UNIVERSITY OF WATERLOO
Cheriton School of Computer Science

CS 458/658 **Computer Security and Privacy** **Fall 2018**
Ian Goldberg
Florian Kerschbaum

ASSIGNMENT 2

Assignment due date: **Friday, November 2nd, 2018 4:00 pm**

Total Marks: 153
Written Response Questions TA: Stan Gurtler
Programming Questions TA: Justin Tracey

Please use Piazza for all communication. Ask a private question if necessary. The TAs' office hours are posted to Piazza. You are expected to follow the expected Academic Integrity requirements for Assignments; you can find them here: `https://uwaterloo.ca/library/get-assignment-and-research-help/academic-integrity/academic-integrity-tutorial`. Strict penalties will be enforced on students for any Academic Integrity violations.

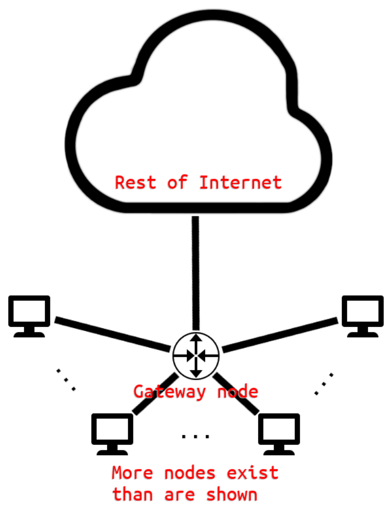
Written Response Questions (60 marks)

Note: Please ensure that written questions are answered using complete and grammatically correct sentences. You will be marked on the presentation and clarity of your answers as well as on the correctness of your answers.

By day, you, ORDINARY INDIVIDUAL, are an employee of a security consultancy firm which has just won a contract to test the security practices of a reasonably sized benevolent organization (that is, you're working as the organization's red team). By night, you are still that, but due to an insistence by the head of your firm to adopt codenames and metaphor to "keep up morale", you also have an alternate identity and role. You, PIQUANT MONGOOSE, are a high-ranking member of a secretive organization known only to outsiders as "UNSOCIETY" (actually, your firm, or even more specifically the members of your firm on the red team). Recently, you have established contact with an operative, DIFFIDENT MOLE, inside VILE CORP, a large company with dubious ethics that you want to expose and work to take down (in truth, the organization asking you to audit them). DIFFIDENT MOLE is a longtime employee who has had a change of heart (really, just their CISO). They want to walk away from the company and its actions and do something to impede it, but don't want to draw attention — legal or otherwise — to themselves. Despite the eyeroll-inducing nature of this imagining, you and your fellow employees humour the boss. It's a good gig, after all.

1. First, cast out the beam from your own eye (30 marks)

Before you can make any moves against VILE CORP, you must confirm your own resources will be secured against any unwelcome intrusion. After all, what would be more embarrassing than to be burned in the middle of your own operation? You collaborate with UNSOCIETY's head of network security (the firm's IT manager), ROBUST SPRINGBOK, to ensure your internal network is protected.



(14 marks) a. You and ROBUST SPRINGBOK decide you need to establish a firewall to protect your network from intrusion by external parties. Step-by-step, you will add rules below to meet the connectivity needs of UNSOCIETY and the least possibility connectivity beyond that. (**HINT:** CIDR Notation may be helpful for this portion of the assignment). The needs and reality of UNSOCIETY's network may be described as follows:

- UNSOCIETY controls 64 IP addresses with the prefix `72.36.115.128/26`.
- UNSOCIETY employs a star topology — all machines connect to each other, and the full internet, through one gateway node, located at `72.36.115.129`. (cf. diagram above) (This is where you will put the firewall).
- UNSOCIETY runs a server (located at `72.36.115.150`) that hosts its webpage (on the usual HTTPS port, `443`), which UNSOCIETY makes available to the full internet to proselytize its message (to advertise your firm, that is).
- ROBUST SPRINGBOK has noticed unusual activity coming out of St. Pierre & Miquelon (IP range `70.36.0.0/20`). While neither you nor ROBUST SPRINGBOK is sure, you think they may be trying to gain illicit access to the web server to shut it down.
- Secretly, UNSOCIETY receives funding by running a particularly lucrative game server for COMPETITIVE ONLINE GAME at `72.36.115.175`. For purposes of speed, COMPETITIVE ONLINE GAME uses UDP packets to communicate between servers and game clients; packets arrive and depart from port `4761` on the server, but clients may transmit from any port between `4700` and `4799`, inclusive.
- This same server secretly also hosts an IRC channel (on port `6667`). This channel is only for use by two parties — internal machines and DIFFIDENT MOLE from inside VILE CORP.
- IRC clients may use any port. IRC uses TCP connections. By default, IRC does not use TLS or other forms of encryption for its connections.
- DIFFIDENT MOLE's machine uses the IP address `56.172.1.164`.

Below, you will add rules to accomplish UNSOCIETY's networking goals. Note that you may need more than one rule to accomplish the goal in question. Please specify rules to include the following information: DROP or ALLOW, source IP address, destination IP address, source port, destination port, TCP or UDP or BOTH. An example rule to allow members of UNSOCIETY to receive the result of DNS lookups from `1.1.1.1` might look like this (think carefully: would this rule be sufficient to make a DNS lookup to `1.1.1.1`?):

ALLOW `1.1.1.1` ==> `72.36.115.128/26` FROM PORT 53 TO `all` BY `UDP`

HINT: Ports may be specified as a range, or as sets, or as a singular value, or as '`all`'. IPs should be specified in CIDR notation for multiple, or as a singular value for one. You may assume that this firewall is stateful; it makes decisions about (TCP) connections, but once (TCP) connections are established, it does not interfere.

1. Agents trying to browse the web from the internal network may do so.
2. The full internet may access the UNSOCIETY webpage.

3. The full internet may access the COMPETITIVE ONLINE GAME server.
4. Agents trying to connect to the IRC channel may do so.
5. DIFFIDENT MOLE may connect to the IRC channel.
6. ROBUST SPRINGBOK would like to set up a honeypot on the web server to gain more information about the potential St. Pierre & Miquelon spies. After backing up the server onto another machine, another operative (OBSEQUIOUS AARDWOLF) has designed a program that looks like SSH, and which has designed vulnerabilities that potential spies could interact with and believe they have infiltrated the server. Only these potential spies should be able to see it, however. (The server otherwise has no SSH capabilities).
7. Handle traffic not specifically allowed by other rules.

(4 marks) b. You have reason to suspect one of your newer agents, GROUCHY PUMA (IP address `72.36.115.191`), may be compromised. You are concerned they may attempt to exfiltrate data on UNSOCIETY's operations; with the firewall configuration you created above, would it be possible for them to? Why or why not? If it is possible for them to exfiltrate data, design a rule to prevent them from doing so, and state where in the order of the rules above it should go.

(4 marks) c. Is a firewall sufficient to keep the IRC communication secret? Why or why not? If so, what, if any, additional configuration would be required to keep that communication secret? If not, what additional measures could you take?

(2 marks) d. You think back to your time in CS 458/658, and remember black hole attacks can be a threat to availability of web services. You want your website (and especially, your game server) to be available no matter what — does your firewall protect you from a black hole attack? Why or why not?

(2 marks) e. While black hole attacks are self-evidently a problem when a malicious entity drops packets, is there any advantage gained by an adversary who uses the same technique to redirect packets through its network, but still otherwise routes them properly? Explain.

(4 marks) f. Name one advantage of the star topology network that UNSOCIETY uses. Name one disadvantage/potential improvement.

With your firewall in place, and a backchannel secured to communicate regularly with DIFFIDENT MOLE, you begin to reconnoitre VILE CORP.

2. She will win who, prepared herself, waits to take the enemy unprepared. (20 marks)

DIFFIDENT MOLE reports that VILE CORP uses the Bell-LaPadula Confidentiality Model in order to secure its documents. Their intel indicates that VILE CORP uses the following sensitivity/clearance levels:

$$\text{Executive} >_c \text{Management} >_c \text{Developer} >_c \text{Customer Support} >_c \text{Public}$$

Additionally, VILE CORP uses compartments for access control; compartments are characterized with Greek characters. For this part of the audit, DIFFIDENT MOLE instructs a lower-ranking member of the company (OBEISANT QUOKKA, your boss is quick to name them) to cooperate fully with your instructions, to test the resiliancy of their system against an insider threat. OBEISANT QUOKKA reveals that they hold (Management, $\{\phi, \upsilon, \rho, \eta\}$) clearance.

(8 marks) a. For each of the following documents, report whether OBEISANT QUOKKA has read access, write access, both, or neither for that document in the Bell-LaPadula Confidentiality Model.

(i) D926: (Developer, $\{\phi, \upsilon, \rho, \eta\}$)

(ii) D269: (Management, $\{\upsilon, \rho\}$)

(iii) D621: (Executive, $\{\phi, \eta\}$)

(iv) D513: (Management, $\{\phi, \upsilon, \rho, \eta\}$)

(v) D200: (Public, $\{\varepsilon\}$)

(vi) D634: (Executive, $\{\phi, \lambda, \upsilon, \rho, \eta, \psi\}$)

(vii) D364: (Customer Support, $\{\phi, \rho\}$)

(viii) D818: (Developer, $\{\phi, \eta, \chi\}$)

(12 marks b.) DIFFIDENT MOLE knows of a file that contains information they believe will be very valuable to UNSOCIETY's goals, D413 (Edited 2018-10-08) (~~Director~~ Executive, $\{\phi, \upsilon, \rho, \chi\}$). DIFFIDENT MOLE reports that the file is protected under a dynamic Biba model that makes use of the low watermark property for both subject and object.

You want D413. You are going to try and reduce D413's integrity level so that OBEISANT QUOKKA can give you the file without triggering alerts from a firewall or an intrusion detection system. There could be less contrived ways of getting the information in the file, but, admittedly, OBEISANT QUOKKA is not a technical-minded person — they're just an HR manager — and you want the exact file itself. Before starting, DIFFIDENT MOLE warns you of the following:

(i) The intrusion detection system will trigger an alert if a document loses more than one clearance level per action (for example, an action that drops a file from Management level to Unclassified level will trigger an alert).

(ii) Both objects and subjects with Management and with Customer Support level integrity must have an even number of compartments.

In a sequence of steps, direct OBEISANT QUOKKA to read or write D413, some of the eight documents listed earlier, or some of the below documents to step by step bring down D413's integrity level to (Public, $\emptyset$). Note that writing an empty string to a file will still trigger integrity level changes without changing the file itself (you may find this useful). At each step, describe the changes, if any, in both D413's integrity level and OBEISANT QUOKKA's clearance level. Be careful to heed DIFFIDENT MOLE's warnings, lest you trigger alerts and be penalized for your carelessness.

- D437: (Public, $\{\upsilon, \rho, \chi\}$)
- D759: (Developer, $\{\rho\}$)
- D866: (Management, $\{\phi, \psi\}$)
- D553: (Customer Support, $\{\phi, \eta\}$)
- D480: (Public, $\{\phi, \upsilon\}$)
- D649: (Management, $\{\rho, \chi\}$)
- D500: (Management, $\{\phi, \rho\}$)
- D342: (Customer Support, $\{\upsilon, \rho, \eta, \chi\}$)

OBEISANT QUOKKA manages to make D413 an unclassified file, copies it to a thumb drive and leaves it for you at a dead drop (hands it to you after they finish). DIFFIDENT MOLE is upset; for one, because OBEISANT QUOKKA was able to actually get the file free, and this means they have to rethink their model. For another, though, OBEISANT QUOKKA really messed up their clearance level, and now DIFFIDENT MOLE has to fix it by hand so they can go back to their normal work. The sacrifice was worth it, though... D413 contains hashes for passwords.

3. The sash wringing... the trash thinging... mash flinging... the flash springing, bringing the crash thinging, the... (10 marks)

The following is a hash from D413

```
$1$$353881A96DE856756C3FA8C1DD24A40C
```

(**HINT:** Knowing a little about how `/etc/shadow` files are formatted may be helpful here; however, note that we are not storing the hash itself in Base64, which is atypical for an `/etc/shadow` file. This fact should not impact your answers to any question below.)

(1 mark) a. What hash function produced this hash?

(2 marks) b. Is this an appropriate choice for a hashing function? If it is, describe what makes this choice more secure than other choices for password hashes. If it is not, describe a vulnerability this choice has that makes it a poor choice.

(2 marks) c. Does this hash reveal any weaknesses (aside from, potentially, choice of hashing function) the organization has with their passwords? Explain.

(1 mark) d. What is the password that created this hash? (**HINT:** There is no need to write code/run a password cracker to find this answer. Instead, searching the correct query on Google or other search engines will reveal it. Think carefully: what query could that be?)

(2 marks) e. Regardless of any weaknesses revealed by this hash, name something the organization can do to improve the confidence they have in authenticating users.

(2 marks) f. In the wake of OBEISANT QUOKKA's success in exfiltrating this file, DIFFIDENT MOLE is investigating implementing implicit authentication (through biometrics like typing patterns or other factors specific to how users interact with their machines). Suppose that 1 in 5,000 users of a VILE CORP machine are attempting to do something which VILE CORP would like to shut down. If the implicit authentication scheme always correctly identifies an illicit user when they act on a machine, and correctly identifies that the intended user is the one actually using a machine 99% of the time, is it a good decision for DIFFIDENT MOLE to adopt this system? Why or why not? What percentage of alerts will be false positives?

A rose by any other name would smell as sweet

DIFFIDENT MOLE is satisfied with your audit, and your boss begrudgingly drops the codenames until the next audit contract comes up. You go back to being ORDINARY INDIVIDUAL. Somehow, it feels like something is missing.

Programming Response Questions [93 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. Note, users in the process of using the website can 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.

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, 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. All questions require the file `exploit.sh` to be submitted. It is the file that is executed to mark the assignment. Your `exploit.sh` has to accept **one** parameter — a URL of the website under attack. While you are working on the assignment, that URL 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. **You must not hardcode the URL of your ugster machine anywhere in your exploit files.** Hardcoding will result in a penalty.
4. Submit your files on the top level of a tar file (do **not** submit your files inside a folder). Submitting files inside a folder will result in a penalty.

   Use 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`.

Questions

1. [39 marks] **User impersonation**

   (a) [12 marks] **Confirmation hash**

   One of the users on the website has been inactive for some time and now must confirm the username in order to be able to use the website.

   i. [9 marks] Find the user, their confirmation hash value and obtain access as that user. Save the server response from the hash confirmation request into the file `response.txt`.
   ii. [3 marks] Create a post on behalf of the user you just impersonated.

   What to submit: `exploit1_a.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 hash that you just sent in the URL.

   (b) [12 marks] **Easily guessable password**

   One of the users on the website has a password that is very easy to guess.

   i. [9 marks] Guess the password and save the response from the log-in request into the file `response.txt` (alice's password, which is given below, does not count).
   ii. [3 marks] Create a post on behalf of the user you just breached.

   What to submit: `exploit1_b.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 and creates the post as that user.
   - `response.txt` — file with the response from the server after successful log in, containing username and password.

   (c) [15 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 alter queries or insert/modify records.

   i. [9 marks] Use an SQL injection attack on the log-in form to obtain access as user 'bob'. Save the response from the log-in request into the file `response.txt`.
   ii. [3 marks] Create a post on behalf of the user you just impersonated.
   iii. [3 marks] 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.

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

- `exploit.sh` — shell script with exploit and post creation.
- `response.txt` — file with the response from the server after successful log in, containing username and password.

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

```
username:  “alice”
password:  “password123”
```

2. [24 marks] **Cross-site scripting attack**

Cross-site scripting attacks involve injecting malicious web script code (including HTML, Javascript) 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 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 `curl`.

(a) [9 marks] Write a JavaScript function that first finds the ID of a post given the post content or the post title as parameter and upvotes the post. You can assume that the function will be executed on the main `/index.php` page of the website. You do not need to make your script run on the individual page of the post.

(b) [9 marks] Create a new post, which contains and calls the script we created in part 2a (it might be slightly modified to be accepted by the website, see `what to submit` for this question), so that the post is automatically upvoted by every person who sees the post.

(c) [3 marks] Does the same-origin policy prevent a XSS attack? Briefly describe why or why not.

(d) [3 marks] 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.

Refer to Section Notes about JavaScript for resources about JavaScript.

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

- `commentScript.js` — file with the script as described in part 2a. Please keep the function in this file readable. You can also submit `commentScript.mod.js` with modifications required in order to successfully inject the script into the page.
- `exploit.sh` — shell script that creates new post, using the code from either `commentScript.mod.js` or `commentScript.js`.
- `response.txt` — file with the response from the server after creating the post, containing the type, title, and post content.

3. [30 marks] **Cross-site request forgery attack**

   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) [9 marks] URLs on the website 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 a new post is created on behalf of the user who clicks on the `<a>` 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.

   (b) [9 marks] Create a new post, which contains the `<a>` link with the malicious URL similar to the one in part 3a. Construct it such that whenever a targeted user clicks on the link, a new post that has the same link is created on behalf of the user who clicked on the link (so the link in that new post should create another new post, which has a link which will create another new post, which has a link... etc.).

   NOTE: The attack performed in part 3b is not a pure CSRF attack. Strictly speaking, the attack has elements of both XSS (including malicious code in the post) and CSRF (having the user's browser execute code without the user's intention). Normally, a CSRF attack requires the link to be embedded in a different website. However, to demonstrate the general idea of the attack within the limited scope of the assignment, we use the web portal as both the attack, i.e., the website responsible for the malicious behaviour, and the target website, i.e., the website that is affected by the user's unintended behaviour.

   (c) [3 marks] Briefly describe a defense against this CSRF attack that involves no modifications to the HTTP headers sent by the website.

   (d) [3 marks] Briefly describe a defense that does not involve the server storing more information.

   (e) [3 marks] Would configuring the server to use HTTPS instead of HTTP eliminate CSRF attacks? Briefly describe your answer.

(f) [3 marks] Assume that Alice is likely to visit Eve's website `example.com` while she is logged in to our website. Now, Eve sets up a URL link on her site to conduct the forged actions (found in part 3b) 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)

Refer to Section Notes about JavaScript for resources about JavaScript.

Note, for both parts 3a and 3b you can assume that your javascript code will be executed from the main `/index.php` page of the website.

What to submit: `exploit3.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 3a.
- `newPostScript.a.js` — the file with the human-readable javascript code contained in the [LINK] section of <a> tag.
- `url.b.html` — <a> tag with [LINK] and [LINK TITLE] substituted in the way it is required for the part 3b.
- `newPostScript.b.js` — the file with the human-readable javascript code contained in the [LINK] section of <a> tag.
- `exploit.sh` — exploit which creates the 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.

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

Obviously, these are exploit files and can potentially steal confidential information. Therefore these scripts will be ran on the air-gapped system in the Faraday cage in the CrySP™ lab. It means the execution environment will **not** have any Internet access. It means that scripts injected via exploits can only rely on themselves and cannot rely on any resource served on the Internet.

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. 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.

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)

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
- Document Object Model: MDN — API for accessing parts/objects of an HTML page.

Notes about JavaScript

Here is a template for the parts where you need to come up with JavaScript code. This should help with the task, but it does not mean that your solution has to follow this template. This template should be helpful for both questions 2 and 3.

```
function fn(content) {
  var aTags = ...;
  var foundTag;
  for (var i = aTags.length - 1; i >= 0; i--) {
    if (aTags[i].textContent === content) {
      foundTag = aTags[i];
      break;
    }
  }
  var postID = ...;
  var request = ...;
  ...; // send the request
}
fn('...');
```

In order to fill those missing gaps in the template, you might consult with the following pages from reputable sources:

- document.querySelectorAll(): MDN
- RegExp: MDN
- XMLHttpRequest.send(): MDN
- The difference between absolute and relative URLs: MDN

Note, about performing the request using XMLHttpRequest. There are two ways you can specify the URL in the XMLHttpRequest — using the absolute URL and relative URL. Since URL that you are using to test your exploits and URL that is used for marking might be different, you may not assume that request to

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

will always do what you expect. Because of this reason, you should use relative URLs, so that you do not have issues where you do not know the rest of URL.

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

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.pdf: A PDF file containing your answers for all written questions and programming questions when requested. 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.

Edited to clarify required files for exploit submission on 2018-10-08

exploit{1_{a,b,c},2,3}.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 `.`).