UNIVERSITY OF WATERLOO
Cheriton School of Computer Science

**CS 458/658** **Computer Security and Privacy** **Spring 2017**

**Stefanie Roos, Ian McKillop**

# ASSIGNMENT 2

Assignment due date: **June 30th, 3:00 pm**

**Total marks:** 60
**Written Response Questions TA:** Bailey Kacsmar (Office hours: Tuesdays at 1:30pm in DC3332), bkacsmar@uwaterloo.ca
**Programming Question TA:** Miti Mazmudar (Office hours: Thursdays at 1:30pm in DC3332), miti.mazmudar@uwaterloo.ca

Please be sure to use complete, grammatically-correct sentences. You will be marked on the presentation and clarity of your answers as well as the content.

## Written Response Questions [30 marks]

1. [Total: 15 marks] **Security Policies - BIBA Integrity Model/Bell-La Padula Confidentiality Model**
   Consider the following hierarchy for course material access:
   **Student** $\leq$ **TA** $\leq$ **Professor**
   Let Alice be a TA with the following clearance: (TA, {assignments, exams})
   Let Bob have the following clearance (Professor, {assignments, exams, marks})
   Let Carol have the following clearance (Student, {assignments, marks})
   Document types consist of: assignments, marks, slides, exams, evaluations

   (a) [5 marks] Using the Bell-La Padula Confidentiality Model, state whether Alice has read access, write access, both, or neither, for each of the following files: [5 marks]
      i. file1: (Student, {assignments})
      ii. file2: (TA, {slides, assignments})
      iii. file3: (TA, {exams, evaluations})
      iv. file4: (TA, {assignments, exams})
      v. file5: (Professor, {assignments, exams, marks})

   (b) [10 marks]For the following sequence of accesses, indicate the change in the integrity levels of the subject and the object under a Dynamic BIBA model, using the low watermark property (for both subject and object).

i. Alice writes to a file A with integrity: (Professor, {assignments, marks})

ii. Carol reads from a file B with integrity: (TA, {assignments, marks})

iii. Bob writes to a file C with integrity: (Professor, {assignments, marks, slides})

iv. Alice writes to a file D with integrity: (Professor, {assignments}). Bob reads from file D and writes to file E with integrity: (Professor, {marks, assignments})

v. Carol writes to a file F with integrity: (TA, {marks}). Alice reads from file D.

2. [Total: 9 marks] **Password Cycling and Similarities**
Hint: You might wish to consider the optional readings on the course webpage for this question. Alice has an account at website CoolSite. Alice used her favorite password creation method and generates the password *kittens*.

i. CoolSite does not store their passwords securely and Alice's password is leaked.

ii. Alice has another account on NewAwesomeSite which has a password policy. NewAwesomeSite's policy is that all passwords must now contain at least 1 capital letter, at least 1 lower case letter, a numeric value and be at least 8 characters long. Alice adapts her password *kittens* by capitalizing the first letter and adding a random digit.

iii. NewAwesomeSite revisits their password policy, and decides that their latest policy requires that all passwords contain at least 1 capital letter, at least 1 lower case letter, on digit, at least 1 symbol from the set (!, #, $, %, &, *) and be at least 12 characters long. Alice changes her password from ii. by adding a random symbol and adding three random digits.

(a) What are two problems with Alice's password choice in 'i'? Explain.

(b) Assuming that Eve knows Alice password for CoolSite and also knows her strategy for creating a new password, how many possibilities does Eve have to test to figure out the password in ii.(worst-case)? Explain.

(c) Name one possible password Alice may choose to satisfy NewAwesomeSite's latest policy (as stated in (iii)).

(d) Given that Eve knows Alice's initial password, Alice strategy for extending the password, and the latest password policy, how many possible passwords would have to be tested to find Alice's password? Consider both the case that Eve knows the password after ii. and the case that she does not.

3. [Total: 6 marks] **Password Hashing**
Alice needs to design a password scheme for a secure user account database. She considers to hash all passwords with SHA-1 and a user-specific 18-bit salt. Find at least three problems with the design, explain them and propose alternatives that fix the problems.

## Programming Question [30 marks]

### Background

You are tasked with performing a security audit of a custom-developed *web application* for your organization. The web application is a content sharing portal, where any user can view content, which includes articles, links, images, and comments. Registered users can log in and then post content. You are provided black-box access to this application, that is, you can access the website in the same manner as a user.

The website uses PHP to generate HTML content dynamically on the server side. The server stores the application data, which includes usernames, passwords, comments, articles, links and images, on a SQLite3 database at the server. In this manner, we consider two systems, a relational database and HTML forms, that by default do not validate user inputs.

The assignment consists of three questions, each considering one distinct website vulnerability. Your tasks can be one or more of the following: i) gather information about attack and defence strategies, ii) implement attack strategies, or iii) design defence strategies (theoretically, as implementation would in-depth knowledge of PHP). You might wish to start with reading the material provided in Section 'Attacks, defenses and protocols' at the end of the assignment before tackling the questions.

1. [Total: 5 marks] **SQL Injection:**
   User-generated data is directly passed into database queries in several contexts, for example, to insert or modify user-generated content in a database, or to query a database that contains user credentials for authentication. A user can craft their input to include malicious database queries, such as to delete records or tables.

   (a) Use a SQL injection attack on the log-in form to obtain access as another user. [4 marks]

   (b) Input sanitization is a common but insufficient method to prevent SQL injection attacks, as it is hard to eliminate all corner-cases for successful injections. After gathering information on this issue (e.g.; on the Internet), briefly describe a method to prevent SQL injection attacks, which separates the query data from the query syntax. [1 mark]

   For the following two sections, use the following credentials to log-in to the system:
   username "alice"
   password "passw0rd".
   If required for testing your exploit scripts, you can insert upto 10000 articles, links, and images.

2. [Total: 9 marks] **Cross-site scripting:**
Cross-site scripting attacks involve injecting malicious web script code (including HTML, Javascript) into a webpage form without the input being validated. As a result, the document object model (DOM) of the HTML webpage changes whenever the code is executed. Depending on whether the change persists across reloads of the webpage, XSS attacks are classified as stored (persistent) or non-persistent attacks.

   As a tester, to observe changes in the DOM of the HTML webpage, you may need to observe the source code of the page and monitor the HTTP requests and responses. This can be done using standard 'Developer tools' menus in browsers and/or using adequate flags for command-line utilities like wget.

   (a) Identify *all* fields in forms on this website that do not prohibit invoking the Javascript environment through the (<script>) tags. Conduct stored XSS attacks through these vulnerable fields as follows: For each of these fields, pass simple Javascript code as input so that a distinctive pop-up appears on your user Alice's homepage whenever she loads this site. Your script should result in XSS attacks against all vulnerable fields. [7 marks]

   (b) Does the same-origin policy prevent a XSS attack? Briefly describe why or why not. [1 mark]

   (c) As a maintainer of this website, you need to *mitigate* this specific XSS attack. That is, you need to stop some malicious code, which has been inserted into the database, from being executed when delivered to a client, without modifying the database content or database handler functions. Recommend a strategy to achieve this (you may research online) and briefly describe how it works. [1 mark]

3. [Total: 16 marks] **Cross-site request forgeries:**
Cross-site request forgeries work on the premise that any identifying state information maintained on the client is sent by the browser with any subsequent HTTP requests to the website. That is, there is no way to identify whether an HTTP request is 'genuinely' sent by the user or if the browser has been 'duped' into sending it. Cross-site request forgeries exploit this fact: HTTP requests are generated by the user's browser, which result in significant transactions, without their knowledge.

   (a) Test this website for CSRF vulnerabilties. Assume that a user Alice has logged in to the site and can be socially engineered to click on a URL with the same domain as this site, from a phishing email while she is logged in. Your script should carry out at least one action through CSRF-vulnerable forms and any parameters, values should not be in a textual format (i.e. "username=alice"). [6 marks]

   (b) Observe that the response to the forged HTTP request is delivered to Alice's machine and not the attacker's. Suppose that the response contains an update on sensitive information. A stored XSS attack can be used to send this information back to the attacker. Write a script to conduct a stored XSS attack on one of the vulnerable fields found

in Question 2a, such that the cookie value is sent to cookie_stealer.php or shown as a pop-up. [6 marks]

(c) Read through the links on defences against CSRF attacks, including CORS and the same-origin policy in the "Attacks, defense and protocols" section before you proceed with the following questions. [1 mark each]

   i. Briefly describe a defense against this CSRF attack that involves no modifications to the HTTP headers sent by the website.

   ii. Briefly describe a defense that does not involve the server storing more information.

(d) Would configuring the server to use HTTPS instead of HTTP eliminate CSRF attacks? Briefly describe your answer. [1 mark]

(e) Assume that Alice is likely to visit Eve's website dancingpigs.com while she is logged in to our website. Now, Eve sets up a URL link on her site dancingpigs.com to conduct the forged actions (found in part 1) from Alice's account on our website. Under the current configuration of the server, would the HTTP request be accepted by the server? Give an appropriate reason for your answer. (Hint: Consider CORS headers.) [1 mark]

**Rules for exploit script execution** You need to submit exploit scripts for questions **1a, 2a, 3a, 3b**. For each of your exploits, you should submit a tarball (sploit[1-4].tar) each containing the following:

- **sploit.sh** - shell script to run your exploit. The scripts should *not* be within a subdirectory in the tar file.
- any other files needed to run your exploits

Each script must accept **exactly two command-line arguments**, which will be the **IP address** and **destination port** of the web server, as follows. **The script should output a message describing what it is doing.** `./sploit.sh 10.0.3.135 80`

**If we can't figure out how your exploit works, you will not earn any marks for it.** If in doubt about how to present an exploit (or whether an exploit is acceptable), please send an email to the programming TA or check during office hours.

## What to hand in

Using the "submit" facility on the student.cs machines, hand in the following files:

**a2.pdf** A PDF file that contains your answers to all written response questions, plus answers to the following questions in the programming part: 1b, 2b, 2c, 3c, 3d, 3e.

**sploit[1-4].tar** (uncompressed) tar files containing your completed exploits for the programming question.

**Note:** You must include your name, your uWaterloo userid, and your student number at the top of the first page of a2.pdf. Failure to do so will result in a deduction of 5 marks from your final score on this assignment! Also, be sure to "embed all fonts" into your PDF files. Some students' files were unreadable in the past; if we can't read it, we can't mark it.

## Useful Information For Programming Sploits

A web server is running on each of the ugster machines, and a directory has been created using your ugster userid. Your copy of the web application can be found at:

`http://ugsterXX.student.cs.uwaterloo.ca/userid`

...where `ugsterXX` and `userid` are the machine and credential assigned to you previously (see Piazza).

If you need to reset the webserver, simply access the following URL:

`http://ugsterXX.student.cs.uwaterloo.ca/reset/userid`

This will delete all changes made to your copy of the web application, and restore it to its original state.

As with the previous assignment, the ugster machines are only accessible to other machines on campus. If you are working from home, you will need to first connect through another machine (such as `linux.student` or the campus VPN).

**Note 1:** If one of your exploits forces the application into an unusable state, you can restore the default state by accessing the URL above. If the web server itself becomes unusable, please report this on Piazza, along with a description of what you were doing when the problem occurred. *Do not perform denial of service attacks* on either the ugster machine or the web server.

**Note 2:** **Do not** connect to the web server with a userid different from the one that is assigned to you. Any student that is caught messing around with another student's web application by connecting to the wrong URL will receive an **automatic zero** on the assignment, and further penalties may be imposed.

### Attacks, defenses, and protocols

The following links may be helpful in designing exploit scripts and defenses.

- Cross-site scripting attacks and defenses: `https://www.owasp.org/index.php/Cross-site_Scripting_%28XSS%29`
- Cross-site request forgery and defenses: `https://www.owasp.org/index.php/Cross-Site_Request_Forgery_%28CSRF%29`
- SQL Injection attacks and defenses: `https://www.owasp.org/index.php/SQL_Injection`
- HTTP: `http://en.wikipedia.org/wiki/Hypertext_Transfer_Protocol` (if you are new to how HTTP/web works)
- JS: `https://www.w3schools.com/js/`
- Document Object Model: `https://developer.mozilla.org/en-US/docs/Web/API/Document_Object_Model/Introduction` (API for accessing parts/objects of an HTML page)
- Same-origin policy: `https://www.w3.org/Security/wiki/Same_Origin_Policy`
- Cross-origin resource sharing: `https://developer.mozilla.org/en-US/docs/Web/HTTP/Access_control_CORS`

### Writing your exploits

You may use any language of your choosing (within reason) to exploit the server; the only restriction is that your exploits run correctly from the ugster machines. You may use external tools to aid you if they are not directly required for exploit execution (i.e. they help you gather information, etc.).

A basic understanding of HTML forms will also be helpful for completing this assignment; a good starting point for learning about forms might be:

`http://www.cs.tut.fi/~jkorpela/HTML3.2/5.25.html`

Here are some other links that you may find useful:

- cURL: `http://curl.haxx.se/docs/manpage.html` (information about headers, GET, POST)

- cURL Scripting: `http://curl.haxx.se/docs/httpscripting.html` (if you don't know how HTML forms work)

- Shebang: `http://en.wikipedia.org/wiki/Shebang_(Unix)` (if you are new to scripting)

- Web sessions: `http://en.wikipedia.org/wiki/Session_(computer_science)` (go to the section on Web Server Session Management)

- Netcat: `http://www.stearns.org/doc/nc-intro.v0.9.html` - Making network connections from a command line