NTNU – Trondheim
Norwegian University of
Science and Technology

Department of Computer and Information Sciences

Examination paper for TDT 4237
Software Security

Academic contact during examination: Carl-Fredrik Sørensen

Phone: 951 19 690

Examination date: December 9th, 2015

Examination time: 09:00 AM to 1:00 PM

Permitted examination support material: Code D – No printed or hand-written materials allowed. Approved calculator allowed.

Other information: Exam developed by Carl-Fredrik Sørensen and checked by Per Håkon Meland

Language: English

Number of pages: 5

Number of pages enclosed: 0

Checked by:

_____

Date Signature

Introduction

In this exam, you can score a maximum of 70 points. The remaining 30 points for the semester comes from the compulsory exercises.

If you feel that any of the problems require information that you do not find in the text, then you should

- Document the necessary assumptions
- Explain why you need them

Your answers should be brief and to the point.

Problem 1 – (40 points)

a) (10%) It is hard to keep a private key to a system protected. Describe different methods an attacker may use to get access to such a key, enabling the attacker to get access to valuable digital assets within a company. Note: Do not dig into technical details; give a few examples of each identified method.

Main challenge with private keys tend to be related to key distribution, secure storage, access, renewal and revocation. The students should know the difference between passwords and private keys. A password is something that a human can remember, a key seldom is

This will depend on which type of keys that are used. This task can be modelled using attack trees, where different types of keys are exposed to different types of attacks.

- Typically, *social engineering* is a quite common way to get access to keys like passwords. Passwords are quite often easy to guess if no password policy exists. Insiders tends to be the weakest link to get hold on keys. Phishing, spoofing, pretexting are examples of techniques to try to get access to keys.
- Weakly engineered systems can contain passwords or tokens that can be obtained by information gathering about the system, checking exceptions, logs etc.
- Enable a Man-in-the-middle attack to get hold on private keys while a company/person tries to log into a system.
- Eavesdropping
- Brute-force
- Threats
- Get a job in the company

b) (5%) Explain the differences between multilevel and multilateral security policies. Give examples of established models of security policies.

Multilevel security policy – Is concerned with a separation of access between levels in an application, information hierarchy. Each user is assigned to a group. The group is authorised to a certain access level, which defines what information and services that are accessible

Figure 9.1: Multilevel security

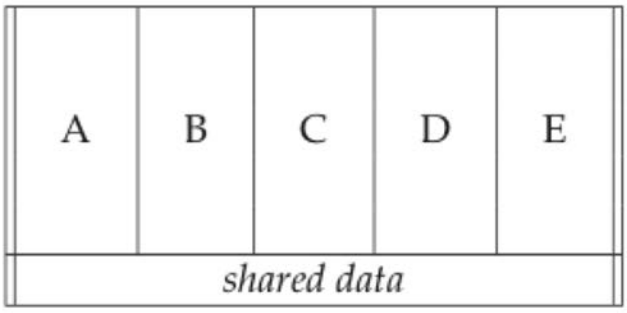


Figure 9.2: Multilateral security

Multilateral security policy – What is multilateral security?
In one sentence: controlling information flow across a database or shared data storage
Ideally, anyone will have access to exactly what needed, and nothing more. Assigning privileges to certain users or groups for accessing information or services, and blocking all others. To protect information from leaking between compartments on the same level.

The "ideal" of Multilateral Security can be described as follows:
1. Considering Conflicts: Different parties involved in a system may have different, perhaps conflicting interests and security goals.
2. Respecting Interests:
a. Parties can define their own interests.
b. Conflicts can be recognized and negotiated.
c. Negotiated results can be reliably enforced. Supporting Sovereignty: Each party is only minimally required to place trust in the honesty of others.
d. Each party is only minimally required to place trust in the technology of others.

Multilateral Security in general refers to all "classical" security goals, i.e. confidentiality, integrity, availability, or accountability can be in the interest of one party, but not necessarily in that of another.

Bell-LaPadula and Biba are examples of multilevel security policies, which focus respectively on confidentiality and integrity.
Compartmentation, the Chinese Wall and the Lattice Model are three different models to implement multilateral security policies.

It is not expected that the students know all the models for multilateral security policies.

c) (5%) When doing a risk assessment of a system, which methods or ways can be used to calculate/place a probability and impact/consequence to rank a risk? Give example(s) to illustrate such a method.

Probabilities are mostly based on the understanding of the likelihood that a risk can happen. Empirical knowledge will play an important role, both statistics gathered, similar attacks to other systems locally or in other businesses. E.g. how often has an attack happened? How easy is it to exploit a certain vulnerability? What can an attacker gain by doing an attack?

Impact are harder to estimate from a single person perspective, it is thus necessary to understand how the business is affected by the risk. It is here important that the different stakeholders in a company have an understanding of the business goals and assets, and role the system have to these assets (increase/protect/etc.). Impact can often be concerned with loss of business, loss of operation, loss of reputation, loss of customers, loss of monetary values, type of information and exposure of this information etc.

d) (10%) What is the main differences between BSIMM and OpenSAMM? How can a company enhance software security practices by using these frameworks?

The BSIMM is a descriptive model that can be used to measure any number of prescriptive SSDLs. Descriptive models describe what is actually happening.

Prescriptive models describe what you should do
• Software Assurance Forum for Excellence in Code (SAFECode)
• Software Assurance Maturity Model (SAMM)
• Microsoft SDL
• Cigital Touchpoints
• (OWASP CLASP)

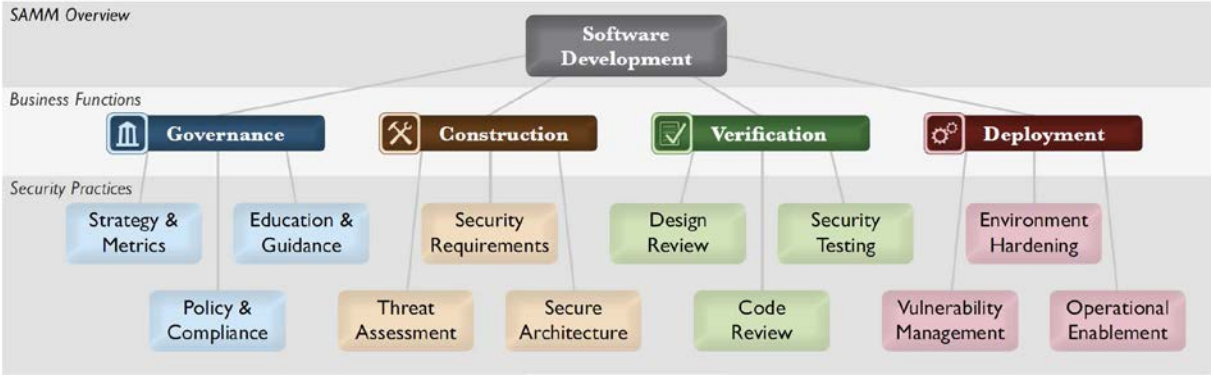
Big idea: Build a maturity model from actual data gathered from 9 well known large-scale software security initiatives

- Create a software security framework
- Interview nine firms in-person
- Discover 110 activities through observation
- Organize the activities in 3 levels
- Build scorecard

The model has been validated with data from 67 firms. Every firm has a methodology they follow (often a hybrid). BSIMM describes and measures multiple prescriptive approaches

The Software Security Framework (SSF)			
Governance	**Intelligence**	**SSDL Touchpoints**	**Deployment**
Strategy and Metrics	Attack Models	Architecture Analysis	Penetration Testing
Compliance and Policy	Security Features and Design	Code Review	Software Environment
Training	Standards and Requirements	Security Testing	Configuration Management and Vulnerability Management

OpenSAMM is a prescriptive method to get an understanding of how to become more mature in managing security issues.



0 - Implicit starting point representing the activities in the Practice being unfulfilled
1- Initial understanding and ad hoc provision of Security Practice
2 - Increase efficiency and/or effectiveness of the Security Practice
3 - Comprehensive mastery of the Security Practice at scale

The frameworks can thus both help to assess the current security level in different areas (BSIMM), and to provide best practices and guidelines for improving the security level in different areas (OpenSAMM).

e) (5%) Describe how a CSRF attack works and how to test for CSRF vulnerabilities. Explain by using an example.

Cross-Site Request Forgery (CSRF) is an attack that forces an end user to execute unwanted actions on a web application in which they're currently authenticated. CSRF attacks specifically target state-changing requests, not theft of data, since the attacker has no way to see the response to the forged request. With a little help of social engineering (such as sending a link via email or chat), an attacker may trick the users of a web application into executing actions of the attacker's choosing. If the victim is a normal user, a successful CSRF attack can force the user to perform state changing requests like transferring funds, changing their email

address, and so forth. If the victim is an administrative account, CSRF can compromise the entire web application.

CSRF is an attack that tricks the victim into submitting a malicious request. It inherits the identity and privileges of the victim to perform an undesired function on the victim's behalf. For most sites, browser requests automatically include any credentials associated with the site, such as the user's session cookie, IP address, Windows domain credentials, and so forth. Therefore, if the user is currently authenticated to the site, the site will have no way to distinguish between the forged request sent by the victim and a legitimate request sent by the victim.

CSRF attacks target functionality that causes a state change on the server, such as changing the victim's email address or password, or purchasing something. Forcing the victim to retrieve data doesn't benefit an attacker because the attacker doesn't receive the response, the victim does. As such, CSRF attacks target state-changing requests.

It's sometimes possible to store the CSRF attack on the vulnerable site itself. Such vulnerabilities are called "stored CSRF flaws". This can be accomplished by simply storing an IMG or IFRAME tag in a field that accepts HTML, or by a more complex cross-site scripting attack. If the attack can store a CSRF attack in the site, the severity of the attack is amplified. In particular, the likelihood is increased because the victim is more likely to view the page containing the attack than some random page on the Internet. The likelihood is also increased because the victim is sure to be authenticated to the site already.

For more information about CSRF, see https://www.owasp.org/index.php/Cross-Site_Request_Forgery_%28CSRF%29

How to review code for CSRF vulnerabilities.

How to test for CSRF, see https://www.owasp.org/index.php/Testing_for_CSRF_(OTG-SESS-005)

How to prevent CSRF, see https://www.owasp.org/index.php/CSRF_Prevention_Cheat_Sheet

f) (5%) What are software security touchpoints, and how do these influence the software development process?

Software security touchpoints are the sub-processes in a software development process, where security issues should be considered. See http://www.swsec.com/resources/touchpoints/ and **The Trustworthy Computing Security Development Lifecycle: https://msdn.microsoft.com/en-us/library/ms995349%28d=printer%29.aspx**

Here are the touchpoints, 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

Problem 2 – (40 points)

Case description:
Chronic obstructive pulmonary disease (COPD), also known as chronic obstructive lung disease (COLD), chronic obstructive airway disease (COAD), in Norwegian, Kronisk obstruktiv lungesykdom (KOLS), among others, is a type of obstructive lung disease characterized by chronically poor airflow. It typically worsens over time. The main symptoms include shortness of breath, cough, and sputum production.

Patients with KOLS should daily be able to register simple data about their health situation and condition to allow follow-up by the health care system. Examples of data could be the current airflow, spit colour, exercises, etc. The patient data use a mobile device like a smart phone or pad to register the data. The information is then transmitted through a cloud solution to the hospital. The cloud solution collects data from all patients. The data is then transmitted to the local hospital. The local hospital has a separate patient database/health journal system located on a separate health network.

a) (5%) Describe the attack surface of this solution.

In this task, it is expected that all main parts in the solution is described and presented as part of the attack surface. A dataflow diagram can be used to show the different parts of the solution, where all communication channels and main components are described. All parts can be attacked – either by external agents, or by internal agents.
Data Flow Diagram or similar would be a bonus. The main idea here is that the student should separate between components that have different risks.
Main components: Mobile device, cloud solution, local patient database/health journal system.
Main interfaces: Mobile device/app for input/response, communication of data to cloud solution, communication from the cloud solution to hospital system.

b) (5%) ***Identify and describe security threats (unwanted security incident) and possible threat agents/attackers to this solution.***

In this task, a good answer would use different threat modelling techniques to identify threat agents and attackers, and to identify security threats. The modelling techniques are misuse case diagrams, attack trees, and dataflow diagrams, with different properties. Misuse cases would be a good method here. Should be at _least_ 4-5 threats and a couple of attackers.

Some threats:

- Confidentiality: Sensitive data are exposed
- Availability: System is unusable/inaccessible
- Integrity: Information is altered
- User/device information is exposed
- Sessions are hijacked or CSRF
- Systems are taken over
- Mobile device is lost or stolen

c) (10%) Make a risk assessment of the solution using RMF

In this task, it is expected that the RMF is followed:

1. Understand business context: Identify business goals and assets
2. Identify and link business and technical risks. The technical risks should be related to a business risk. The different artifacts should be analyzed.
3. Synthesize & rank the risks

Examples of BR and TR:
BR1: Leak sensitive information about patients
- TR1-1: Vulnerable communication channel in transfer of data
- TR1-2: Information sent or stored without any encryption
- TR1-3: No access control
- TR1-4: Mobile device is lost or stolen, giving access to thief

BR2: System is unavailable which may harm patients because of loss of opportunity to follow-up the patient.
- TR2-1: System is not scaled for the current number of users
- TR2-2: System crash and cannot be recovered

BR3: Information provided is tampered or false, and the integrity of the system is not to be trusted
- TR3-1: Information is changed by a third-party during communication or storage
- TR3-2: An attacker steals user session and act on behalf of user
- TR3-3: Mobile device is lost or stolen, giving access to untrusted parties

BR4: Bad usability leading to patients not using the system or entering wrong information
BR5: System is not trusted which may harm reputation of solution provider, health institution.
BR6: Health workers do not get necessary information to treat patients correctly or at all.
BR7: The solution is not compliant to the national laws and regulations.

d) (5%) How would you mitigate the threats and risks?

This task is a continuation of the previous task (2c), where a mitigation strategy is defined to manage the identified risks. This is basically step 4 in RMF.

Possible mitigations:
- Encrypted communication
- Encrypted storage
- User data is not stored at the cloud database
- A mobile device ID (e.g. MAC-address or similar) is used as authentication instead of user credentials (would require a first-time authentication of device along with user credentials, which can be done outside the app by using e.g. a federated ID (MinID, BankId, etc.).
- Test and sanitize input
- Secure sessions against CSRF
- Logging of incidents, location, IPs, MACs etc to discover forgery, unexpected locations, etc.

e) (10%) State which security requirements and other security issues you believe should be addressed in the implementation of the solution.

The requirements stated should be concise, precise, and testable. The requirements should be based on the analysis done in 2a-2d, and linked to the identified risks.

f) (5%) Discuss any ethical concerns that should be taken into account in this solution?

The important thing here is that the student shows an ability to reason about pros and cons about security and human concerns.
Different issues may be discussed in this task. Important considerations should be about who are users, and what is collected of data, who have access to these data. The presence of an health related app on a device could be problematic since the app normally needs to be downloaded from an App storage. This may harm confidentiality of the client who thus can be identified by the actors presenting app stores. The information collected can possible harm the patient if this information is available for other actors than the treating personnel. The collection of data in a cloud storage from many patients could potentially be a problem, especially if you are able to use "big data" techniques to trace information back to specific persons. Privacy and management of privacy is a concern with ethical considerations. On the other side, so can it be discussed whether privacy is of greater importance than the ability to help patients fast. Day-to-day monitoring of patients in their homes are cheaper than other options, spending less money on personal meetings and lab hours at physicians and hospitals. Using specific algorithms to analyze the gathered information will in addition reduce costs and possible patient risk if left unattended.

Problem 3 – (20 points)

For each of the code snippets listen below, your task is to:

- Identify the main security vulnerabilities/issues.
- Explain why this is a security issue.
- Fix the code. You may use pseudo-code for this. Remember to explain your solution.

Code snippet 1
The following code uses a statistical PRNG to create a URL for a receipt that remains active for some period of time after a purchase (DO NOT DO THIS).

```
String GenerateReceiptURL(String baseUrl) {
        Random ranGen = new Random();
        ranGen.setSeed((new Date()).getTime());
        return(baseUrl + ranGen.nextInt(400000000) + ".html");
}
```

Issue: Insecure randomness. Insecure randomness errors occur when a function that can produce predictable values is used as a source of randomness in security-sensitive context.

There are two types of PRNGs: statistical and cryptographic. Statistical PRNGs provide useful statistical properties, but their output is highly predictable and forms an easy to reproduce numeric stream that is unsuitable for use in cases where security depends on generated values being unpredictable. Cryptographic PRNGs address this problem by generating output that is more difficult to predict. For a value to be cryptographically secure,

it must be impossible or highly improbable for an attacker to distinguish between it and a truly random value. In general, if a PRNG algorithm is not advertised as being cryptographically secure, then it is probably a statistical PRNG and should not be used in security-sensitive contexts.

source: https://www.owasp.org/index.php/Insecure_Randomness

The following code uses Java's SecureRandom class to generate a cryptographically strong pseudo-random number (DO THIS):

```
public static int generateRandom(int maximumValue) {
        SecureRandom ranGen = new SecureRandom();
        return ranGen.nextInt(maximumValue);
}
```

Code snippet 2
In the following method, a DNS lookup failure will cause the Servlet to throw an exception.

```
protected void doPost (HttpServletRequest req,
                                        HttpServletResponse res)
                        throws IOException {
        String ip = req.getRemoteAddr();
        InetAddress addr = InetAddress.getByName(ip);
        ...
        out.println("hello " + addr.getHostName());
}
```

Issue: Information leakage. Revealing system data or debugging information helps an adversary learn about the system and form a plan of attack. An information leak occurs when system data or debugging information leaves the program through an output stream or logging function.

Missing Catch Block

If a Servlet fails to catch all exceptions, it may reveal debugging information that will help an adversary form a plan of attack.

When a Servlet throws an exception, the default error response the Servlet container sends back to the user typically includes debugging information. This information is of great value to an attacker. For example, a stack trace might show the attacker a malformed SQL query string, the type of database being used, and the version of the application container. This information enables the attacker to target known vulnerabilities in these components.

Source: https://www.owasp.org/index.php/Information_Leakage

A good error handling mechanism always tries to capture all exceptions and returns a generic error message that does not reveal any details about the error and the application. Depending on the platform and container the application is running on, there can be different options.

- Set a generic custom error page for all unhandled exceptions at the container level. (Normally, this is set in the configuration file.) The generic custom error page should have a simple error message that does not reveal any details about the exception happened.

Code snippet 3
***login.php*:**

```
<?php
session_start();
?>
<html>
 <body>
<?php
if (isset($_SESSION[“user”])) {
    echo “<p>Welcome back, “ . $_SESSION[“user”] . “!<br>”;
    echo ‘<a href=”process.php?action=logout”>Logout</a></p>’;
}
else {
?>
  <form action=”process.php?action=login” method=”post”>
   <p>The username is: admin</p>
   <input type=”text” name=”user” size=”20”>
   <p>The password is: test</p>
   <input type=”password” name=”pass” size=”20”>
   <input type=”submit” value=”Login”>
  </form>
<?php
}
?>
 </body>
</html>
```

process.php:

```
<?php
session_start();

switch($_GET["action"]) {
    case "login":
        if ($_SERVER["REQUEST_METHOD"] == "POST") {
            $user = (isset($_POST["user"]) &&
                ctype_alnum($_POST["user"]) ? $_POST["user"] : null;
            $pass = (isset($_POST["pass"])) ? $_POST["pass"] : null;
            $salt = '$2a$07$my.s3cr3t.SalTY.str1nG;

            if (isset($user, $pass) && (crypt($user . $pass, $salt) ==
                crypt("admintest", $salt))) {
                $_SESSION["user"] = $_POST["user"];
            }
        }
        break;

    case "logout":
        $_SESSION = array();
        session_destroy();
        break;
}
```

```
header("Location: login.php");
?>
```

Issue : CSRF
The `login.php` script begins by initializing the session data. It then checks to see if `$_SESSION["user"]` has been set, and if so displays a welcome message along with a link to logout. Otherwise it displays the login form.

The `process.php` script also begins by initializing the session data, and then checks to see if there is an action to work with. We perform some basic input validation using PHP's ternary operator along with the `ctype_alnum()` and `crypt()` functions, and then set or destroy the session variable accordingly. The user is redirected back to `login.php` at the end of the script.

Now let's focus on the file an attacker might create to exploit the code in our previous examples. This is the exploit code, `harmless.html`:

```
<html>
 <body>
  <p>This page is harmless... Or is it?</p>
  <!-- Address to target website -->
  <img src="process.php?action=logout" style="display: none;">
 </body>
</html>
```

If you visit `login.php` and log in to your account, and then while logged in you proceed to visit the attacker's page, you will be automatically logged out even though you didn't click the logout link. The browser sends a request to the server to access the `process.php` script, expecting it to be an image file. The processing script has no way of differentiating between a valid request initiated by a user clicking on the logout link and a cleverly-crafted request the browser was tricked into sending.

The `harmless.html` file could be hosted on an entirely different server than the one you're logged into, and it would still work because the attacker's page is making a request on your behalf using the session you have open in the background. It doesn't even matter if the website you're logged into is on a private network, the request will be submitted from your IP address as if you made the request yourself, making a trace to the source of the attack nearly impossible.

Additionally, if you allow your users to link to images as a profile avatar or the like, without proper escaping and sanitizing of the user supplied data the attack may even be possible within your own web domain.

While logging someone out of a website isn't that impressive, `harmless.html` could have just as easily contained a hidden inline frame (as opposed to an image tag) with a form that automatically submits when the page is loaded, which would make any of the attacks mentioned at the beginning of this guide fair game.

Source : http://www.sitepoint.com/preventing-cross-site-request-forgeries/

In order to ensure that an action is actually being performed by the user rather than a third party, you need to associate it with some sort of unique identifier which can then be verified. To prevent the attack, we can modify `login.php` as follows:

```
<?php
// make a random id
$_SESSION["token"] = md5(uniqid(mt_rand(), true));
echo '<a href="process.php?action=logout&csrf=' . $_SESSION["token"] .
'">Logout</a></p>';
```

Then to verify the identifier, we can modify `process.php` as follows:

```
case "logout":
    if (isset($_GET["csrf"]) && $_GET["csrf"] == $_SESSION["token"]) {
        $_SESSION = array();
        session_destroy();
    }
    break;
```

With these simple modifications, `harmless.html` will no longer work because the attacker has been given the additional task of having to guess an additional random token value. To protect forms, you can also include the identifier inside of a hidden field as follows so it is submitted along with the rest of the form data.

```
<input type="hidden" name="csrf" value="<?php echo $_SESSION["token"]; ?>">
```

Code snippet 4

```
class Example
{
   private $hook;

   function __construct()
   {
      // some PHP code...
   }

   function __wakeup()
   {
      if (isset($this->hook)) eval($this->hook);
   }
}

// some PHP code...

$user_data = unserialize($_COOKIE['data']);

// some PHP code...
```

Issue: Object injection.

PHP Object Injection is an application level vulnerability that could allow an attacker to perform different kinds of malicious attacks, such as Code Injection, SQL Injection, Path Traversal and Application Denial of Service, depending on the context. The vulnerability occurs when user-supplied input is not properly sanitized before being passed to the unserialize() PHP function. Since PHP allows object serialization, attackers could pass ad-hoc

serialized strings to a vulnerable unserialize() call, resulting in an arbitrary PHP object(s) injection into the application scope.

In order to successfully exploit a PHP Object Injection vulnerability two conditions must be met:

- The application must have a class which implements a PHP magic method (such as __wakeup or __destruct) that can be used to carry out malicious attacks, or to start a "POP chain".
- All of the classes used during the attack must be declared when the vulnerable unserialize() is being called, otherwise object autoloading must be supported for such classes.

In this example an attacker might be able to perform a Code Injection attack by sending an HTTP request like this:

```
GET /vuln.php HTTP/1.0
Host: testsite.com
Cookie:
data=O%3A8%3A%22Example2%22%3A1%3A%7Bs%3A14%3A%22%00Example2%00hook%22%3Bs%
3A10%3A%22phpinfo%28%29%3B%22%3B%7D
Connection: close
```

Where the cookie parameter "data" has been generated by the following script:

```
class Example2
{
   private $hook = "phpinfo();";
}

print urlencode(serialize(new Example2));
```

Source: https://www.owasp.org/index.php/PHP_Object_Injection

Prevention

Do not use unserialize() function with user-supplied input, use JSON functions instead.