Web Security

Ethical Hacking

Alessandro Brighente
Eleonora Losiouk

Master Degree on Cybersecurity

UNIVERSITÀ
DEGLI STUDI
DI PADOVA

SPRITZ
SECURITY & PRIVACY
RESEARCH GROUP

Introduction to Web Security

SPRITZ
SECURITY & PRIVACY
RESEARCH GROUP

UNIVERSITÀ
DEGLI STUDI
DI PADOVA

- Websites and web applications, as networks and hardware devices, may be vulnerable to different cyberattacks
- Web security refers to the security exploitation and defense measures over websites and web applications
- An attacker needs to first recognize the components of the application and how it expects to interact with the user

Cross-Site Scripting

- Cross-Site Scripting (XSS) is the most common vulnerability in the internet
- Comes out of increased interaction with the user
- Take advantage of the fact that web applications execute scripts on the users' browser
- Dynamically created scripts pose risks on web applications, especially if they can be contaminated or modified

XSS Types

- Three top categories
  - Stored: the code is stored on a database before execution
  - Reflected: the code is reflected by a server
  - DOM-Based: code both stored and executed in the browser

- There exist also other types, but these have been identified as those from which common web applications need to watch out for on a daily basis

Discovery and Exploitation

- Imagine that you are not happy with a service provided by a company, and that you have a space on their website to report feedbacks
- You want to emphasize how discontent you are, but the writing space does not provide the option for bold text
- So you do something like

```
I am <strong>very</strong> unhappy with the service
```

Discovery and Exploitation

- The costumer care employee reads your message and sees the bold word
- This is the chain of actions:
  - User submits the comment via web form
  - The comment is stored in the database
  - The comment is requested via HTTP request from the employee
  - The comment is injected in the web page
  - The comment is interpreted as DOM rather than text
- Due to an architectural mistake we have something potentially very harmful for a company

Discovery and Exploitation

SPRITZ
Security & Privacy
Research Group

Università
degli Studi
di Padova

- Since the text is appended to the DOM it is interpreted as DOM markup
- Remember we included the strong tag
- If tag is executed, then the attacker might encode some more malicious actions to take advantage of this vulnerability
- Script tags are the most popular way to take advantage of XSS vulnerabilities
- What if code is embedded in the text?

Discovery and Exploitation

- I am very unhappy with the service

<script> …

```
const customerData = [];
customers.forEach((customer) => {
customerData.push({
firstName: customer.querySelector('.firstName').innerText,
lastName: customer.querySelector('.lastName').innerText,
email: customer.querySelector('.email').innerText,
phone: customer.querySelector('.phone').innerText
});
}); … </script>
```

Discovery and Exploitation

- In this case, we are delivering a stored XSS attack
- The code for the attack is stored in the application owner's database
- When the script tag hits the DOM it starts its execution as it is interpreted through the script tags, not as text
- The code runs without requiring interaction from the user on the other side
- The code traverses the DOM and steals privileged data belonging to the company
- There would also be a part for conversion in a JSON format

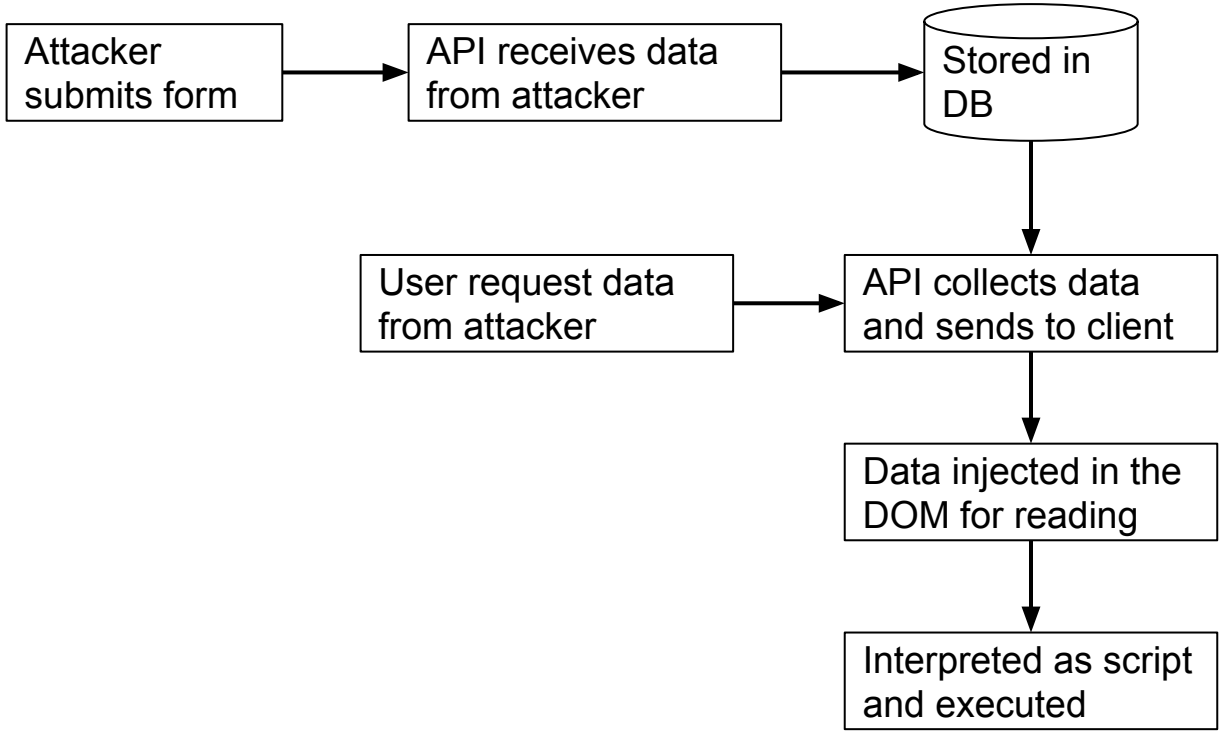
Discovery and Exploitation

SPRITZ
SECURITY & PRIVACY
RESEARCH GROUP

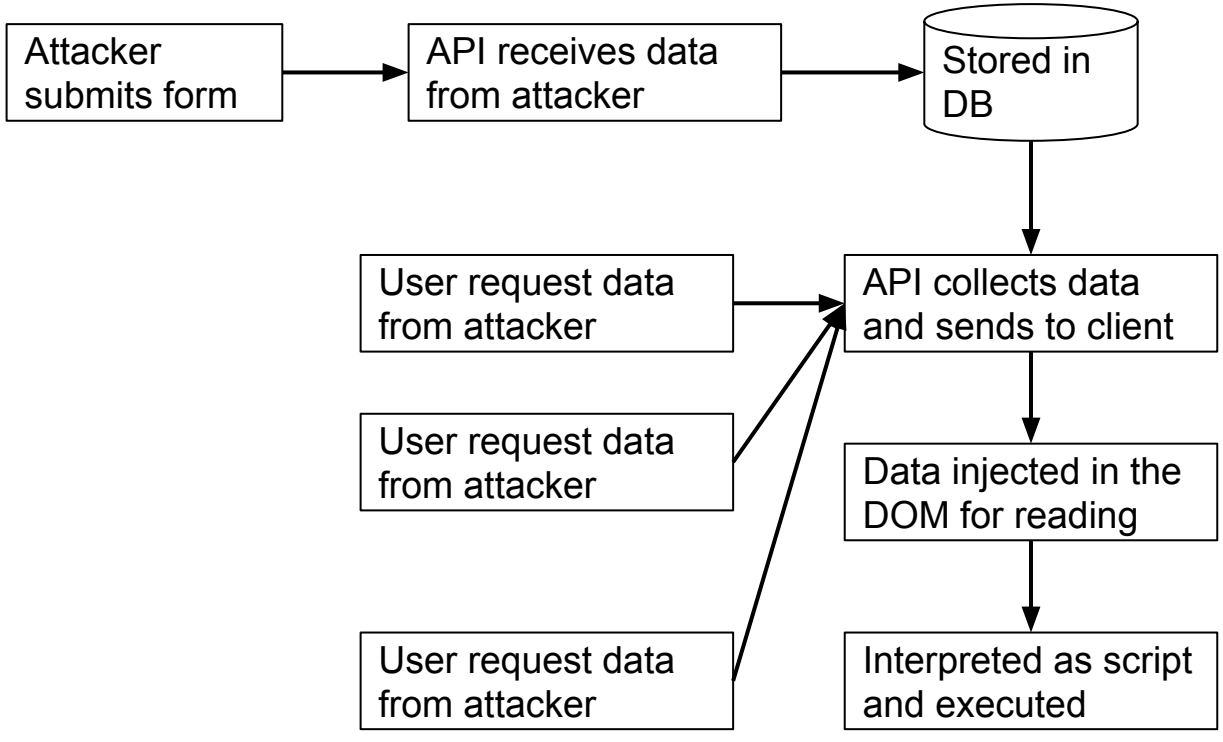
UNIVERSITÀ
DEGLI STUDI
DI PADOVA

- Since the code is inside a script tag, it will not be spotted by the employee of the costumer service
- He will only see the text part, whereas that in the script tag will just be executed
- The text is interpreted as text, and the part in the tag as if a legitimate developer wrote it
- Since the code is in the database, every time someone downloads the comment they will also run the script

Stored XSS Attacks

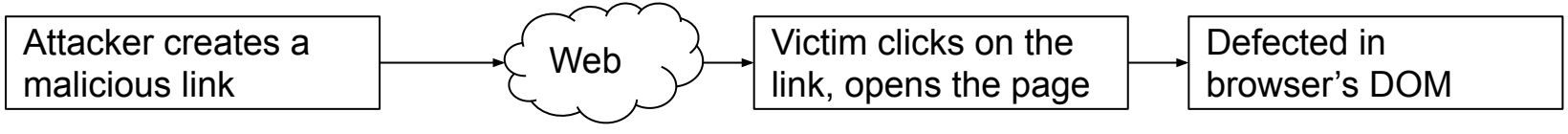


Stored XSS Attacks



Reflected XSS

- They work as stored XSS, but are not stored on a database nor should they regularly hit a server
- A reflected XSS does not to be relayed, it affects the code of the client directly in the browser



Example of Reflected XSS

- We want to look up for documentation to open a new bank account
- The service provider website has a search bar we can use
- Our search redirects us to a URL:

  support.mega-bank.com/search?query=open+savings+account
- We see the corresponding heading
- We try to modify the URL in

  support.mega-bank.com/search?query=open+checking+account
- We see the corresponding heading

Example of Reflected XSS

- We know that the heading and URL are correlated
- We then try to add a strong tag in the URL
- If we see the bold word on the header, bingo!
- Let's include a script tag in the heading
- We have found an XSS vulnerability, but this will not be stored in a database
- The server reads it and send it back to the client

Reflected XSS

- Since they are not stored in a database they are more difficult to detect
- They often target a user directly
- The example we showed can exploit different channels: advertisement, emails with URLs,...
- Also in this case we can discover different types of information at the victim's side
- It is however harder to distribute

DOM Based XSS

- Takes advantages of vulnerabilities in the DOM
- Can be either reflected or stored
- Some browsers might be vulnerable to certain attacks, some others are not due to changes in the DOM implementations
- Much more difficult to find, as they take advantage of specific vulnerabilities in implementations
- They never require interaction with the server

DOM Based XSS

- They require a source and a sink
  - Source = DOM object capable of storing text
  - Sink = DOM API capable of executing scripts stored as text
- Since they never interact with a server, no static analysis tool can find them
- The difficulty relies in the task of finding bugs in a specific DOM
- Different version of a web browser may or may not be vulnerable to the same DOM XSS

DOM Based XSS Example

- Navigating a website we notice that we can navigate the content and apply some filtering policy
- Since the resources at the portal side are limited, searching and sorting take place at the client side
- Searching “oil” produces a URL:

```
investors.mega-bank.com/listing?search=oil
```

- And filtering for US produces the URL:

```
investors.mega-bank.com/listing#usa
```

DOM Based XSS Example

SPRITZ
SECURITY & PRIVACY
RESEARCH GROUP

UNIVERSITÀ
DEGLI STUDI
DI PADOVA

- Changes in URL not necessarily imply that an interaction with the server is happening
- Modern webApps have their own JavaScript-based routers
- Therefore, query parameters may cause DOM XSS on the local machine
- Malicious code in sources does not cause any trouble unless there is another piece of code that makes use of it

DOM Based XSS Example

- Grab hash from URL and find matches

```
const hash = document.location.hash;

const funds = [];

const nMatches = findNumberOfMatches(funds, hash);
```

- Write number of matches and append hash to the DOM

```
document.write(nMatches + 'matches found for' + hash);
```

DOM Based XSS Example

- Grab hash from URL and find matches

source

```
const hash = document.location.hash;

const funds = [];

const nMatches = findNumberOfMatches(funds, hash);
```

- Write number of matches and append hash to the DOM

```
document.write(nMatches + 'matches found for' + hash);
```

sink

DOM Based XSS Example

SPRITZ
SECURITY & PRIVACY
RESEARCH GROUP

UNIVERSITÀ
DEGLI STUDI
DI PADOVA

- The attacker generates the following lik

```
investors.mega-bank.com/listing#<script>alert(document.cookie);</script>
```

- The sink document.write will execute the hash value as a script, displaying the current session cookies
- Based on what we have seen we can also do something more harmful
- No interaction with the server, and legitimate strings to not cause problems
- May go undetected for a long time