All this text wasn’t written by me (Tiago a.k.a TigaxMT).

All of it was transcribed by me from HackerOne video lessons.

All credits and thanks go to them.

Introduction

WELCOME

Welcome to Hacker 101! I'm Cody Brocious and I'll be your instructor. Throughout this class, here's what you will learn:

- How to identify, exploit, and remediate the top web security vulnerabilities , as well as many more arcane bugs
- How to properly handle cryptography
- How to design and review applications from a security standpoint
- How to operate as a bug bounty hunter or security consultant
- Much more

You can use any OS you'd like for this course, so long as you're able to run Java apps.

I strongly recommend you brush up on performing web requests with your language of choice. If using Python, the 'requests' package is my personal favorite.

MUST-HAVE TOOLS

- Burp Proxy(free edition is perfectly fine)
    - This allows you to watch all HTTP(S) communication, intercept and modify requests, and replay existing requests
    - You will use this constantly in both your coursework and exams
- Firefox
    - Firefox allows you to set proxy settings specifically in the browser, rather than setting them system-wide
    - This will be your friend when you're testing, to isolate an application

INTRODUCTION TO SECURITY

You're here to break things before anyone else does, so that vulnerabilities can be fixed before attackers get to them.

To do that, you need to understand how attackers operate, what their goals are, and how they think.

THINKING LIKE A BREAKER

The most important tenet of the breaker mindset is this: pushing a button is the most effective way to discover what it does.

If you don’t understand what an application is doing and why it’s doing it, you’re going to have a hard time finding ways to break it.

IMBALANCE

The key difference between defending and attacking is this:

Defenders have to find every bug; attackers only need to find a few.
This means that attackers will always have the advantage over defenders.

The primary consequence of this imbalance is that you will never find every bug, especially under time pressure.

This means you have to prioritize to ensure that where there are still bugs , the impact will be relatively low.
Making an accurate assessment of high-risk areas is critical.

ATTACKER GOALS

When assessing an application, find every bit of functionality you can. Once you have a rough list of all of the bits of the application, consider this: If I was an attacker, what would my goal be?

Maybe I’d want credit card numbers from an e-commerce site; maybe I’d want to destroy or falsify data in a server monitoring application.

PRIORITIZATION

Once you have a good picture of what an attacker might want, you can start to rank areas of the application in terms of payoff: if I compromise area X, does that give me low-value information or high-value? What about Y instead?

When possible, asking developers the question “what keeps you up at night?” will often point areas to check.

FINDINGS

For the purpose of this course, you should include the following for each vulnerability:

- Title – E.g. “Reflected Cross-Site Scripting in profiles”
- Severity
- Description – Brief description of what the vulnerability is
- Reproduction Steps – Brief description of how to reproduce the bug; preferably with a small proof of concept
- Impact – What can be done with the vulnerability?
- Mitigation – How is it fixed?
- Affected assets – Generally a list of affected URLs.

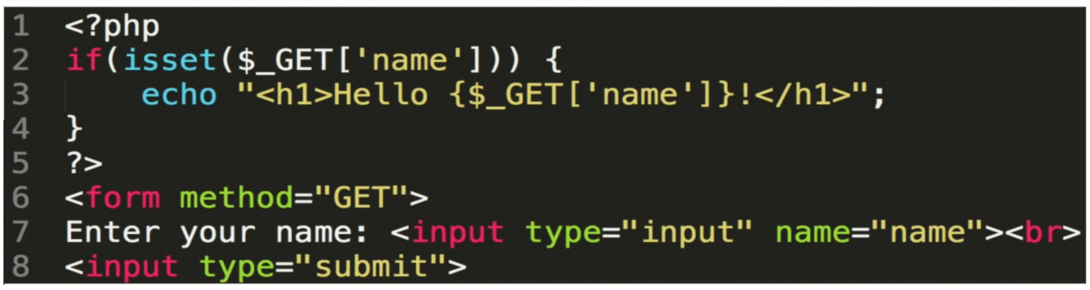
SEVERITY

This is handled differently just about everywhere, but I recommend basing severity on difficulty of exploitation and potencial business impact. The following rankings are what I use:

- Informational – Issue has no real impact
- Low – The bussiness impact is minimal
- Medium – Potential to cause harm to users, but not revealing data
- High – Potential to reveal user data or aids in exploitation of other vulnerabilities
- Critical – High risk of personal/confidential data exposure, general system compromise, and other severe impacts to the business

CONTRIVED EXAMPLE

Let’s take a look at a contrived but every common example:



```php
<?php
if(isset($_GET['name'])) {
    echo "<h1>Hello {$_GET['name']}!</h1>";
}
?>
<form method="GET">
Enter your name: <input type="input" name="name"><br>
<input type="submit">
```

SPOT THE BUG

The code is pretty straightforward:

1. Was a ‘name’ parameter passed via GET?
   a) If so, print ‘Hello <name>!’ in an h1 tag
2. Print a form for the user to enter their name

With such basic code, what could really go wrong?

PROBLEM

What would happen if you were to go to this page?
http://vulnerable.example.com/page.php?name=<script>alert(1);</script>

The HTML would then be: <h1> Hello <script>alert(1);</script>!</h1>

REFLECTED XSS

What you’ve just seen is an example of reflected cross-sites scripting (a.k.a Reflected XSS or rXSS).

In essence, a parameter that an attacker controls is directly reflected back to user. This could allow injection of raw HTML or Javascript (depending on where the XSS takes place) and allow an attacker to perform actions in the context of another user.

This is obviously a contrived example and we’ll cover much more in terms of XSS in the next couple sesions. But think about all the places where your input gets reflected on a website on a daily basis.

How many of those inputs are vulnerable? How many will safely sanitize the data? You’d be surprised.

NEXT SESSION

Next session we’ll be getting deep into how browsers and the web in general work. You’ll also learn about Cross Site Request Forgery, one of the most common and important vulnerabilities.

For now, get your proxy set up and play around with it – take a look at the flow of data when you browse to several sites.