Web Security: Part II

Emilio Coppa
coppa@diag.uniroma1.it

Sapienza University of Rome

Credits

These slides are based on teaching material originally created by:

- Marco Squarcina (marco.squarcina@tuwien.ac.at), S&P Group, TU WIEN
- Mauro Tempesta (mauro.tempesta@tuwien.ac.at), S&P Group, TU WIEN
- Fabrizio D'Amore (damore@diag.uniroma1.it), Sapienza University of Rome

OS vs. Browser Analogies

Operating System

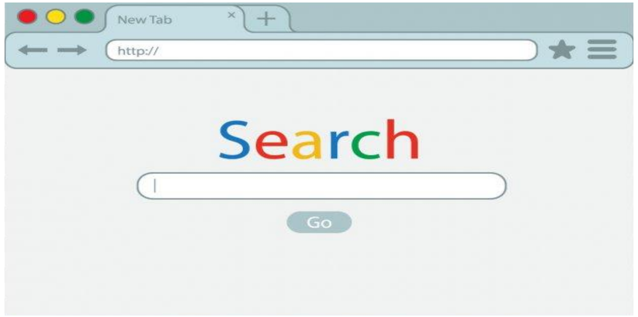
- **Primitives**
  - System calls
  - Processes
  - Disk
- **Principals: Users**
  - Discretionary access control
- **Vulnerabilities**
  - Buffer overflows
  - ...

Web Browser

- **Primitives**
  - DOM, Web APIs
  - Frames
  - Cookies and local storage
- **Principals: Origins**
  - Mandatory access control
- **Vulnerabilities**
  - Cross-site scripting (XSS)
  - ...

Javascript and Same Origin Policy (SOP)

Browser: Basic Execution Model



- Each browser window/tab/frame:
  - **Loads content**
  - **Renders pages**
    - Processes HTML, stylesheets and scripts to display the page
    - May involve fetching additional resources / pages like images, frames, etc.
  - **Reacts to events (via JavaScript)**
    - User actions: OnClick, OnMouseover, …
    - Rendering: OnLoad, OnUnload, …
    - Timing: setTimeout, clearTimeout, …

JavaScript in Web Pages

- Scripts can be embedded in a page in multiple ways:
  - Inlined in the page:

    ```
    <script>alert("Hello World!");</script>
    ```

  - Stored in external files:

    ```
    <script type="text/javascript" src="foo.js"></script>
    ```

  - Specified as event handlers:

    ```
    <a href="http://www.bar.com" onmouseover="alert('hi');">
    ```

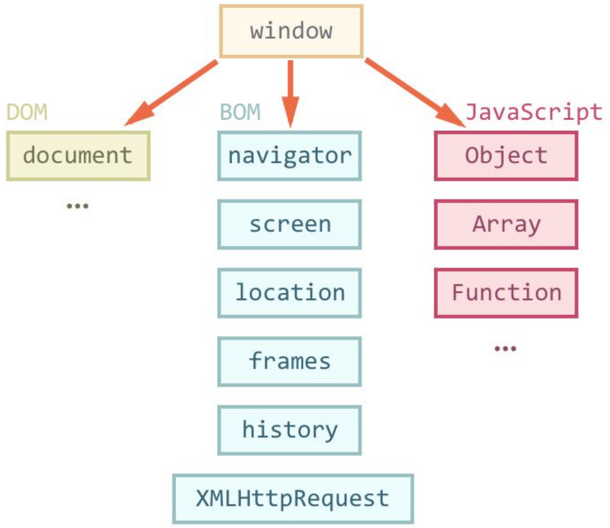
  - Pseudo-URLs in links:

    ```
    <a href="javascript:alert('You clicked');">Click me</a>
    ```

DOM and BOM [recap]

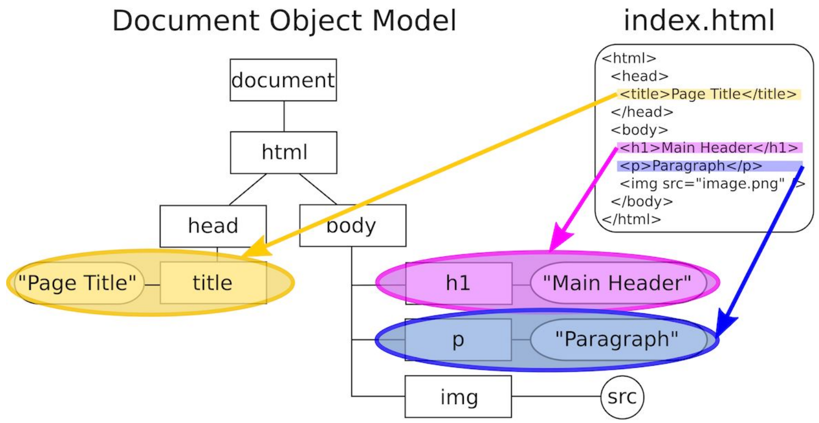
JavaScript can interact with the HTML page and the browser through the DOM and the BOM.

- **Browser Object Model (BOM)**
  - Browser-specific Web APIs
  - Elements are:
    Window, Frames, History, Location, Navigator (browser type & version), ….
  - For example for Firefox: [API]



DOM and BOM [recap]

- **Document Object Model (DOM)**
  - Living Standard by WHATWG/W3C https://dom.spec.whatwg.org
  - Object-oriented representation of the page structure
  - Properties: document.forms, document.links, ...
  - Methods: document.createElement, document.getElementsByTagName, ...
  - By interacting with the DOM, scripts can read and modify the contents of the webpage



```
<html>
  <head>
    <title>Page Title</title>
  </head>
  <body>
    <h1>Main Header</h1>
    <p>Paragraph</p>
    <img src="image.png" />
  </body>
</html>
```

Reading Properties with JavaScript [recap]

- Consider the following snippet of HTML code:

```
<UL id="t1">
  <LI>Item 1</LI>
</UL>
```

- JavaScript provides many methods to access the various properties of the corresponding DOM tree:

```
document.getElementById('t1').nodeName
 // -> returns 'UL'
document.getElementById('t1').getAttribute('id')
 // -> returns 't1'
document.getElementById('t1').innerHTML
 // -> returns '<li>Item 1</li>'
document.getElementById('t1').children[0].nodeName
 // -> returns 'LI'
document.getElementById('t1').children[0].innerText
 // -> returns 'Item 1'
```

Page Manipulation with JavaScript [recap]

- JavaScript can dynamically modify the DOM, e.g., to add a new item to the list:

```
let list = document.getElementById('t1');
let item = document.createElement('LI');
item.innerText = 'Item 2';
list.appendChild(item);
```

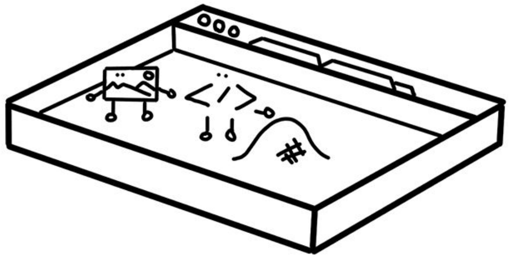
- Or to add an event handler to the items in the list:

```
let list = document.getElementById('t1');
list.addEventListener('click', (event) => {
  alert(`Clicked: ${event.target.innerText}`);
});
```

Browser Sandbox

Goal: **safely execute JavaScript** code provided by a remote website by enforcing isolation from resources provided by other websites

- No direct file access
- Limited access to
  - OS
  - network
  - browser data
  - content that came from other websites



Same Origin Policy (SOP)

- SOP is the **baseline security policy** implemented by web browsers
- An **origin** is defined as the triplet **(protocol, domain, port)**
- **Scripts** running on a page hosted at a certain origin can **access only** resources from the **same origin**:
  - access (read / write) to DOM of other frames
  - access (read / write) to the cookie jar (different concept of origin, we will see it later) and local/session storage
  - access (read) to the body of a network response
- Some aspects are **not** subject to SOP:
  - inclusion of resources (images, scripts, ...) → **See later CSP**
  - form submission
  - sending requests (e.g., via the fetch API)

  } → **See later CSRF**

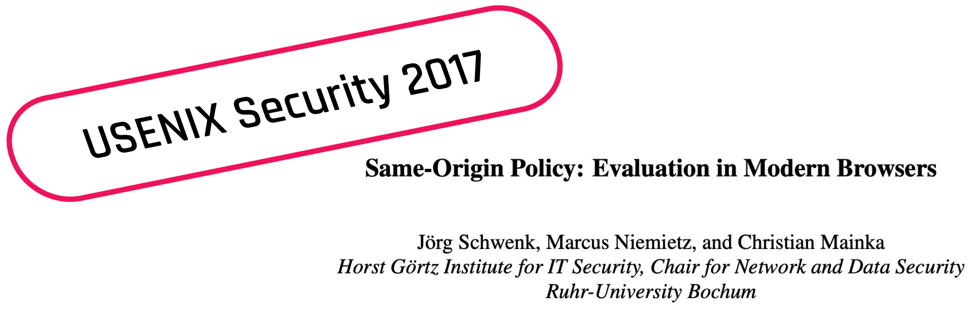
Reference: https://developer.mozilla.org/en-US/docs/Web/Security/Same-origin_policy

Examples

Sample URL: https://example.com/index.htm

URL	Same origin?	Reason
https://example.com/profile.htm	Yes	Only the path differs
http://example.com/index.htm	No	Different protocol
https://shop.example.com/index.html	No	Different hostname
https://example.com:456/index.htm	No	Different port (default HTTPS port is 443)

SOP is hard!


USENIX Security 2017

Same-Origin Policy: Evaluation in Modern Browsers

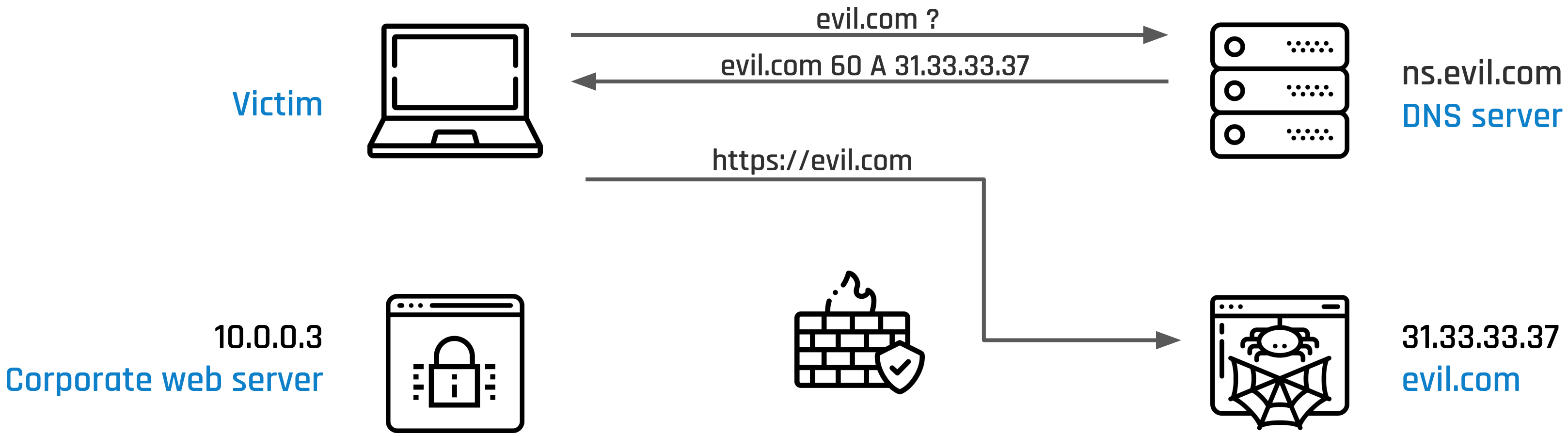
Jörg Schwenk, Marcus Niemietz, and Christian Mainka
Horst Görtz Institute for IT Security, Chair for Network and Data Security
Ruhr-University Bochum

- Despite being a fundamental web security mechanism, **there is no formal definition of SOP**!
- Full policy of current browsers is complex
  - Evolved via "penetrate-and-patch"
  - Different features evolved in **slightly different policies**
  - A recent study evaluated 10 different browsers and discovered **that browsers behave differently** in 23% of the tests (focus only on DOM access)

See also: Site Isolation: Process Separation for Web Sites within the Browser

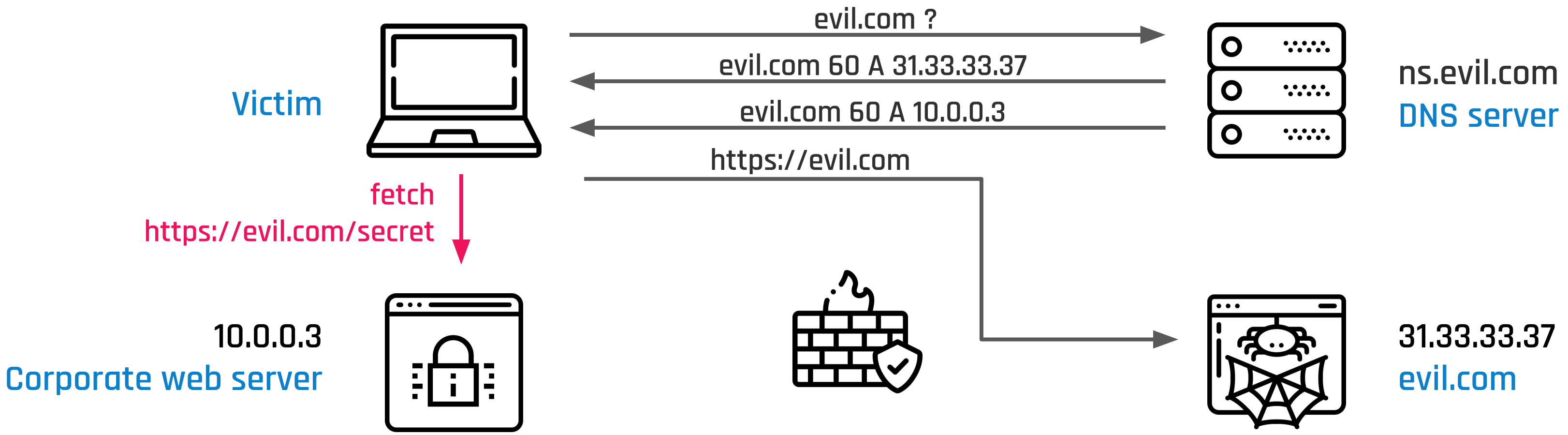
DNS Rebinding

- 12 years-old attack that sidesteps the SOP by abusing DNS
- Can be used to breach a private network by causing the victim's browser to access computers at private IP addresses and leak the results to unauthorized parties



DNS Rebinding (2)

- 12 years-old attack that sidesteps the SOP by abusing DNS
- Can be used to breach a private network by causing the victim's browser to access computers at private IP addresses and leak the results to unauthorized parties



Mitigations against DNS Rebinding

- **DNS Pinning**
  - Browsers could lock the IP address to the value received in the first DNS response
  - Compatibility issue with some dynamic DNS uses, load balancing, etc.

- Web servers can **reject** HTTP requests with an **unrecognized Host headers**
  - **Default catchall virtual hosts** in the web server configuration should be avoided