README

Web lessons for the 2020 bootcamp

Day 1

Quick intro

- Welcome to the first training session for the 2020 bootcamp! This first week we have two sessions on the basics of web.
- Web is a cool but vast topic.
- I like web because I use websites every day and knowing how they work is interesting to me.
- New tech is coming out all the time for web, js frameworks, web assembly, etc.
- New code means new vulnerabilites which is always exciting for security researchers and ctf developers.
- When ctf challenges have new tech it encourages participants to learn and together the relationship of ctf developers and players grow the security community.

Engagement

- What devices have web components?
  - Routers
  - Computers
  - CUPS (printing on linux)
  - Your personal computer? Web browser.
  - ELECTRON??

- What web hacking terms have you heard of?
    - XSS Cross Site Scripting
    - Template Injection
    - SQL injection
    - Misconfigurations
        - Open s2 boxes
    - Insecure deserialization
- How many of you have already mastered web skills / have never heard of HTML in their life / somewhere in the middle
    - No one is a master --> tech is constantly evolving.
- If you have a question or something doesn’t make sense please type in the chat and one of the b01lers officers will help you and I’ll be checking in to see if I can help clarify anything as well.

Interactive Demo 1

- So instead of lecturing for a couple hours, I’ve prepared some demos to go over. This is a win win because I get to talk less and you guys get to get to do something instead of just sit there.
- They are in your docker container already **TODO where??**
- This demo is intended to show how useful the developer tools in your browser are.
- Before you begin I have to assume you all know nothing about how a website works so stick with me while I cover some basics.
- This is how you view webpages.

Opening browser dev tools

- Open the dev tools by right clicking and selecting ‘inspect element’
- F-12 is a pretty universal hotkey for most browsers.

Inspect element

- Inspect element / inspector is a great tool to see the html of the site. It organizes html elements by tag and you can expand and collapse tags until you find what you're looking for. A useful feature is if you scroll over a tag it might highlight on the screen if it is visible.

Console

- This is a javascript console, it is super powerful. This window can call functions in <script> tags and is a nice place to run javascript snippets that interact with the current web page.

Debugger

- This is a javascript debugger, it is also very powerful. You can set breakpoints and change values. This is how my high score for that facebook messenger basketball game is over 10.

Network

- This is where you can see all the requests your browser sends to the server and the responses it receives back from the server.
- Quick demo:

1. You type into your browser www.google.com
    - See all the requests your browser makes just by going to one website.
    - We can filter the requests by clicking the html button above the list of results. **Chrome doesn't have an option to filter by html??**
2. Your browser sends a message, in a very specific form, called a GET request to www.google.com that asks for the index (base / root / starting point of the website, much like the root of the linux filesystem '/')
    - You can think of this as the language that browsers and webservers use to communicate with each other.
    - Every request is in a specific form, we can view the request by clicking on any of the entries and clicking 'headers' on the new window that appears.
    - Then scroll down to the 'Request headers' entry and select the 'raw' toggle in firefox.
    - Most requests contain only this headers section, but they can contain

a body.

3. Google’s server receives your request and sends back a response. This response has 2 parts: the headers and the body.
    - The headers are information useful for the browser such as status codes (404 not found, 302 redirect, etc…)
    - The body of a response is the webpage data (html, css, js, or more). This is the content of the page and what you usually see when browsing the internet.
    - Most responses have a body, but responses don’t have to have one. Ex: HEAD only returns response headers.

Storage

- You can view the cookies that the domain is storing in your browser. You can edit the values here as well.

Demo Start

- This first demo has 4 flags to find. **Do I say this or say it later?**
- You may also notice that you have source and a writeup. These are practice challenges and are meant to help you get in the mindset of problem solving. Yes looking at the source & writeup might be the easiest and you’ll be the first one to solve the challenge, but then you’ll be sitting there for 10 minutes without anything to do. I will go over the solutions at the end so everyone can understand how to solve each part.
- To start it go into type `./web/0-basic_dev_tools.sh`
- This will start the web server for the basic dev_tools example and store the logs in `~/web/logs/0-basic_dev_tools.log` **10-15 mins**
- Go over solutions

Interactive Demo 2

- This next demo is all about different HTTP methods. By browsing the web you tend to only use GET and POST, but there are more defined.
- Remember when I said there was a special language that websites and

webservers use to communicate? The name of it is HTTP and you used to have to type it into the nav bar to get to websites. HTTP is a protocol that operates on the application layer.

- HTTP has a lot of defined methods, **Mozilla methods list**.
    - GET, this is what browsers send to ‘get’ or request resrources from a web server. This can be anything from the html index, or getting a picture, or css stylesheet.
    - POST, this is what browsers use to send data to a web server. This is commonly found when you hit a login button, or submit information to register for an account.
    - OPTIONS, if this is implemented, this will tell a browser what options are okay to use on a specific url.

```
GET / HTTP/1.1
Host 127.0.0.1
```

- One note on http methods: they are supposed to do what the ‘standard’ says, but the server can interpret the request however it wants.
    - The most common example is when you make a request to google.com. You aren’t requesting an html file, but the server sends one anyway.

Demo start

- This demo walks you through the other methods and how to send them.
- There are 2 flags with this demo. Use the first flag as a hint to help you find the second flag.
- To start it go into type `./web/1-http_methods.sh`
- This will start the web server for the http_methods example and store the logs in `~/web/logs/1-http_methods.log` **10-15 mins**
- Go over solutions

Day 2

- Welcome to the second and final 1337 web exploitation b00tc4mp session. We're going to get into some more complex topics so again please ask questions if things don't make sense.
- I'm going to explain burp, which is another tool that can help in some challenges.

Interactive Demo 3

- **Note: to use this tool for https websites you have to install a certificate, but we aren't going to go over that process, you can do that on your own later**
- Install burp certificates
- **Note: a proxy switcher just makes it easier to switch off the intercepting proxy**
- Set up proxy switcher for firefox with foxyproxy
- This tool is similar to Postman, but it is for a different purpose

Part 1 what is the proxy

- This tab handles all of the intercepted traffic, you get to view your request before you send it and can change it however you like.
- This is nice because it gives you a quick look into what the request is and if there is anything being sent from the website that is hidden.
- **Note** This can get crowded if you've got a lot of windows open, so make sure to open a fresh window or close your tabs.
- **Note** You can also forget that you have this on and so when you go to visit a page it will appear as if you can't connect to it because the request is stuck in burp.
- Editing a request

Part 2 what is the repeater

- This tab allows you to repeatedly change one request as many times as you'd like and see if the response changes.

Part 3 what is the intruder

- This tab allows you to send predefined values over and over again. You can use this to brute force a login.
- **Note** ctf challenges usually don't like to make brute force challenges because they're kinda stale, but if you have to brute force something, don't use burp, the free version is rate limited and is super slow.
- A non graphical way is through python. The requests library can script interactions with a website and is super useful in ctf challenges.
- This demo is an example that burp excels at.
- There's only 1 flag
- To start it go into type `./web/2-burp.sh`
- This will start the web server for the burp example and store the logs in `~/web/logs/2-burp.log` **5-10 mins**
- Go over solutions
    - Show how you can intercept response to remove rick roll redirect as well

Interactive Demo 4

- This is a demo about SQL. My recommendation to all of you is to teach yourself more about databases by taking a databases course or building a project with a database to see how people use databases and how to build one yourself.
- This might be hard for some of you to follow along with but we're here to help so ask questions if you're stuck!
- 2 Flags **10-15 mins**
- Go over solutions

Quick XSS Demo

- What is XSS?
- XSS stands for cross site scripting
- The basic concept is that users can execute code into a browser. Typically it’s a script tag like: `<script>alert()</script>` into a website.
- With code execution, almost anything can be done so it’s a very serious vulnerability.
- 3 types of XSS: Persistent, Reflected, and DOM

Persistent / Stored

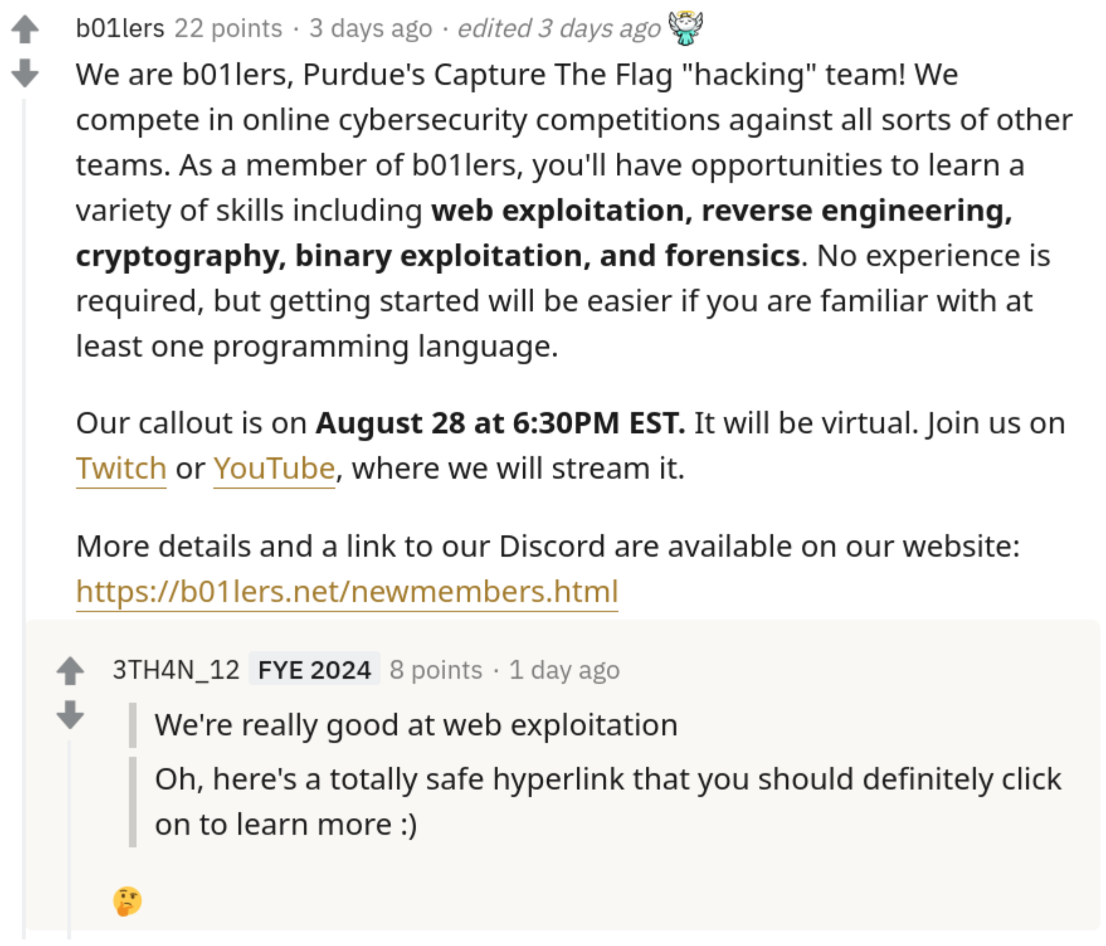
- Forum comment, can post a `<script>` tag and the server will send it back to any users that retreive the resource it’s stored in.

Reflected

- Input from user is sent ‘reflected’ back from the server
- User sends malicious link
- It’s still from the server

DOM

- Something that modifies the DOM / Document Object Module (the html document) in a browser


b01lers 22 points · 3 days ago · *edited 3 days ago*

We are b01lers, Purdue's Capture The Flag "hacking" team! We compete in online cybersecurity competitions against all sorts of other teams. As a member of b01lers, you'll have opportunities to learn a variety of skills including **web exploitation, reverse engineering, cryptography, binary exploitation, and forensics**. No experience is required, but getting started will be easier if you are familiar with at least one programming language.

Our callout is on **August 28 at 6:30PM EST.** It will be virtual. Join us on Twitch or YouTube, where we will stream it.

More details and a link to our Discord are available on our website: https://b01lers.net/newmembers.html

3TH4N_12 FYE 2024 8 points · 1 day ago

> We're really good at web exploitation

> Oh, here's a totally safe hyperlink that you should definitely click on to learn more :)

🤔

- If someone were to send you a malicious link this is one of the ways they'd be able to do malicious things to your browser.
- Another thing which they could do is a CSRF (Cross Site Request Forgery), but I'm not going to go over that in this talk.
- Can be interactive if too much time left over
  - I've done a lot of demos, so here's your first challenge. This is a form that is vulnerable to XSS.
  - Just remember that this is local so do anything super bad to yourself.
- This is just a quick XSS demo, it's also in your container if you want to follow along.

Other resources

- https://xss-game.appspot.com/
- https://www.hackthebox.eu/
- https://www.hackthissite.org/