Ethical Student Hackers

Web Hacking

Slides: shefesh.com

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/conduct

- GET requests
- SQL Injections
- Cookies
- Cross Site Scripting

GET Requests

- Parameters can be given when loading page
- A GET request adds these to the end of a URL using ? = & signs
    - ? starts first parameter name
    - = assigns the value
    - & goes before each subsequent parameter
- You can edit these parameters in the URL (activities fair)
- POST sends values but not put in URL
    - Why is this useful?

https://duckduckgo.com/?t=ffab&q=shefesh

GET Examples

- https://www.youtube.com/watch?v=dQw4w9WgXcQ

- https://www.google.co.uk/search?q=ShefESH&sca_esv=568184447&source=hp&ei=BIg…&iflsig=AO6…&ved=0ah…&uact=5&oq=ShefESH&gs_lp=Egd…

SQL

SQL - Structured query language

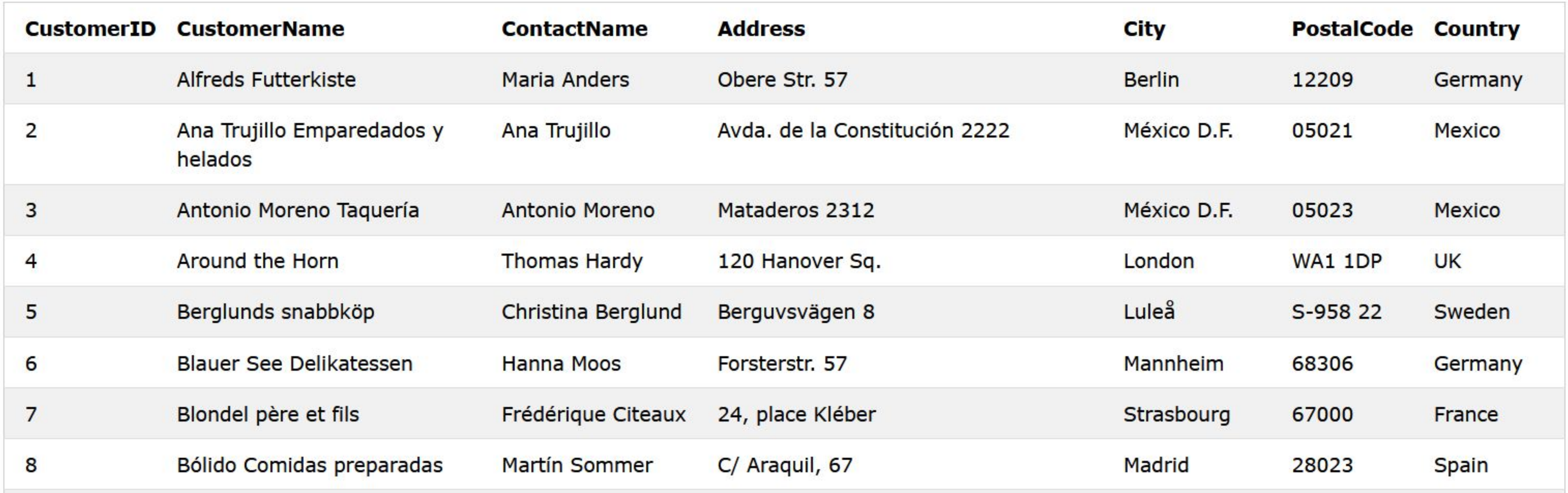
Used to retrieve or modify data in databases

SELECT **INSERT INTO** **DELETE**

UNION **UPDATE**

```
SELECT [fields] FROM [table] (WHERE [condition]);
```

```
SELECT * FROM users WHERE admin = true;
```



CustomerID	CustomerName	ContactName	Address	City	PostalCode	Country
1	Alfreds Futterkiste	Maria Anders	Obere Str. 57	Berlin	12209	Germany
2	Ana Trujillo Emparedados y helados	Ana Trujillo	Avda. de la Constitución 2222	México D.F.	05021	Mexico
3	Antonio Moreno Taquería	Antonio Moreno	Mataderos 2312	México D.F.	05023	Mexico
4	Around the Horn	Thomas Hardy	120 Hanover Sq.	London	WA1 1DP	UK
5	Berglunds snabbköp	Christina Berglund	Berguvsvägen 8	Luleå	S-958 22	Sweden
6	Blauer See Delikatessen	Hanna Moos	Forsterstr. 57	Mannheim	68306	Germany
7	Blondel père et fils	Frédérique Citeaux	24, place Kléber	Strasbourg	67000	France
8	Bólido Comidas preparadas	Martín Sommer	C/ Araquil, 67	Madrid	28023	Spain

```
SELECT * FROM customers WHERE
```

https://www.w3schools.com/sql/

SQLi

SQL Injection - Exploitation of SQL queries with unsanitized user input

In-band SQLi

- Attacker is able to use the same communication channel to both launch the attack and gather results

Inferential SQLi

- attacker is able to reconstruct the database structure by sending payloads, observing the web application's response and the resulting behavior of the database server

Out-of-band SQLi

- an attacker is unable to use the same channel to launch the attack and gather results

SQLi

Bypassing a login form

- A login query may look like this:
  - SELECT * FROM users WHERE username = ‘$username’ AND password = ‘$password’;

SQLi

Data exfiltration

- A search query may look like this:
  - ”SELECT * FROM products WHERE name LIKE ‘%” + user_input + “%’;”

SQLi

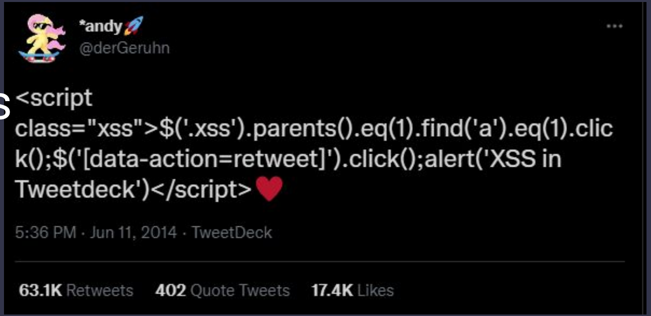
Attack

- To do an SQLi attack, you have to “trick” the server into running your SQL command.
    - ”SELECT * FROM products WHERE username=’’ OR 1=1 -- ‘ and password =’$password’

XSS

Cross Site Scripting (XSS) - Sending of malicious code to websites via unsanitized user input

- DOM - an element in the Document Object Model is changed by a feature on the page - e.g. a button
- Reflected - the payload is delivered in the URL and then rendered on the page - e.g. a search bar
- Stored - the payload is saved to a persistent storage location and later rendered - for example, a commenting system



*andy
@derGeruhn

<script class="xss">$('.xss').parents().eq(1).find('a').eq(1).click();$('[data-action=retweet]').click();alert('XSS in Tweetdeck')</script>♥

5:36 PM · Jun 11, 2014 · TweetDeck

63.1K Retweets 402 Quote Tweets 17.4K Likes

Self retweeting XSS Attack in Tweetdeck

XSS

DOM XSS

Select your language:

```
<select><script>
document.write("<OPTION
value=1>"+decodeURIComponent(do
cument.location.href.substring(docu
ment.location.href.indexOf("default="
)+8))+"</OPTION>");
document.write("<OPTION
value=2>English</OPTION>");
</script></select>
```

Invoked with

http://www.some.site/page.html?default=French

XSS Attack

http://www.some.site/page.html?default=<script>alert(document.cookie)</script>

XSS

Reflected XSS

```
<% String eid =
request.getParameter("eid"); %>
Employee ID: <%= eid %>
```

Display employee id entered into HTTP request

Usually used in phishing

Send via phishing

```
http://www.some.site/page.html?eid=<script>alert(document.cookie)</script>
```

XSS

Stored XSS

$sql = "INSERT INTO MyGuests (firstname, lastname, email) VALUES ($_GET['firstname'], $_GET['lastname'], $_GET['email'])";

Enter guest into database

<?php echo("<p>" . $email . "</p>"); ?>

Invoked with http://www.some.site/add_guest?firstname=John&lastname=Doe&email=test@test.com

XSS Attack http://www.some.site/add_guest?firstname=John&lastname=Doe&email=<script>alert(document.cookie)</script>

XSS

Preventing XSS

DOM based XSS - HTML encoding and JavaScript encode all untrusted input

https://cheatsheetseries.owasp.org/cheatsheets/DOM_based_XSS_Prevention_Cheat_Sheet.html#guideline

Reflected & Stored XSS - Deny all untrusted data where possible, HTML encode, attribute encode, JavaScript encode...Encode as much as possible!

https://cheatsheetseries.owasp.org/cheatsheets/Cross_Site_Scripting_Prevention_Cheat_Sheet.html#xss-prevention-rules-summary

Cookies

Cookies are given to you by the server and store information about you in your browser

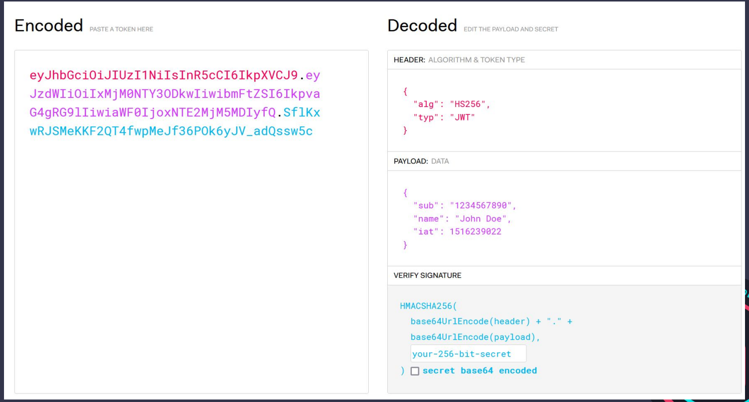
They often represent user sessions and privileges

You can modify cookies to whatever you want, but they are often signed for integrity

Example schemes include JWT tokens

https://jwt.io

You can view your cookies in the F12 > application screen



Encoded PASTE A TOKEN HERE

```
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.ey
JzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6Ikpva
G4gRG9lIiwiaWF0IjoxNTE2MjM5MDIyfQ.SflKx
wRJSMeKKF2QT4fwpMeJf36POk6yJV_adQssw5c
```

Decoded EDIT THE PAYLOAD AND SECRET

HEADER: ALGORITHM & TOKEN TYPE

```
{
  "alg": "HS256",
  "typ": "JWT"
}
```

PAYLOAD: DATA

```
{
  "sub": "1234567890",
  "name": "John Doe",
  "iat": 1516239022
}
```

VERIFY SIGNATURE

```
HMACSHA256(
  base64UrlEncode(header) + "." +
  base64UrlEncode(payload),
  your-256-bit-secret
) ☐ secret base64 encoded
```

Practical

Try out what you have learnt:

http://35.179.134.203:5000

Slides: shefesh.com

Upcoming Sessions

What’s up next?

www.shefesh.com/sessions

1st October: Introduction to Linux

7th October: OSINT/Reconnaissance

Any Questions?

www.shefesh.com

Thanks for coming!