NTNU
Institutt for datateknikk
og informasjonsvitenskap

Eksamensoppgave i/Exam

TDT4237 – Programvaresikkerhet/Software security

Onsdag/Wednesday 9. Desember/December 2009, kl. 09:00 - 13:00

Oppgaven er utarbeidet av faglærer Lillian Røstad og kvalitetssikrer Torbjørn Skramstad.
Kontaktperson under eksamen er Torbjørn Skramstad (mobil 971 23 246)

Språkform: Bokmål/Nynorsk/Engelsk
Tillatte hjelpemidler: D
Ingen trykte eller håndskrevne hjelpemidler tillatt./No printed or hand-written materials allowed.
Bestemt, enkel kalkulator tillatt./Approved calculator allowed.

Sensurfrist (results available): Mandag 11. Januar 2010 (Monday January 11th 2010)

SOLUTION PROPOSAL

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.

Read each task carefully. Identify what the task asks for.

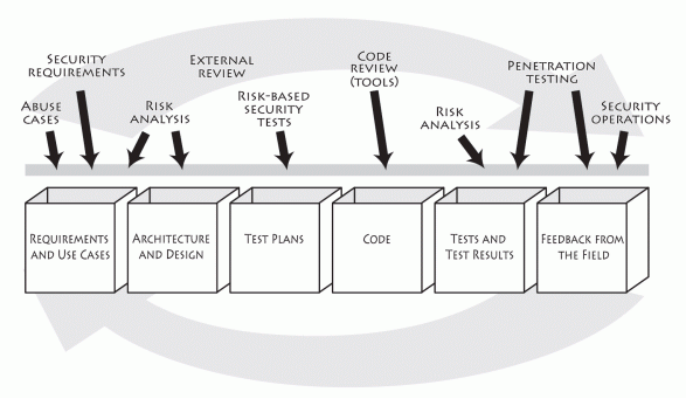
If you find that information is missing in a task you are free to make the assumptions you consider necessary, but remember to explain and clearly mark your assumptions.

Task 1 (25%)

a) What are the different software security touchpoints? Give a brief explanation of each touchpoint.

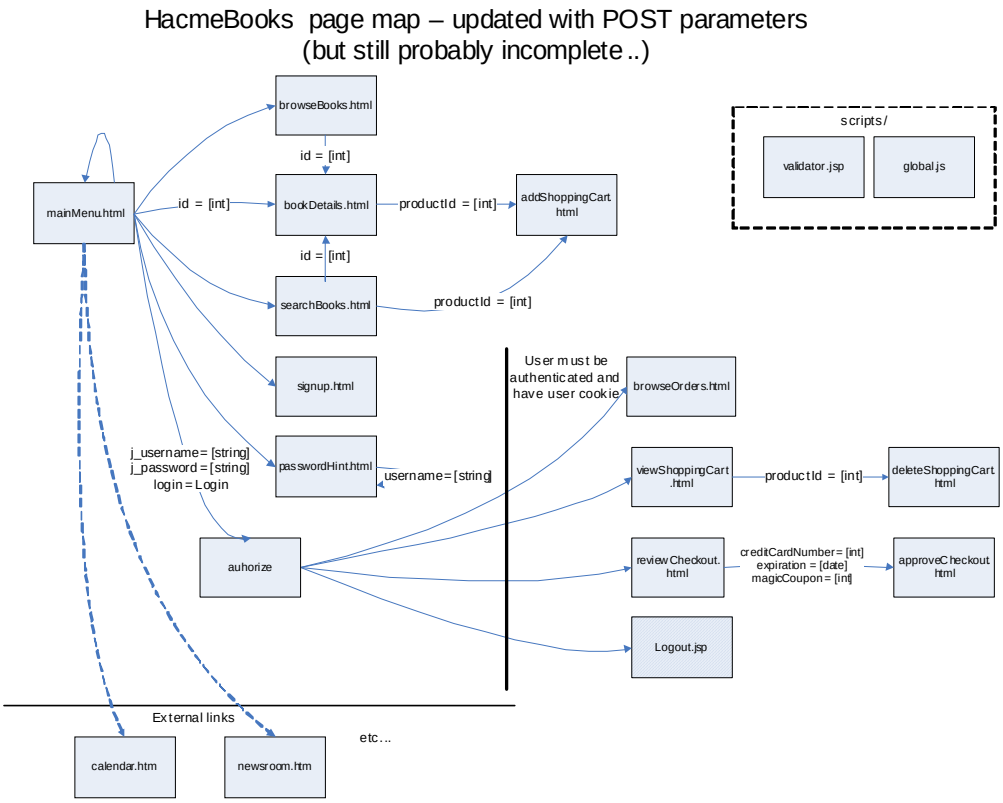
The touchpoints (as defined in "Building Security in") – in order of effectiveness:

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



b) What do you think is the most important touchpoint? Why?
This is a matter of opinion and grading depends on the argument made. The most logical one would be number 1 on McGraw's list (code review). Arguments from the book: whichever development process you follow, one thing will always be produced in a software project: code – which means that you can always apply this touchpoint. It is also good to make the distinction between automatic and manual code review in terms of required effort and payoff. Even if you have very little time and resources, there are numerous tools (many open source and free) out there that can be used for automatic code review. Manual code review ***can*** *be extremely valuable, but it is also very resource-demanding and quality of result depends heavily on the knowledge of the people performing the code review. Tools incorporate knowledge and can be useful to all developers, though they are notoriously prone to false positives and it takes a skilled developer to make the decision whether a reported "possible problem" is an issue or not. Also note that a combination is possible: one should always use tools for automatic code review, and one can in addition perform more in-depth manual reviews of parts of code that has been identified as critical.*

c) What is a pagemap used for? Illustrate with an example.
Ref. to attack #1 in "How to break" – "Panning for gold". A page map is used to gain knowledge about an application. It is typically used when you only have access to the application, and no code or documentation, to map out all the pages included in the application to get an overview of structure and trust zones. Page map can be created by manually navigating the application and make a note of each page found but there are also tools (like wget and a proxy) that can be used to automate the process of discovering pages. Example pagemap used in the lectures included on next page.



d) Explain the path traversal attack. What can an attacker accomplish using this attack?
This attack is about bypassing normal control-flow in the application. It typically relies on knowledge about the application found using "guessing files and directories". The attacker typically wants to try to access restricted parts of the application or gain access to restricted files by directly accessing them.

e) Assume that you are working on a large software development project. For various reasons you **want to** add vulnerabilities to the code. How would you do this? Focus on maximum damage potential and minimal chance of getting caught.
There are many possible answers here, including:

- *Utilize insight into the projects prioritizations to hide vulnerabilities in parts of the code that is least likely to be subject to thorough code review*
- *Utilize your knowledge of the automatic security testing tools being used in the project to add vulnerabilities that are unlikely to be discovered*
- *OWASP has an article full of tips: "How to write insecure code"*
    - *http://www.owasp.org/index.php/How_to_write_insecure_code*

Task 2 – Code Quiz – Spot the bug (25%)

These code snippets illustrate vulnerabilities in Java code. You shall perform a code review to identify any security-relevant issues in each snippet of code.
For each issue you identify, you are expected to: explain what vulnerability the error may cause, classify the error using McGraw's taxonomy and suggest a solution. You should illustrate your solution using (pseudo)code.

SOLUTION:

All code snippets are from:

The CERT Sun Microsystems Secure Coding Standard for Java

Code snippet 1:

In this noncompliant code example, the field `itemsInInventory` can be accessed by multiple threads. However, when a thread is updating the value of `itemsInInventory`, it is possible for other threads to read the original value. This is because read and write are sequential operations and the post decrement operator is non-atomic.

As a result of inadequate load testing, such issues are usually discovered when the system goes into production mode.

```
private int itemsInInventory = 100;

public int removeItem() {
  if(itemsInInventory > 0) {
    return itemsInInventory--;  // Returns new count of items in inventory
  } else {
    return 0;
  }
}
```

This compliant solution demonstrates synchronization in practice by using the `synchronized` keyword in the method definition. Access to `itemsInInventory` is now not only mutually exclusive but also consistent across object states.

```
private int itemsInInventory = 100;

public synchronized int removeItem() {
  if(itemsInInventory > 0) {
    return itemsInInventory--;  // Returns new count of items in inventory
  } else {
    return 0;
  }
}
```

An alternative is to declare the variable `itemsInInventory` as `volatile`. This ensures that the value read is the most recently updated one. If a volatile variable resides in another class, depending on its type it can be updated by using the `AtomicIntegerFieldUpdater`, `AtomicLongFieldUpdater`, and `AtomicReferenceFieldUpdater` classes [JavaThreads 04]. To use a Field Updater, invoke the `newUpdater` method that takes three arguments, `tclass` - the class of the objects holding the field, `vclass` - the class of the field and `fieldName` - the name of the field to be updated.

Code snippet 2:

This noncompliant code example locks on a non-final object that is declared `public`. It is possible that untrusted code can change the value of the lock object and foil any attempts to synchronize.

```
public Object publicLock = new Object();
synchronized(publicLock) {
  // body
}
```

This compliant solution synchronizes on a `private final` object and is safe from malicious manipulation.

```
private final Object privateLock = new Object();
synchronized(privateLock) {
  // body
}
```

Code snippet 3:

This noncompliant code example accepts a file name as an input argument. An attacker can gain insights into the structure of the underlying file system by repeatedly passing different paths to fictitious files. When a file is not found, the `FileInputStream` constructor throws a `FileNotFoundException`.

```
class ExceptionExample {
  public static void main(String[] args) throws FileNotFoundException {
    // Linux stores a user's home directory path in the environment variable
    // $HOME, Windows in %APPDATA%
    FileInputStream fis = new FileInputStream(System.getenv("APPDATA") +
args[0]);
  }
}
```

To overcome the problems, the exception must be caught while taking special care to sanitize the message before propagating it to the caller. In cases where the exception type itself can reveal too much information, consider throwing a different exception altogether (with a different message, or possibly a higher level exception, referred to as exception translation). The `MyExceptionReporter` class described in EXC01-J. Use a class dedicated to reporting exceptions is a good choice, as this compliant solution exemplifies.

Notice that `Throwable` is caught instead of catching specific exceptions. This is a departure from commonly suggested best practices, but is critical in cases where runtime exceptions or errors can reveal sensitive information. Moreover, this solution overcomes the issue of the brute force attack described earlier by accepting a denumerable set of file name choices with the help of a `switch-case` clause. Consequently, the actual file names and paths are hidden from the user of the application.

```
class ExceptionExample {
  public static void main(String[] args) {
    try {
      FileInputStream fis=null;
      switch(Integer.valueOf(args[0])) {
        case 1:
          fis = new FileInputStream("c:\\homepath\\file1");
```

```
          break;
        case 2:
          fis = new FileInputStream("c:\\homepath\\file2");
          break;
        //...
        default:
          System.out.println("Invalid option");
          break;
      }
    } catch(Throwable t) {
      MyExceptionReporter.report(t); // Sanitize
    }
  }
}
```

Code snippet 4:

This noncompliant code example checks for angle brackets in an attempt to validate user input to prevent script injection. A `URI` is constructed from the user input; this `URI` may consist of UTF-8 encoded character sequences. If the dangerous `%` characters comprising the encoding are not filtered out from the user input, the filter fails to achieve its purpose. For example, an attacker can bypass the filter by specifying the hexadecimal encoded form of the sequence `<script>` as `%3C%73%63%72%69%70%74%3E`.

```
Pattern pattern = Pattern.compile("[<>]");
String tainted = "%3C%73%63%72%69%70%74%3E"; // Hex encoded equivalent form of
<script>
if(pattern.matcher(tainted).find()) {
  throw new ValidationException("Invalid Input");
}
URI uri = new URI("http://vulnerable.com/" + tainted);
```

This compliant solution validates the input based on a white-list. The URL is allowed to contain only alphanumeric characters. The encoded forms of the space (" ") and period (".") characters are also allowed. All other characters are treated as invalid and rejected.

```
Pattern pattern = Pattern.compile("[^(a-z)^(A-Z)^(0-9)^(%20)^(%2E)]");
String tainted = "%20index1%2Ehtml"; // " index1.html"
if(pattern.matcher(tainted).find()) {
  throw new ValidationException( "Invalid Input");
}
URI uri = new URI("http://vulnerable.com/" + tainted);
```

Task 3 (20%) –Penetration testing/test plan

You are working at a software security company that specializes on performing penetration testing. You are preparing an offer on doing penetration testing for a potential new customer. This customer develops a system for online blogging. The blogging system allows users to create and customize the look and functionality (by selecting from available standard components or creating their own custom components) of their own blogs.

You are developing a high-level test plan that will be included in the offer. Explain and demonstrate how you would do this task. Make assumptions where necessary.

There are two expected approaches to answering this task:

- *The not-so-good: simply listing all the attacks in "How to break" as a test plan. Very generic. (But expected based on what we have seen in the exercises this fall and exams from previous years.)*
- *The good answer: creating a small set of requirements for the system based on the task text. Using this as a basis for threat-modeling and creating a threat/risk based test-plan that is specific to this task.*
    - *Note that risk analysis may be included but that would require some assumptions to be made. A risk analysis normally takes into account the business goals of the client which is not known from the text. More information than is given is required to rank risks.*

Task 4 – Integrating new and old technology (30%)

As many other banks, the B-Bank still rely on old core systems, typically written in Cobol and running on large mainframes, to perform financial transactions. Mainframes are fast and reliable, but are not intended to tackle the challenge of communicating with web-based customer tools. The B-Bank has therefore decided to start a large software development project, where the goal is, over time, to replace the old mainframe system with a new system based on standard components and web technology. The project is incremental and focuses on functionality. This means that selected parts of the desired functionality is included in each release. This means that for most of the project lifetime (5 years), new and old technology together make up the core systems. The project has as an important vision to create modern customer solutions enabling customers to be as self-serviced as possible.

You have been hired to be the security expert on the project. Your first task is to perform a security analysis of the project and identify the main security challenges. As always, the resources are limited so you are also expected to contribute information to help prioritize security measures. Explain and demonstrate how you would perform this task. Make assumptions where necessary.

The answer to this task should include:

- *Requirements -> security requirements*
- *Threat modeling – including identified countermeasures*
- *Risk analysis to prioritize threats and countermeasures*
- *The answers should not only explain **how** but also provide **examples***
- *Given the time-limits of the exams complete examples are not expected, but they should be sufficient to demonstrate knowledge both of methods and threats and also they should be specific to this task*
- *Note that this task accounts for 30% of the exam overall score, this should be reflected in the effort put into this task*