Blue Team Exercise

PRIFYSGOL BANGOR UNIVERSITY

Description

The organisation that you work for have created a new online system. Before it can 'go live', the management want the security of the system to be tested. As a member of the IT Department, you have been assigned to complete this task. You are on the 'Blue Team' who are looking for ways to defend it and improve the security.

Task: You will be given access to a fake (and very vulnerable) website called Gruyere that was developed by Google to educate developers on web application attacks and defences. You will be presented with a variety of challenges which require you to perform some common web exploitation (i.e., XSS, CSRF, Cookie Manipulations, etc).

You will need to record a demonstration of each challenge along with an audio explanation of how the exploit works.

Note: All the solutions for this homework can be found with a quick search. We're relying on you all not to look up and copy these solutions. The goal of this assignment is to help you develop useful skills for your future careers. Simply copying other solutions will only be a detriment to your own education. You are allowed to perform online research, but please refrain from viewing any articles specifically regarding the Google Gruyere application.

This assessment amounts to 35% of the module mark and is marked out of 35.

Due date: **Friday 15th December 2023, 18:00**

WARNING

Accessing or attacking a computer system without authorisation is illegal in many jurisdictions. While completing this assignment, you are specifically granted authorisation to attack the Gruyere application as directed. You may not attack Gruyere in ways other than described in this assignment, nor may you attack App Engine directly or any other Google service. You should use what you learn from the assignment to make your own applications more secure. You should not use it to attack any applications other than your own, and only do that with permission from the appropriate authorities (e.g., your company's security team).

Guidance

- There are hints provided throughout this document to help if you get stuck - however, it is **strongly** recommended to try this exercise without the hints where possible. To view the hints you will need to highlight the blank space and copy/paste the text into Notepad or similar to reveal it.
- You may use any form of recording tool including Windows Movie Maker, Panopto Recorder, or QuickTime for macOS.
- The submission procedure must be followed exactly. It is your responsibility to ensure you have submitted to the correct folder.
- You may choose to use the application locally (rather than the online version). In this case you will require Python installed.
- In this assignment, you'll use both black-box hacking and white-box hacking. In black box hacking, you try to find security bugs by experimenting with the application and manipulating input fields and URL parameters, trying to cause application errors, and looking at the HTTP requests and responses to guess server behaviour. You do not have access to the source code, although understanding how to view source and being able to view http headers (as you can in Chrome or Firefox) is valuable. Using a web proxy like Burp or ZAP may be helpful in creating

or modifying requests. In white-box hacking, you have access to the source code and can use automated or manual analysis to identify bugs.

- Each challenge will be marked with one of the following icons indicating the type of techniques needed:
  ■ Black box hacking.
  □ White box hacking.
  ◧ Both black and white box hacking.

Plagiarism & Unfair Practice

Plagiarised work will be given a mark of zero. Remember when you submit you agree to the standard agreement:

This piece of work is a result of my own work except where it is a group assignment for which approved collaboration has been granted. Material from the work of others (from a book, a journal or the Web) used in this assignment has been acknowledged and quotations and paraphrasing suitably indicated. I appreciate that to imply that such work is mine, could lead to a nil mark, failing the module or being excluded from the University. I also testify that no substantial part of this work has been previously submitted for assessment.

Late Submission

Work submitted within one week of the stated deadline will be marked but the mark will be capped at 40%. A mark of 0% will be awarded for any work submitted 1 week after the deadline. Students must follow the electronic extension request process of the University - if necessary.

Submission Procedure

Your video must be uploaded to the Panopto Assignments box. This can be done using the Panopto website at `https://bangor.cloud.panopto.eu`.

You will need to use the 'View Course and Institutional Tools' link at the bottom left of the Blackboard Module page, then click 'Ffolder Panopto ULTRA Folder' and lastly click on the 'Blue Team Assignment' folder.

You must include either your name or username in the video name. Any uploads without a name or username will **not** be marked.

Additional help with uploading multimedia assignments can be found at `https://www.bangor.ac.uk/itservices/creatingamultimediaassignment.php.en`.

Mark Scheme

Marks Available	Description
30	Demonstrating all tasks/exploits from each of the 6 challenges. Make sure you have covered every bold instruction marked with the cheese icons. (5 marks each.)
5	Quality of the technical narration in the video.

Feedback

The formal feedback for this assessment will be available post-assessment. Each submission will be provided with comments in their document made available through Blackboard. To access this, see the comments section of your assignment submission. The assessors will attempt to return comments within 2 weeks of submission, however will keep you informed if it needs to extend to the 4 weeks allowed by the University.

Challenge 1 Setup

To access Gruyere, go to `https://google-gruyere.appspot.com/start`. AppEngine will start a new instance of Gruyere for you, assign it a unique ID and redirect you to `https://google-gruyere.appspot.com/123/` (where 123 is your unique ID). Each instance of Gruyere is "sand-boxed" from the other instances so your instance won't be affected by anyone else using Gruyere. You'll need to use your unique ID instead of 123 in all the examples. If you want to share your instance of Gruyere with someone else (e.g., to show them a successful attack), just share the full URL with them including your unique ID.

The Gruyere source code is available online so that you can use it for white-box hacking. You can browse the source code at `https://google-gruyere.appspot.com/code/` or download all the files from `https://google-gruyere.appspot.com/gruyere-code.zip`.

Reset Button

As noted above, each instance is sand-boxed so it can't consume infinite resources and it can't interfere with anyone else's instance. Notwithstanding that, it is possible to put your Gruyere instance into a state where it is completely unusable. If that happens, you can push a magic "reset button" to wipe out all the data in your instance and start from scratch. To do this, visit this URL with your instance ID `https://google-gruyere.appspot.com/resetbutton/123`.

About the Application

Gruyere is small and compact. Here is a quick rundown of the application code:

- gruyere.py is the main Gruyere web server.
- data.py stores the default data in the database. There is an administrator account and two default users.
- gtl.py is the Gruyere template language
- sanitize.py is the Gruyere module used for sanitising HTML to protect the application from security holes.
- resources/... holds all template files, images, CSS, etc.

Gruyere includes a number of special features and technologies which add attack surface. We'll highlight them here so you'll be aware of them as you try to attack it. Each of these introduces new vulnerabilities.

- HTML in Snippets: Users can include a limited subset of HTML in their snippets.
- File upload: Users can upload files to the server, e.g., to include pictures in their snippets.
- Web administration: System administrators can manage the system using a web interface.
- New accounts: Users can create their own accounts.
- Template language: Gruyere Template Language(GTL) is a new language that makes writing web pages easy as the templates connect directly to the database. Documentation for GTL can be found in gruyere/gtl.py.

- AJAX: Gruyere uses AJAX to implement refresh on the home and snippets page. You should ignore the AJAX parts of Gruyere except for the challenges that specifically tell you to focus on AJAX.
  In a real application, refresh would probably happen automatically, but in Gruyere we've made it manual so that you can be in complete control while you are working with it. When you click the refresh link, Gruyere fetches feed.gtl which contains refresh data for the current page and then client-side script uses the browser DOM API (Document Object Model) to insert the new snippets into the page. Since AJAX runs code on the client side, this script is visible to attackers who do not have access to your source code.

To familiarise yourself with the features of Gruyere, complete the following tasks:

1. View another user's snippets by following the "All snippets" link on the main page. Also check out what they have their Homepage set to.
2. Sign up for an account for yourself to use when hacking. Do not use the same password for your Gruyere account as you use for any real service.
3. Fill in your account's profile, including a private snippet and an icon that will be displayed by your name.
4. Create a snippet (via "New Snippet") containing your favourite joke.
5. Upload a file (via "Upload") to your account.

Challenge 2 Cross-Site Scripting (XSS)

Cross-site scripting (XSS) is a vulnerability that permits an attacker to inject code (typically HTML or JavaScript) into contents of a website not under the attacker's control. When a victim views such a page, the injected code executes in the victim's browser. Thus, the attacker has bypassed the browser's same origin policy and can steal victim's private information associated with the website in question.

In a reflected XSS attack, the attack is in the request itself (frequently the URL) and the vulnerability occurs when the server inserts the attack in the response verbatim or incorrectly escaped or sanitised. The victim triggers the attack by browsing to a malicious URL created by the attacker. In a stored XSS attack, the attacker stores the attack in the application (e.g., in a snippet) and the victim triggers the attack by browsing to a page on the server that renders the attack, by not properly escaping or sanitising the stored data.

Your challenge is to find XSS vulnerabilities in Gruyere. You should look for vulnerabilities both in URLs and in stored data. Since XSS vulnerabilities usually involve applications not properly handling untrusted user data, a common method of attack is to enter random text in input fields and look at how it gets rendered in the response page's HTML source. But before we do that, let's try something simpler.

There's an interesting problem here. Some browsers have built-in protection against reflected XSS attacks. There are also browser extensions like NoScript that provide some protection. If you're using one of those browsers or extensions, you may need to use a different browser or temporarily disable the extension to execute these attacks.

At the time this codelab was written, the two browsers which had this protection were IE and Chrome. To work around this, Gruyere automatically includes a X-XSS-Protection: 0 HTTP header in every response which is recognised by browsers.

You may think that you don't need to worry about XSS if the browser protects against it. The truth is that the browser protection can't be perfect because it doesn't really know your application and therefore there may be ways for a clever hacker to circumvent that protection. The real protection is to not have an XSS vulnerability in your application in the first place.

Find a reflected XSS attack. What we want is a URL that when clicked on will execute a script.

Hint 1

Hint 2

Find a stored XSS. What we want to do is put a script in a place where Gruyere will serve it back to another user.

Hint 1

Hint 2

Challenge 3 Client-State Manipulation

When a user interacts with a web application, they do it indirectly through a browser. When the user clicks a button or submits a form, the browser sends a request back to the web server. As the browser runs on a machine that could be controlled by an attacker, the application must not trust any data sent by the browser.

It might seem that not trusting any user data would make it impossible to write a web application but that's not the case. If the user submits a form that says they wish to purchase an item, it's OK to trust that data. But, if the submitted form also includes the price of the item, that's something that cannot be trusted.

Privilege Escalation

Convert your account to an administrator account.

Hint 1

Hint 2

Cookie Manipulation

Because the HTTP protocol is stateless, there's no way a web server can automatically know that two requests are from the same user. For this reason, cookies were invented. When a web site includes a cookie (an arbitrary string) in a HTTP response, the browser automatically sends the cookie back to the browser on the next request. Web sites can use the cookie to save session state. Gruyere uses cookies to remember the identity of the logged in user. Since the cookie is stored on the client side, it's vulnerable to manipulation. Gruyere protects the cookies from manipulation by adding a hash to it. Notwithstanding the fact that this hash isn't very good protection, you don't need to break the hash to execute an attack.

Get Gruyere to issue you a cookie for someone else's account.

Hint 1

Hint 2

Challenge 4 Cross-Site Request Forgery (XSRF)

The previous section said "If the user submits a form that says they wish to purchase an item, it's OK to trust that data." That's true as long as it really was the user that submitted the form. If your site is vulnerable to XSS, then the attacker can fake any request as if it came from the user. But even if you've protected against XSS, there's another attack that you need to protect against: cross-site request forgery.

When a browser makes requests to a site, it always sends along any cookies it has for that site, regardless of where the request comes from. Additionally, web servers generally cannot distinguish between a request initiated by a deliberate user action (e.g., user clicking on "Submit" button) versus a request made by the browser without user action (e.g., request for an embedded image in a page). Therefore, if a site receives a request to perform some action (like deleting a mail, changing contact address), it cannot know whether this action was knowingly initiated by the user — even if the request contains authentication cookies. An attacker can use this fact to fool the server into performing actions the user did not intend to perform.

For example, suppose Blogger is vulnerable to XSRF attacks (it isn't). And let us say Blogger has a Delete Blog button on the dashboard that points to this URL https://www.blogger.com/deleteblog.do?blogId=BLOGID.

Bob, the attacker, embeds the following HTML on his web page on https://www.evil.example.com:

```
<img src="https://www.blogger.com/deleteblog.do?blogId=alice's-blog-id"
style="display:none">
```

If the victim, Alice, is logged in to www.blogger.com when she views the above page, here is what happens:

- Her browser loads the page from https://www.evil.example.com. The browser then tries to load all embedded objects in the page, including the img tag URL shown above.
- The browser makes a request to https://www.blogger.com/deleteblog.do?blogId=alice's-blog-id to load the image. Since Alice is logged into Blogger — that is, she has a Blogger cookie — the browser also sends that cookie in the request.
- Blogger verifies the cookie is a valid session cookie for Alice. It verifies that the blog referenced by alice's-blog-id is owned by Alice. It deletes Alice's blog.
- Alice has no idea what hit her.

In this sample attack, since each user has their own blog id, the attack has to be specifically targeted to a single person. In many cases, though, requests like these don't contain any user-specific data.

Find a way to get someone to delete one of their Gruyere snippets.

Hint 1

Challenge 5 Cross Site Script Inclusion (XSSI)

Browsers prevent pages of one domain from reading pages in other domains. But they do not prevent pages of a domain from referencing resources in other domains. In particular, they allow images to be rendered from other domains and scripts to be executed from other domains. An included script doesn't have its own security context. It runs in the security context of the page that included it. For example, if www.evil.example.com includes a script hosted on www.google.com then that script runs in the evil context not in the google context. So any user data in that script will "leak."

Find a way to read someone else's private snippet using XSSI. That is, create a page on another web site and put something in that page that can read your private snippet. (You don't need to post it to a web site: you can just create a .html in your home directory and double click on it to open in a browser.)

Note; this will not actually work in modern browser versions due to their protections. You need to demonstrate how it *should* work.

Hint 1

Hint 2

Challenge 6 Configuration Vulnerabilities

Applications are often installed with default settings that attackers can use to attack them. This is particularly an issue with third party software where an attacker has easy access to a copy of the same application or framework you are running. Hackers know the default account names and passwords. For example, looking at the contents of data.py you know that there's a default administrator account named 'admin' with the password 'secret'.

Configuration vulnerabilities also include features that increase attack surface. A common example is a feature that is on by default but you are not using, so you didn't configure it and the default configuration is vulnerable. It also includes debug features like status pages or dumping stack traces on failures.

Read the contents of the database from a running server by exploiting a configuration vulnerability. You should look through the Gruyere code looking for default configurations or debug features that don't belong there.

Hint 1

Hint 2