account. A key feature of all systems is that they should be able to automatically detect plagiarism.

The grand prize is tickets and travel expenses for the entire team to attend DEF CON in Las Vegas, one of the world's largest hacking conventions.

Your team really wants to win. You realize that you need a plan, because as the quote goes - *failing to plan equals planning to fail!*

Create a test-plan that will help your team win this contest. Explain the techniques you use.

Expected: a risk-based test plan. Do not expect the RMF on this example, though using it is not wrong. Expecting a more simplified approach working from threat-modeling and light-weight risk-analysis (either included in the threat models and/or summarized in a table). The method here is important, but several approaches are acceptable. What is really important is: that the resulting test plan is:

- *Relevant to the case – simply using OWASP Top 10 is not a good answer*
- *Prioritized – and the risk analysis should be the foundation for that*