UNIVERSITY OF TWENTE.

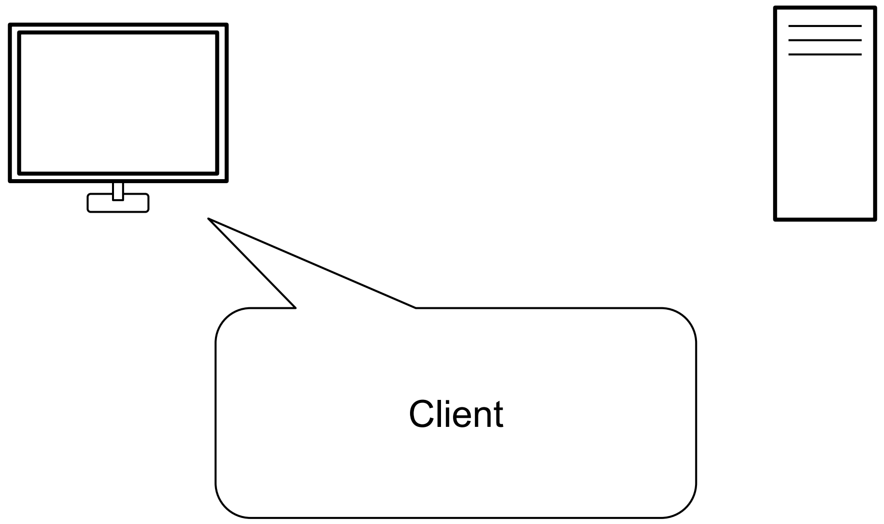
The Basics - Web Hacking

SQL-injection, XSS, path traversal and other techniques used for web hacking

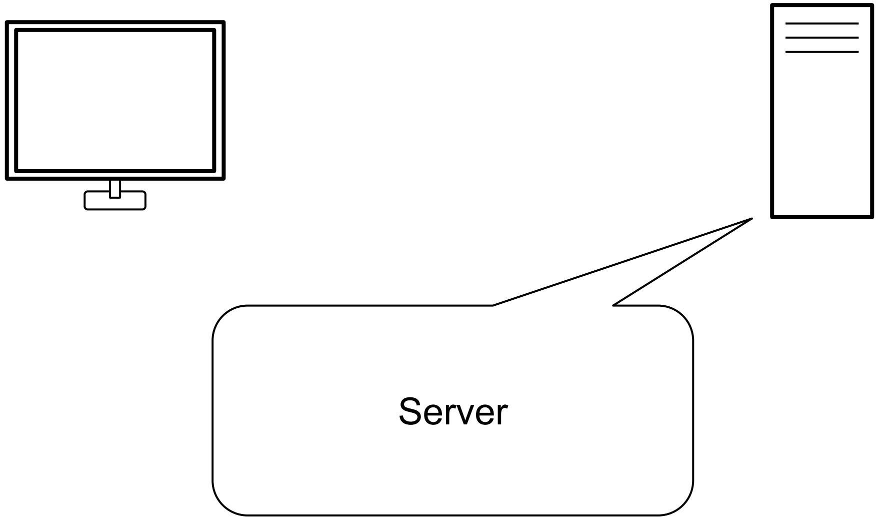
Thijs van Ede (t.s.vanede@utwente.nl)

Web - Basic process

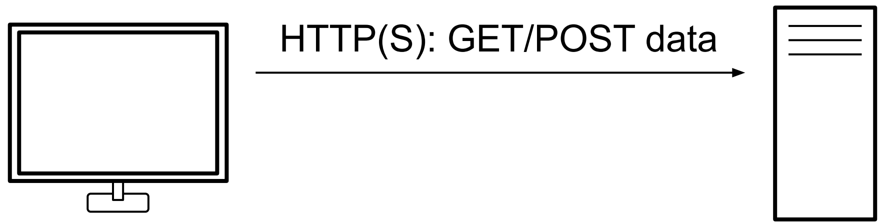
Web - Basic process



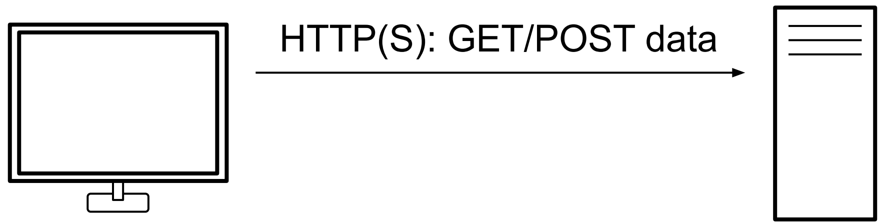
Web - Basic process



Web - Basic process

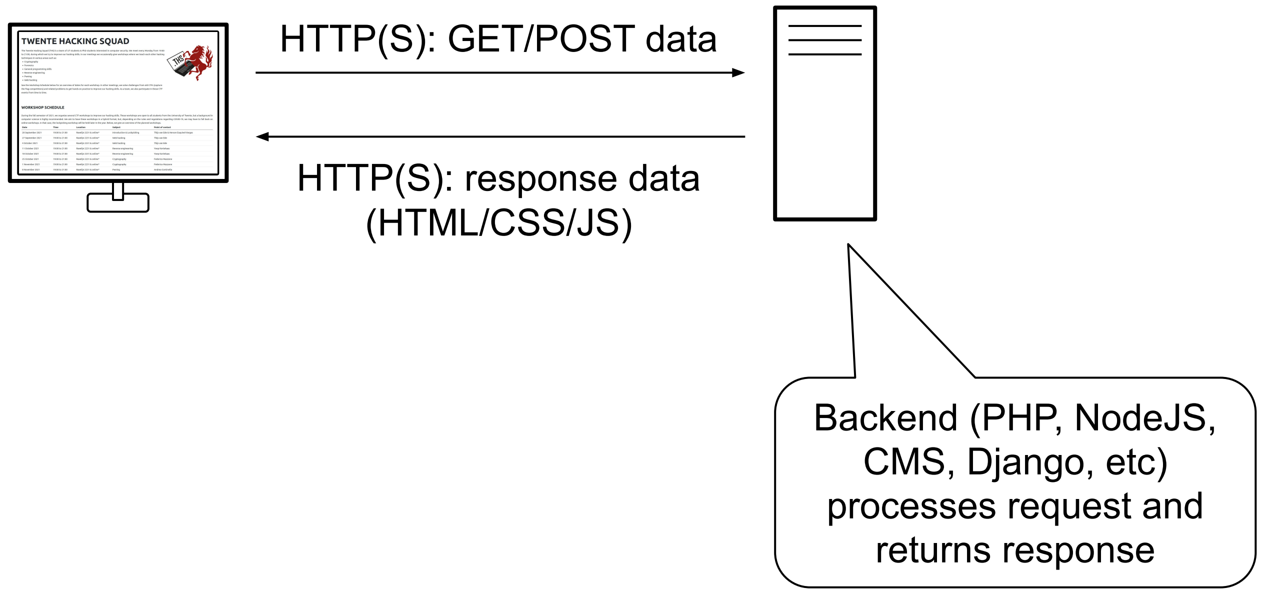


Web - Basic process



Backend (PHP, NodeJS, CMS, Django, etc) processes request and returns response

Web - Basic process


TWENTE HACKING SQUAD
WORKSHOP SCHEDULE
HTTP(S): GET/POST data
HTTP(S): response data (HTML/CSS/JS)
Backend (PHP, NodeJS, CMS, Django, etc) processes request and returns response

Web - Basic process

TWENTE HACKING SQUAD
WORKSHOP SCHEDULE

- Show content to user
- Execute javascript code
- Store local content (e.g., cookies)

Web hacking - How can we exploit this process?

TWENTE HACKING SQUAD

The Twente Hacking Squad (THS) is a team of UT students & PhD students interested in computer security. We meet every Monday from 19:00 to 21:00, during which we try to improve our hacking skills. In our meetings we occasionally give workshops where we teach each other hacking techniques in various areas such as:

- Cryptography
- Forensics
- General programming skills
- Reverse engineering
- Pwning
- Web hacking

See the Workshop schedule below for an overview of dates for each workshop. In other meetings, we solve challenges from old CTFs (capture the flag competitions) and related problems to get hands-on practice to improve our hacking skills. As a team, we also participate in these CTF events from time to time.

WORKSHOP SCHEDULE

During the fall semester of 2021, we organize several CTF workshops to improve our hacking skills. These workshops are open to all students from the University of Twente, but a background in computer science is highly recommended. We aim to have these workshops in a hybrid format, but, depending on the rules and regulations regarding COVID-19, we may have to fall back on online workshops. In that case, the lockpicking workshop will be held later in the year. Below, we give an overview of the planned workshops.

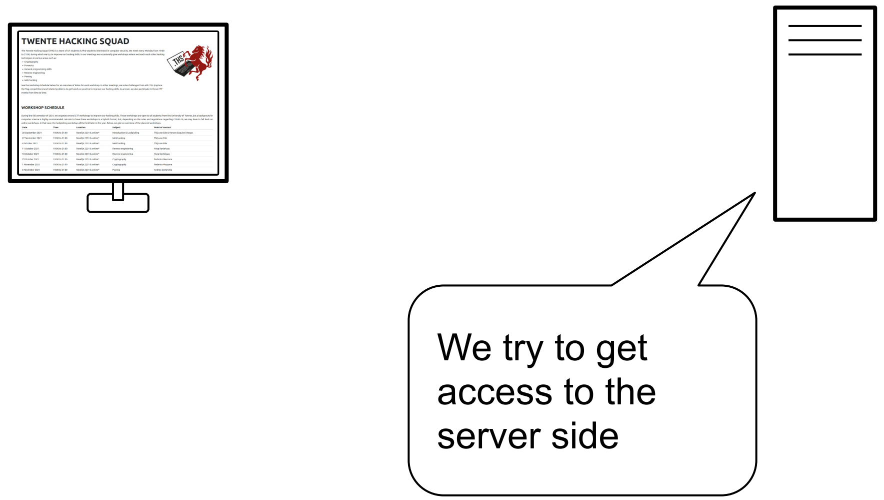
Date	Time	Location	Subject	Point of contact
20 September 2021	19:00 to 21:00	Ravelijn 2231 & online*	Introduction & Lockpicking	Thijs van Ede & Herson Esquivel-Vargas
27 September 2021	19:00 to 21:00	Ravelijn 2231 & online*	Web hacking	Thijs van Ede
4 October 2021	19:00 to 21:00	Ravelijn 2231 & online*	Web hacking	Thijs van Ede
11 October 2021	19:00 to 21:00	Ravelijn 2231 & online*	Reverse engineering	Youp Kortekaas
18 October 2021	19:00 to 21:00	Ravelijn 2231 & online*	Reverse engineering	Youp Kortekaas
25 October 2021	19:00 to 21:00	Ravelijn 2231 & online*	Cryptography	Federico Mazzone
1 November 2021	19:00 to 21:00	Ravelijn 2231 & online*	Cryptography	Federico Mazzone
8 November 2021	19:00 to 21:00	Ravelijn 2231 & online*	Pwning	Andrea Continella

Web hacking - How can we exploit this process?

TWENTE HACKING SQUAD
WORKSHOP SCHEDULE

We have access to everything on the client-side

Web hacking - How can we exploit this process?



Web hacking - How can we exploit this process?

TWENTE HACKING SQUAD
WORKSHOP SCHEDULE

Access local data that should not be sent to the client:

1. Inspect source (ctrl+U)
2. Inspect cookies (F12)
3. Deobfuscate data (https://deobfuscate.io/)

Web hacking - How can we exploit this process?

TWENTE HACKING SQUAD
WORKSHOP SCHEDULE

Bypass/inject client-side processing & communication:

1. Local JavaScript checks
2. Send custom requests & data (Python requests/ scapy)

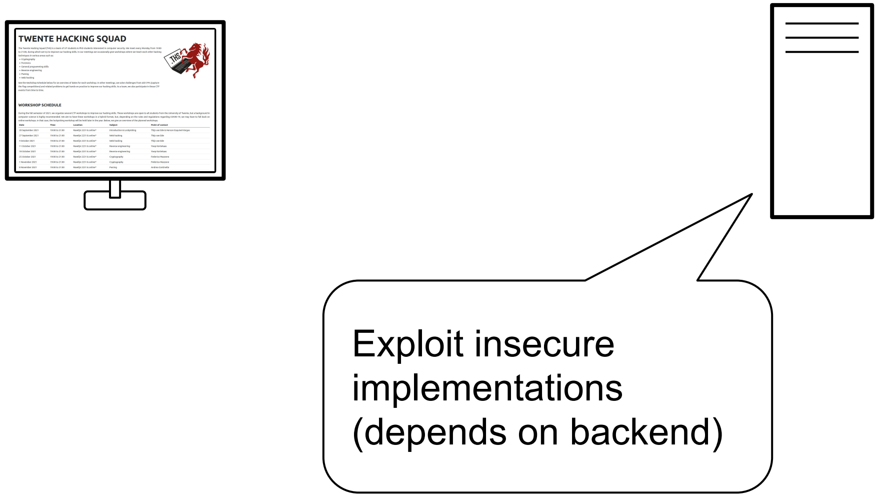
Web hacking - How can we exploit this process?

TWENTE HACKING SQUAD
WORKSHOP SCHEDULE

Try to access resources (especially the ones that should not be accessible):

1. Common files/directories (Gobuster)
2. Guess backend software (Gobuster)
3. Path traversal (Demo)

Web hacking - How can we exploit this process?



Challenges

- Demo
  - https://vm-thijs.ewi.utwente.nl/ctf/traversal.asp?page=index.html

- Challenges
  - overthewire.org - Natas
  - tryhackme.com - OWASP Top 10

- Tools & Libraries:
  - Python requests
  - Python scapy
  - Gobuster

Input sanitization

Unsanitized input

- Web services should treat user input as text, never as code!

Unsanitized input

- Web services should treat user input as text, never as code!
- What happens if you treat it as code?

Unsanitized input

- Web services should treat user input as text, never as code!
- What happens if you treat it as code?
    - Gain access to the website’s database (SQL-injection)

Unsanitized input

- Web services should treat user input as text, never as code!
- What happens if you treat it as code?
    - Gain access to the website’s database (SQL-injection)
    - Run javascript code in other user’s browser (XSS)

Unsanitized input

- Web services should treat user input as text, never as code!
- What happens if you treat it as code?
  - Gain access to the website’s database (SQL-injection)
  - Run javascript code in other user’s browser (XSS)
  - Execute submitted code locally (RCE)

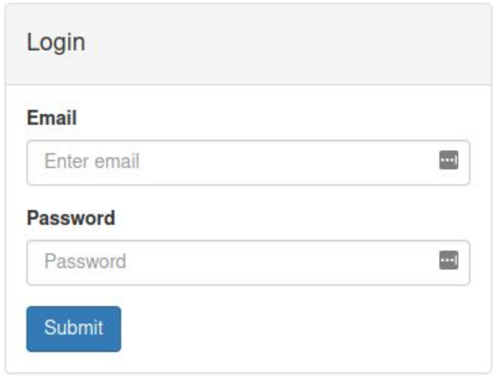
SQL-injection

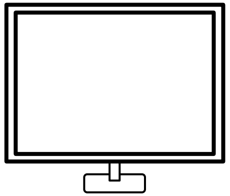
SQL-injection - The Basics

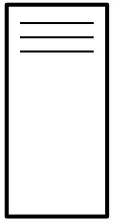
- User data is trusted and gets passed to a database without (proper) checks.

SQL-injection - The Basics

- User data is trusted and gets passed to a database without (proper) checks.
- Example - Website login

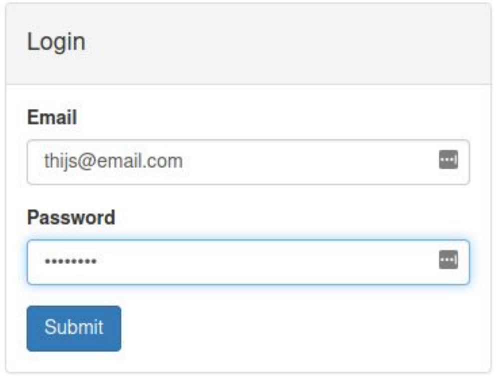






SQL-injection - The Basics

- User data is trusted and gets passed to a database without (proper) checks.
- Example - Website login



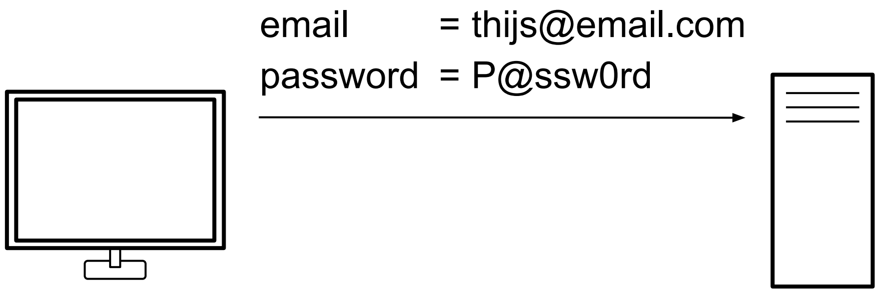
Login

Email

thijs@email.com

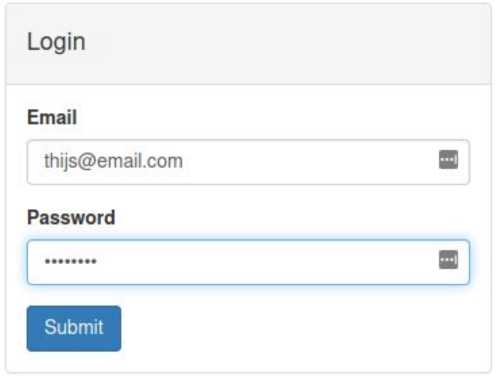
Password

••••••••



SQL-injection - The Basics

- User data is trusted and gets passed to a database without (proper) checks.
- Example - Website login



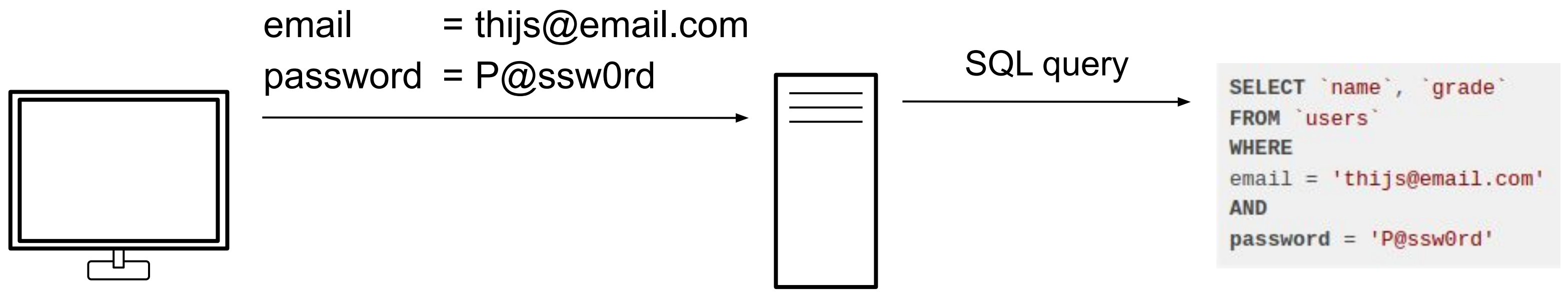
Login

Email

thijs@email.com

Password

•••••••••



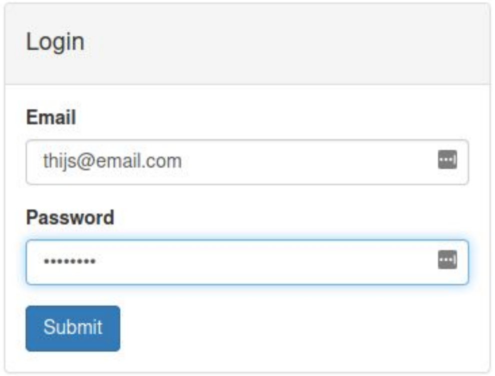
email = thijs@email.com
password = P@ssw0rd

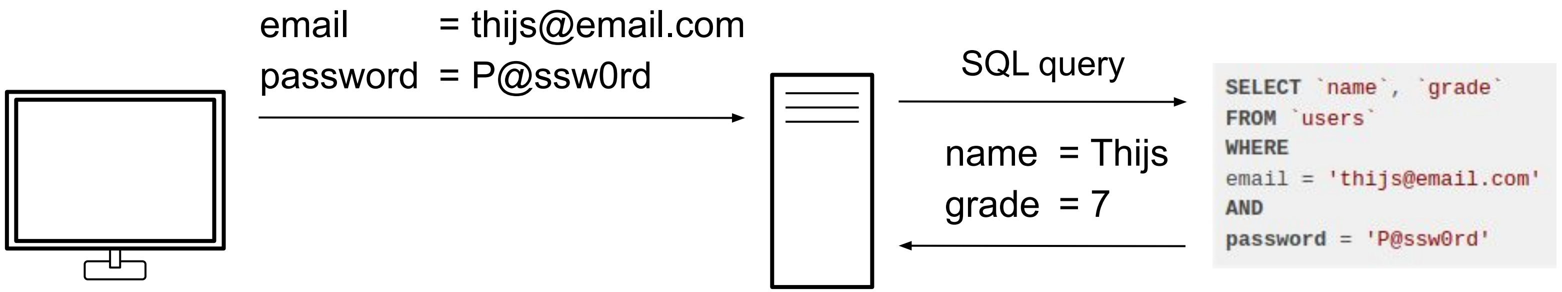
SQL query

```
SELECT `name`, `grade`
FROM `users`
WHERE
email = 'thijs@email.com'
AND
password = 'P@ssw0rd'
```

SQL-injection - The Basics

- User data is trusted and gets passed to a database without (proper) checks.
- Example - Website login

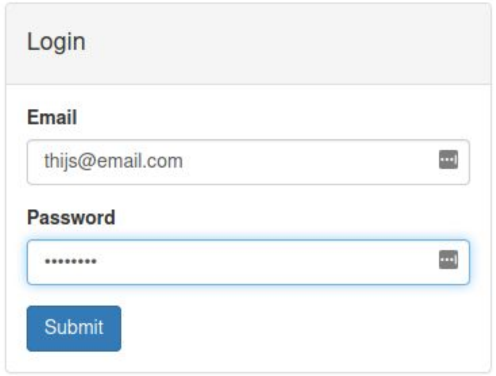


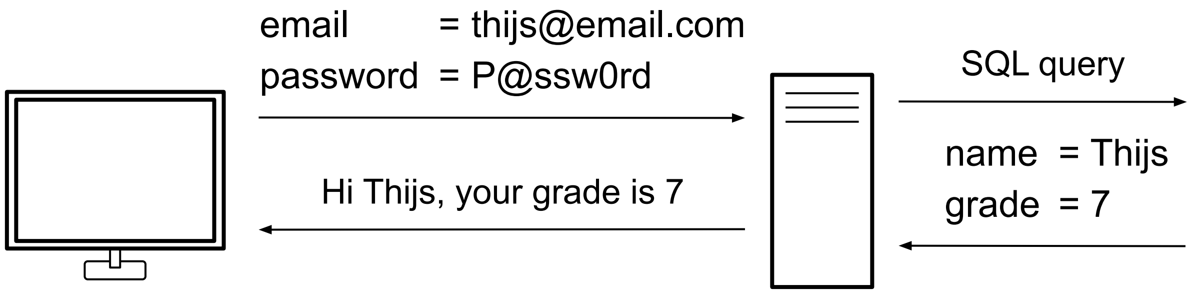


```
SELECT `name`, `grade`
FROM `users`
WHERE
email = 'thijs@email.com'
AND
password = 'P@ssw0rd'
```

SQL-injection - The Basics

- User data is trusted and gets passed to a database without (proper) checks.
- Example - Website login

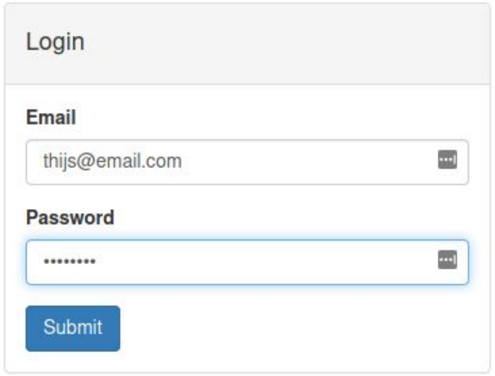


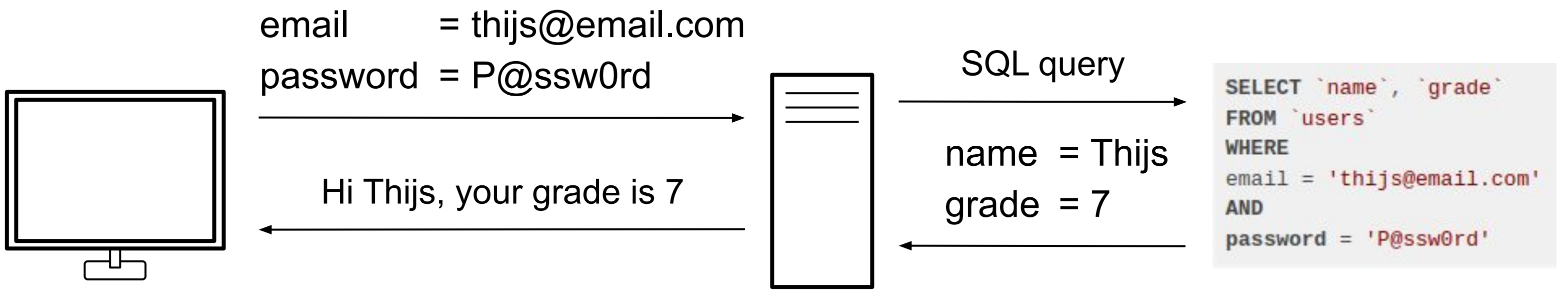


```
SELECT `name`, `grade`
FROM `users`
WHERE
email = 'thijs@email.com'
AND
password = 'P@ssw0rd'
```

SQL-injection - The Basics

- User data is trusted and gets passed to a database without (proper) checks.
- Example - Website login
- Maliciously craft input to gain access

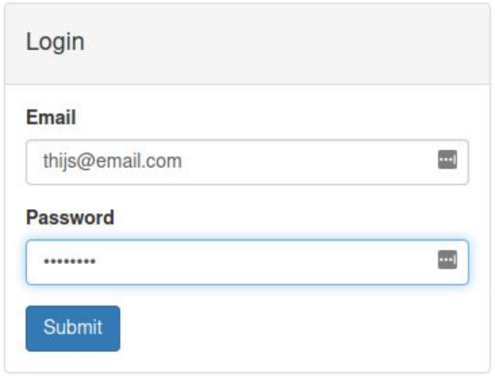




```
SELECT `name`, `grade`
FROM `users`
WHERE
email = 'thijs@email.com'
AND
password = 'P@ssw0rd'
```

SQL-injection - The Basics

- User data is trusted and gets passed to a database without (proper) checks.
- Example - Website login
- Maliciously craft input to gain access


Login

Email

thijs@email.com

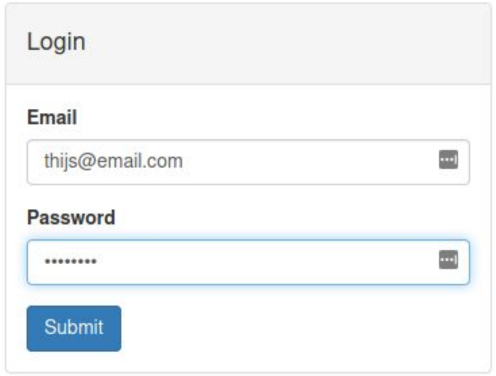
Password

••••••••

```
SELECT `name`, `grade`
FROM `users`
WHERE
email = '<email>'
AND
password = '<password>'
```

SQL-injection - The Basics

- User data is trusted and gets passed to a database without (proper) checks.
- Example - Website login
- Maliciously craft input to gain access


Login

Email

thijs@email.com

Password

••••••••

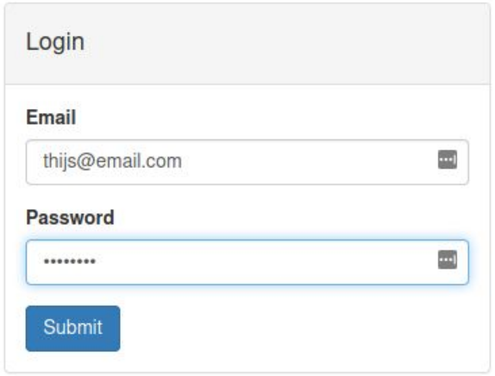
```
SELECT `name`, `grade`
FROM `users`
WHERE
email = '<email>'
AND
password = '<password>'
```

→

$$\underbrace{\texttt{SELECT `name`, `grade` FROM `users` WHERE email = '<email>' AND password = 'incorrect'}}_{\texttt{FALSE}}$$

SQL-injection - The Basics

- User data is trusted and gets passed to a database without (proper) checks.
- Example - Website login
- Maliciously craft input to gain access
  - By adding a ʼ character, we close password string and inject our own code that evaluates to **TRUE**.



```
SELECT `name`, `grade`
FROM `users`
WHERE
email = '<email>'
AND
password = '<password>'
```

$\longrightarrow$

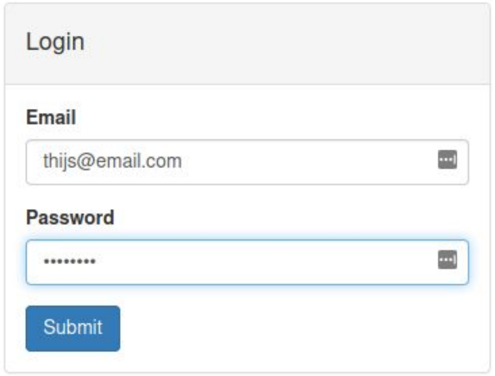
```
SELECT `name`, `grade`
FROM `users`
WHERE
email = '<email>'
AND
password = 'incorrect' OR '1'='1'
```

$\underbrace{\qquad\qquad\qquad\qquad\qquad\qquad}_{\texttt{TRUE}}$

SQL-injection - The Basics

- User data is trusted and gets passed to a database without (proper) checks.
- Example - Website login
- Maliciously craft input to gain access
  - By adding a ʼ character, we close password string and inject our own code that evaluates to **`TRUE`**.
- Input should be sanitized:

`incorrect’ OR ‘1’=’1` ⟶ `incorrect\’ OR \‘1\’=\’1`

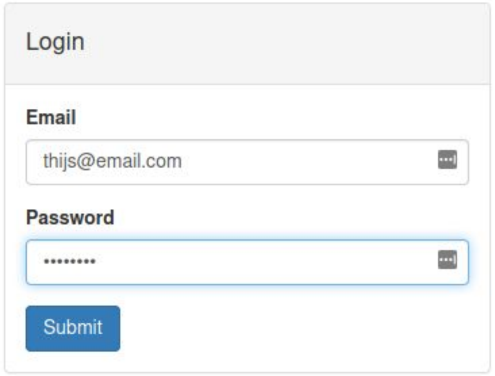


SQL-injection - The Basics

- User data is trusted and gets passed to a database without (proper) checks.
- Example - Website login
- Maliciously craft input to gain access
  - By adding a ′ character, we close password string and inject our own code that evaluates to **`TRUE`**.
- Input should be sanitized:

  `incorrect′ OR ‘1′=′1` $\longrightarrow$ `incorrect\′ OR \‘1\′=\′1`

- Methods differ per language, e.g. for PHP:
  - `mysqli_real_escape_string()`



SQL-injection - Syntax

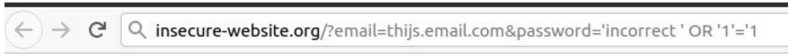
- Syntax must be correct!

SQL-injection - Syntax

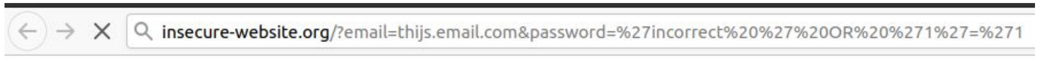
- Syntax must be correct!
- Differs per database, common ones are:
  - MySQL
  - SQLite
  - PostgreSQL
  - MS SQL server
  - Oracle

SQL-injection - Syntax

- Syntax must be correct!
- Differs per database, common ones are:
  - MySQL
  - SQLite
  - PostgreSQL
  - MS SQL server
  - Oracle
- SQL-injection can sometimes be executed through GET parameters



```
insecure-website.org/?email=thijs.email.com&password='incorrect ' OR '1'='1
```



```
insecure-website.org/?email=thijs.email.com&password=%27incorrect%20%27%20OR%20%271%27=%271
```

SQL-injection - UNION attack

- Not limited to the selected `Columns` and **Table**

SQL-injection - UNION attack

- Not limited to the selected `Columns` and **Table**
- **`UNION`** attack

SQL-injection - UNION attack

- Not limited to the selected **`Columns`** and **Table**
- **`UNION`** attack
    - Returns the combination of multiple queries.

SQL-injection - UNION attack

- Not limited to the selected `Columns` and **Table**
- **`UNION`** attack
  - Returns the combination of multiple queries.
  - Used to chain queries and extract data from other **`Columns`** and **Tables**

SQL-injection - UNION attack

- Not limited to the selected `Columns` and **Table**
- **`UNION`** attack
  - Returns the combination of multiple queries.
  - Used to chain queries and extract data from other `Columns` and **Tables**

```
SELECT `name`, `grade`
FROM `users`
WHERE
email = '<email>'
AND
password = 'incorrect' AND '0'='1'
```

SQL-injection - UNION attack

- Not limited to the selected `Columns` and **Table**
- `UNION` attack
  - Returns the combination of multiple queries.
  - Used to chain queries and extract data from other `Columns` and **Tables**

```
SELECT `name`, `grade`
FROM `users`
WHERE
email = ‘<email>’
AND
password = ‘incorrect’ AND ‘0’=’1’
```

Empty

SQL-injection - UNION attack

- Not limited to the selected **`Columns`** and **Table**
- **`UNION`** attack
  - Returns the combination of multiple queries.
  - Used to chain queries and extract data from other **`Columns`** and **Tables**

```
SELECT `name`, `grade`
FROM `users`
WHERE
email = '<email>'
AND
password = 'incorrect' AND '0'='1'

UNION

SELECT * FROM `other_table` WHERE '1'='1'
```

$\}$ **Empty**

SQL-injection - UNION attack

- Not limited to the selected **`Columns`** and **Table**
- **`UNION`** attack
    - Returns the combination of multiple queries.
    - Used to chain queries and extract data from other **`Columns`** and **Tables**

```
SELECT `name`, `grade`
FROM `users`
WHERE
email = '<email>'
AND
password = 'incorrect' AND '0'='1'
```

Empty

```
UNION

SELECT * FROM `other_table` WHERE '1'='1'
```

Returns `other_table`

SQL-injection - UNION attack

- Not limited to the selected **`Columns`** and **Table**
- **`UNION`** attack
  - Returns the combination of multiple queries.
  - Used to chain queries and extract data from other **`Columns`** and **Tables**
- Limitation: each query must return the same number of **`Columns`**

SQL-injection - Finding tables and fields (MySQL)

- We need to know what we are looking for.

SQL-injection - Finding tables and fields (MySQL)

- We need to know what we are looking for.
- Which Tables are in the database?
    - `SELECT table_schema, table_name FROM information_schema.tables`

SQL-injection - Finding tables and fields (MySQL)

- We need to know what we are looking for.
- Which Tables are in the database?
  - `SELECT table_schema, table_name FROM information_schema.tables`
- Which Fields are in a table?

  -
    ```
    SELECT table_name, column_name FROM information_schema.columns
    WHERE table_name = 'table_we_are_looking_for'
    ```

Demo & Challenges

- Demo: https://vm-thijs.ewi.utwente.nl/ctf/sql

- Challenges
  - picoctf.org: Web Gauntlet 1, 2 & 3
    - https://play.picoctf.org/practice/challenge/88?category=1&page=2
    - https://play.picoctf.org/practice/challenge/174?category=1&page=2
    - https://play.picoctf.org/practice/challenge/128?category=1&page=3

- Tools (recommended for actual CTF competitions, not these challenges):
  - BurpSuite (https://portswigger.net/burp)
  - SQLmap (sqlmap.org)

UNIVERSITY
OF TWENTE.

Cross-site scripting (XSS)

XSS - The basics

- Inject a malicious piece of javascript into a webpage

XSS - The basics

- Inject a malicious piece of javascript into a webpage
- Users submit text to a webpage which will then be shown to other users

XSS - The basics

- Inject a malicious piece of javascript into a webpage
- Users submit text to a webpage which will then be shown to other users
    - Social media post

XSS - The basics

- Inject a malicious piece of javascript into a webpage
- Users submit text to a webpage which will then be shown to other users
  - Social media post
  - Forum

XSS - The basics

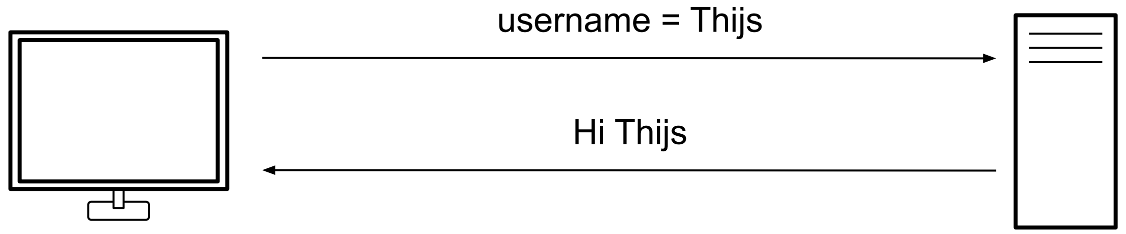
- Inject a malicious piece of javascript into a webpage
- Users submit text to a webpage which will then be shown to other users
  - Social media post
  - Forum
- Submitted text contains HTML formatting including a piece of javascript code

```
<script> // malicious code </script>
```

XSS - The basics

- Inject a malicious piece of javascript into a webpage
- Users submit text to a webpage which will then be shown to other users
    - Social media post
    - Forum
- Submitted text contains HTML formatting including a piece of javascript code

  `<script> // malicious code </script>`
- Example

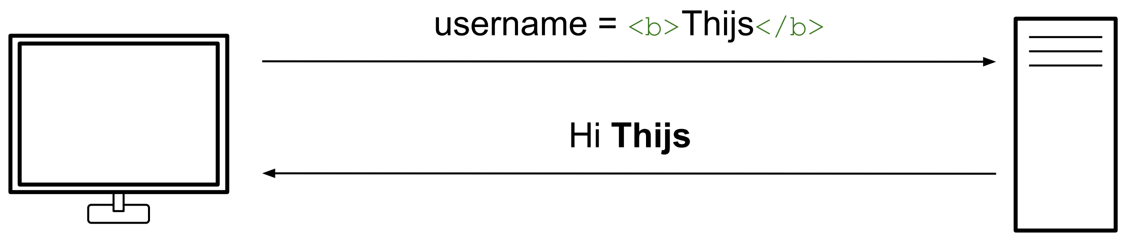


XSS - The basics

- Inject a malicious piece of javascript into a webpage
- Users submit text to a webpage which will then be shown to other users
  - Social media post
  - Forum
- Submitted text contains HTML formatting including a piece of javascript code

```
<script> // malicious code </script>
```

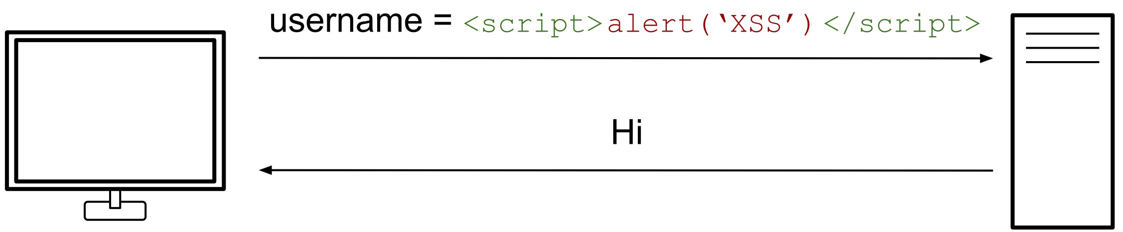
- Example

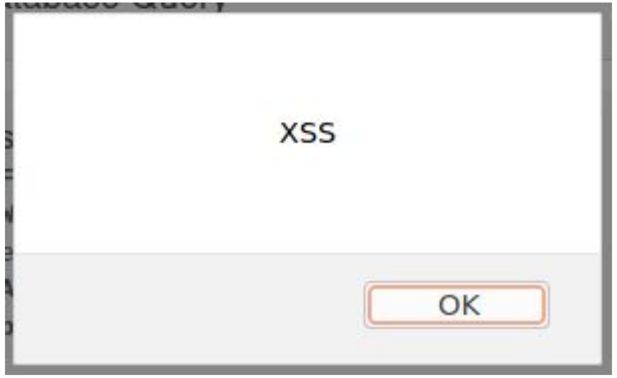


XSS - The basics

- Inject a malicious piece of javascript into a webpage
- Users submit text to a webpage which will then be shown to other users
  - Social media post
  - Forum
- Submitted text contains HTML formatting including a piece of javascript code

  `<script> // malicious code </script>`
- Example





XSS - Challenges

- Find injectable input fields

XSS - Challenges

- Find injectable input fields
    - Check whether <, >, or " characters are allowed

XSS - Challenges

- Find injectable input fields
  - Check whether <, >, or " characters are allowed
  - Check whether <, >, or " are replaced by `&lt;`, `&gt;`, `&quot;`

XSS - Challenges

- Find injectable input fields
  - Check whether <, >, or " characters are allowed
  - Check whether <, >, or " are replaced by `&lt;`, `&gt;`, `&quot;`
- Circumvent inadequate escaping

XSS - Challenges

- Find injectable input fields
  - Check whether <, >, or “ characters are allowed
  - Check whether <, >, or “ are replaced by `&lt;`, `&gt;`, `&quot;`
- Circumvent inadequate escaping
  - Client side escape before submitting

XSS - Challenges

- Find injectable input fields
  - Check whether <, >, or " characters are allowed
  - Check whether <, >, or " are replaced by `&lt;`, `&gt;`, `&quot;`
- Circumvent inadequate escaping
  - Client side escape before submitting
  - Not replacing all necessary characters

XSS - Challenges

- Find injectable input fields
    - Check whether <, >, or " characters are allowed
    - Check whether <, >, or " are replaced by `&lt;`, `&gt;`, `&quot;`
- Circumvent inadequate escaping
    - Client side escape before submitting
    - Not replacing all necessary characters
    - Exploiting user generated code

XSS - Challenges

- Find injectable input fields
  - Check whether <, >, or " characters are allowed
  - Check whether <, >, or " are replaced by `&lt;`, `&gt;`, `&quot;`
- Circumvent inadequate escaping
  - Client side escape before submitting
  - Not replacing all necessary characters
  - Exploiting user generated code
- Maximum input size

XSS - Challenges

- Find injectable input fields
  - Check whether <, >, or " characters are allowed
  - Check whether <, >, or " are replaced by `&lt;`, `&gt;`, `&quot;`
- Circumvent inadequate escaping
  - Client side escape before submitting
  - Not replacing all necessary characters
  - Exploiting user generated code
- Maximum input size
  - Use JQuery

XSS - Attacks

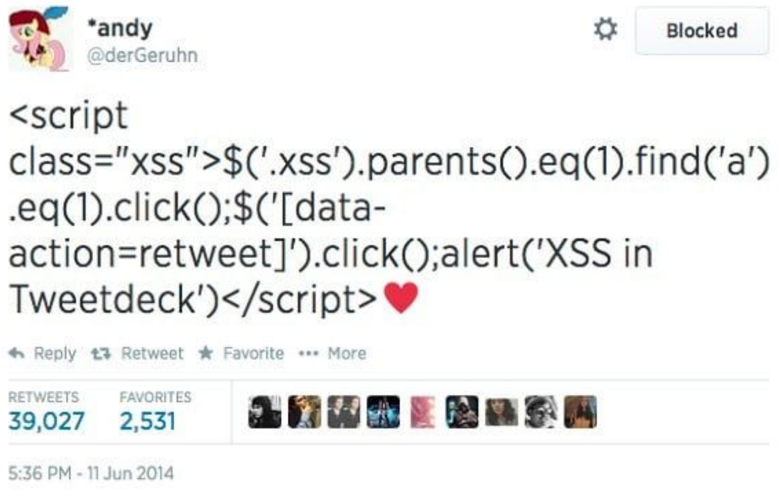
- Annoy users

XSS - Attacks

- Annoy users
    - Constant popups

XSS - Attacks

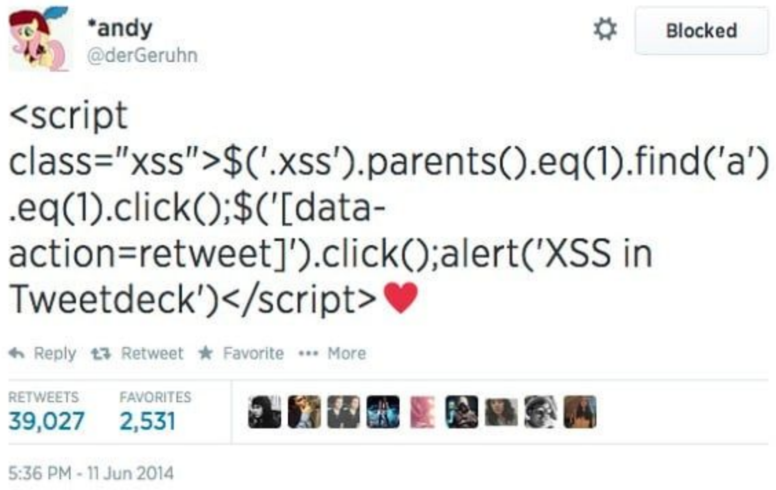
- Annoy users
  - Constant popups
  - Self retweeting tweet


*andy
@derGeruhn

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

XSS - Attacks

- Annoy users
    - Constant popups
    - Self retweeting tweet
- Steal credentials


*andy
@derGeruhn
Blocked

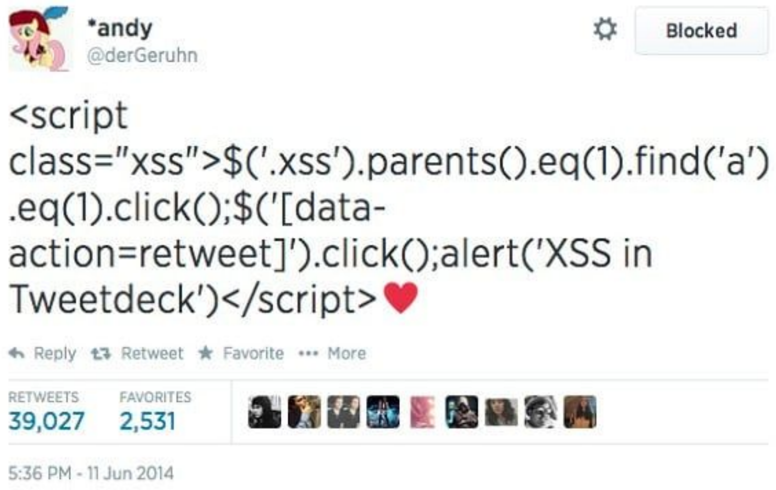
```
<script
class="xss">$('.xss').parents().eq(1).find('a')
.eq(1).click();$('[data-
action=retweet]').click();alert('XSS in
Tweetdeck')</script>♥
```

RETWEETS 39,027 FAVORITES 2,531

5:36 PM - 11 Jun 2014

XSS - Attacks

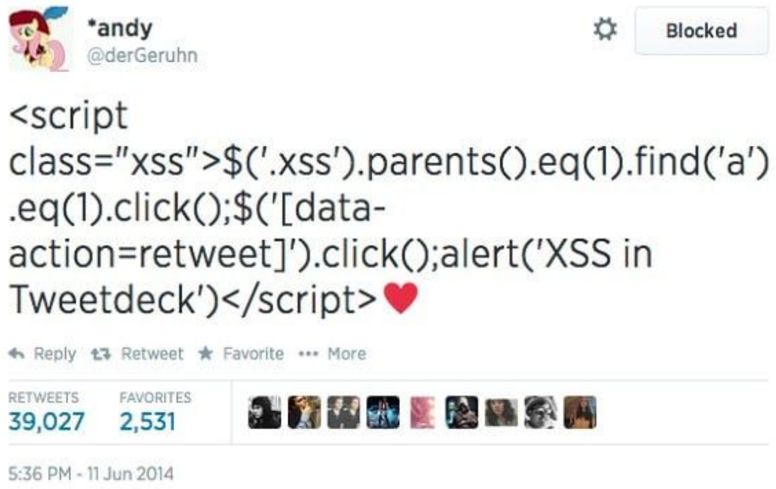
- Annoy users
  - Constant popups
  - Self retweeting tweet
- Steal credentials
  - Listen on input fields



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

XSS - Attacks

- Annoy users
  - Constant popups
  - Self retweeting tweet
- Steal credentials
  - Listen on input fields
  - Steal cookies



*andy
@derGeruhn

Blocked

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

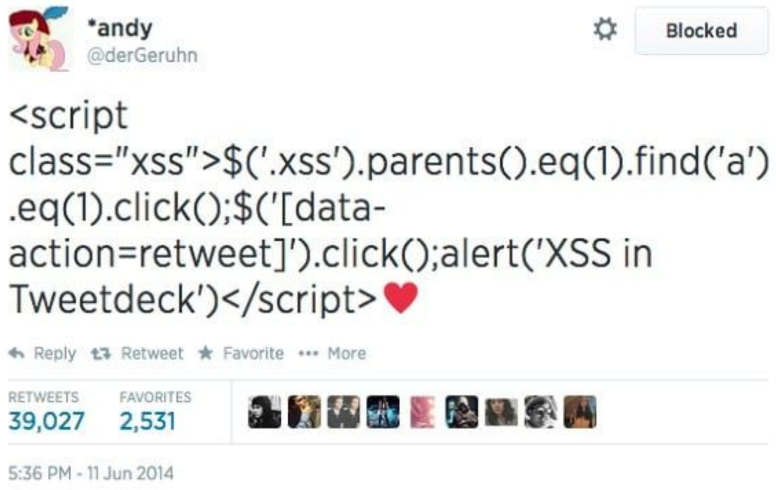
Reply Retweet Favorite More

RETWEETS 39,027 FAVORITES 2,531

5:36 PM - 11 Jun 2014

XSS - Attacks

- Annoy users
  - Constant popups
  - Self retweeting tweet
- Steal credentials
  - Listen on input fields
  - Steal cookies
  - Steal session tokens



*andy
@derGeruhn

Blocked

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

Reply Retweet Favorite More

RETWEETS	FAVORITES
39,027	2,531

5:36 PM - 11 Jun 2014

Challenges

https://alf.nu/alert1

Remote Code Execution

Remote Code Execution

- Depends on the server backend, e.g.,

Remote Code Execution

- Depends on the server backend, e.g.,
  - Prototype pollution (NodeJS/JavaScript)

Remote Code Execution

- Depends on the server backend, e.g.,
  - Prototype pollution (NodeJS/JavaScript)
  - Local File Inclusion (Various)

Remote Code Execution

- Depends on the server backend, e.g.,
  - Prototype pollution (NodeJS/JavaScript)
  - Local File Inclusion (Various)
  - Reverse shell (Various)

Remote Code Execution

- Depends on the server backend, e.g.,
  - Prototype pollution (NodeJS/JavaScript)
  - Local File Inclusion (Various)
  - Reverse shell (Various)
- Uploaded file/code is processed insecurely, e.g.,

Remote Code Execution

- Depends on the server backend, e.g.,
    - Prototype pollution (NodeJS/JavaScript)
    - Local File Inclusion (Various)
    - Reverse shell (Various)
- Uploaded file/code is processed insecurely, e.g.,
    - PHP include

Remote Code Execution

- Depends on the server backend, e.g.,
  - Prototype pollution (NodeJS/JavaScript)
  - Local File Inclusion (Various)
  - Reverse shell (Various)
- Uploaded file/code is processed insecurely, e.g.,
  - PHP include
  - PHP passthru

Remote Code Execution

- Depends on the server backend, e.g.,
  - Prototype pollution (NodeJS/JavaScript)
  - Local File Inclusion (Various)
  - Reverse shell (Various)
- Uploaded file/code is processed insecurely, e.g.,
  - PHP include
  - PHP passthru
- Used functions are vulnerable

Remote Code Execution

- Depends on the server backend, e.g.,
  - Prototype pollution (NodeJS/JavaScript)
  - Local File Inclusion (Various)
  - Reverse shell (Various)
- Uploaded file/code is processed insecurely, e.g.,
  - PHP include
  - PHP passthru
- Used functions are vulnerable
- Google terms to use: RCE <backend function/language>

Demo & Exercises

- Resources
  - https://github.com/swisskyrepo/PayloadsAllTheThings
  - https://www.revshells.com/
- Challenges
  - hackthebox.eu - Web - Gunship
- Tools:
  - netcat (nc) (https://portswigger.net/burp)

UNIVERSITY
OF TWENTE.