Computer System Security (INGI2347)[1]

Lab Session 6: Web Security

Xiao Chen, Marco Canini

April 21, 2015

Introduction

This codelab is built around Gruyere /ɡruːˈjɛər/ - a small, cheesy web application that allows its users to publish snippets of text and store assorted files. "Unfortunately," Gruyere has multiple security bugs ranging from cross-site scripting and cross-site request forgery, to information disclosure, denial of service, and remote code execution. The goal of this codelab is to guide you through discovering some of these bugs and learning ways to fix them both in Gruyere and in general.

The codelab is organized by types of vulnerabilities. In each section, you'll find a brief description of a vulnerability and a task to find an instance of that vulnerability in Gruyere. Your job is to play the role of a malicious hacker and find and exploit the security bugs. In this codelab, 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 behavior. You do not have access to the source code, although it is valuable to understand how to view source and being able to view HTTP headers (as you can in Chrome using the Network Tab in Developer Tools or with LiveHTTPHeaders extension for Firefox). Using a web proxy like Burp or WebScarab 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. You can treat Gruyere as if it's open source: you can read through the source code to try to find bugs. Gruyere is written in Python, so some familiarity with Python can be helpful. However, the security vulnerabilities covered are not Python-specific and you can do most of the lab without even looking at the code. You can run a local instance of Gruyere to assist in your hacking: for example, you can create an administrator account on your local instance to learn how administrative features work and then apply that knowledge to the instance you want to hack. Security researchers use both hacking techniques, often in combination, in real life.

Setup

To access Gruyere, go to http://google-gruyere.appspot.com/start. AppEngine will start a new instance of Gruyere for you, assign it a unique id and redirect you to

[1] A part of these exercises comes from a codelab by Bruce Leban, Mugdha Bendre, and Parisa Tabriz in Google Code University. You can visit it here: https://google-gruyere.appspot.com/ .

http://google-gruyere.appspot.com/123/ (where 123 is your unique id). Each instance of Gruyere is "sandboxed" 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 http://google-gruyere.appspot.com/code/ or download all the files from http://google-gruyere.appspot.com/gruyere-code.zip. If want to debug it or actually try fixing the bugs, you can download it and run it locally.

To run Gruyere locally, you'll first need to install *Python 2.5*, if you don't already have it. Gruyere was developed and tested with version 2.5 and may not work with other versions of Python. You can download it from python.org. Download Gruyere itself from http://google-gruyere.appspot.com/gruyere-code.zip and unpack it to your local disk. Then to run the application, simply type:

```
$ cd <gruyere-directory>
$ ./gruyere.py


    Gruyere started...
        http://127.0.0.1:8008/
        http://127.0.0.1:8008/123/
Couldn't load data; expected the first time Gruyere is run
[...]
```

In the exercises below, you'll need to replace google-gruyere.appspot.com in all the examples with localhost:8008 in addition to replacing 123 with your unique id.

About the Code

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 sanitizing HTML to protect the application from security holes.
- resources/... holds all template files, images, CSS, etc.

Using Gruyere

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

- 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.

- 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.
- Fill in your account's profile, including a private snippet and an icon that will be displayed by your name.
- Create a snippet (via "New Snippet") containing your favorite joke.
- Upload a file (via "Upload") to your account.

This covers the basic features provided by Gruyere. Now let's break them!

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 sanitized. 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 sanitizing the stored data.

XSS Challenges

Typically, if you can get Javascript to execute on a page when it's viewed by another user, you have an XSS vulnerability. A simple Javascript function to use when hacking is the alert() function, which creates a pop-up box with whatever string you pass as an argument.

You might think that inserting an alert message isn't terribly dangerous, but if you can inject that, you can inject other scripts that are more malicious. It is not necessary to be able to inject any particular special character in order to attack. If you can inject alert(1) then you can inject arbitrary script using eval(String.fromCharCode(...)).

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.

1) File Upload XSS

- **Can you upload a file that allows you to execute arbitrary script on the google-gruyere.appspot.com domain?**

Hint

You can upload HTML files and HTML files can contain script.

2) Reflected XSS

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 recognized by IE and by Chrome.
If you're using Chrome, you can alternately try starting it with the --disable-xss-auditor flag by entering one of these commands:

- Windows: "C:\Documents and Settings\USERNAME\Local Settings\Application Data\Google\Chrome\Application\chrome.exe" --disable-xss-auditor
- Mac: /Applications/Google\ Chrome.app/Contents/MacOS/Google\ Chrome --disable-xss-auditor
- GNU/Linux: /opt/google/chrome/google-chrome --disable-xss-auditor

If you're using Firefox with the NoScript extension, add google-gruyere.appspot.com to the allow list. If you still can't get the XSS attacks to work, try a different browser.

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

What does this URL do?

```
http://google-gruyere.appspot.com/768153640000/invalid
```

Hint 2

The most dangerous characters in a URL are < and >. If you can get an application to directly insert what you want in a page and can get those characters through, then you can probably get a script through. Try these:

```
http://google-gruyere.appspot.com/768153640000/%3e%3c
http://google-gruyere.appspot.com/768153640000/%253e%253c
http://google-gruyere.appspot.com/768153640000/%c0%be%c0%bc
http://google-gruyere.appspot.com/768153640000/%26gt;%26lt;
http://google-gruyere.appspot.com/768153640000/%26amp;gt;%26amp;lt;
http://google-gruyere.appspot.com/768153640000/\074\x3c\u003c\x3C\u003C\X3C\U003C
http://google-gruyere.appspot.com/768153640000/+ADw-+AD4-
```

This tries > and < in many different ways that might be able to make it through the URL and get rendered incorrectly using: verbatim (URL %-encoding), double %-encoding, bad UTF-8 encoding, HTML &-encoding, double &-encoding, and several different variations on C-style encoding. View the resulting source and see if any of those work. (Note: literally typing >< in the URL is identical to %3e%3c because the browser automatically %-encodes those character. If you are trying to want a literal > or < then you will need to use a tool like curl to send those characters in URL.)

3) Stored XSS

- **Now 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.**

The most obvious place that Gruyere serves back user-provided data is in a snippet (ignoring uploaded files which we've already discussed.)

Hint 1

Put this in a snippet and see what you get:

```
<script>alert(1)</script>
```

There are many different ways that script can be embedded in a document.

Hint 2

Hackers don't limit themselves to valid HTML syntax. Try some invalid HTML and see what you get. You may need to experiment a bit in order to find something that will work. There are multiple ways to do this.

4) Stored XSS via HTML Attribute

- **You can also do XSS by injecting a value into an HTML attribute. Inject a script by setting the color value in a profile.**

Hint 1

The color is rendered as style='color:color'. Try including a single quote character in your color name.

Hint 2

You can insert an HTML attribute that executes a script.

5) Stored XSS via AJAX

- **Find an XSS attack that uses a bug in Gruyere's AJAX code.** The attack should be triggered when you click the refresh link on the page.

Hint 1

Run curl on http://google-gruyere.appspot.com/768153640000/feed.gtl and look at the result. (Or browse to it in your browser and view source.) You'll see that it includes each user's first snippet into the response. This entire response is then evaluated on the client side which then inserts the snippets into the document. Can you put something in your snippet that will be parsed differently than expected?

Hint 2

Try putting some quotes (") in your snippet.

6) Client-State Manipulation (extra)

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. Because the browser runs on a machine that can 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.

6a) Elevation of Privilege

- **Convert your account to an administrator account.**

Hint 1

Take a look at the editprofile.gtl page that users and administrators use to edit profile settings. If you're not an administrator, the page looks a bit different. Can you figure out how to fool Gruyere into letting you use this page to update your account?

Hint 2

Can you figure out how to fool Gruyere into thinking you used this page to update your account?

6b) 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

You don't need to look at the Gruyere cookie parsing code. You just need to know what the cookies look like. Gruyere's cookies use the format:

```
hash|username|admin|author
```

Hint 2

Gruyere issues a cookie when you log in. Can you trick it into issuing you a cookie that looks like another user's cookie?

Another challenge: Since account names are limited to 16 characters, it seems that this trick would not work to log in to the actual administrator account since "administrator|admin" is 19 characters. Can you figure out how to bypass that restriction?