Detection and Prevention of Changes in the DOM Tree

Junaid Iqbal, Ratinder Kaur, Natalia Stakhanova

Canadian Institute for Cybersecurity (CIC), University of New Brunswick (UNB)

Problem Statement

DOM-based Cross-Site Scripting (XSS) is a type of XSS attack wherein the attack payload is executed as a result of modifying the DOM environment in the victim's browser used by the original client-side script, so that the client-side code runs in an unexpected manner. DOM-based XSS vulnerabilities are very difficult to detect because the target of the malicious payload is the browser, as compared to other XSS vulnerabilities where target is server.

Document Object Model (DOM)

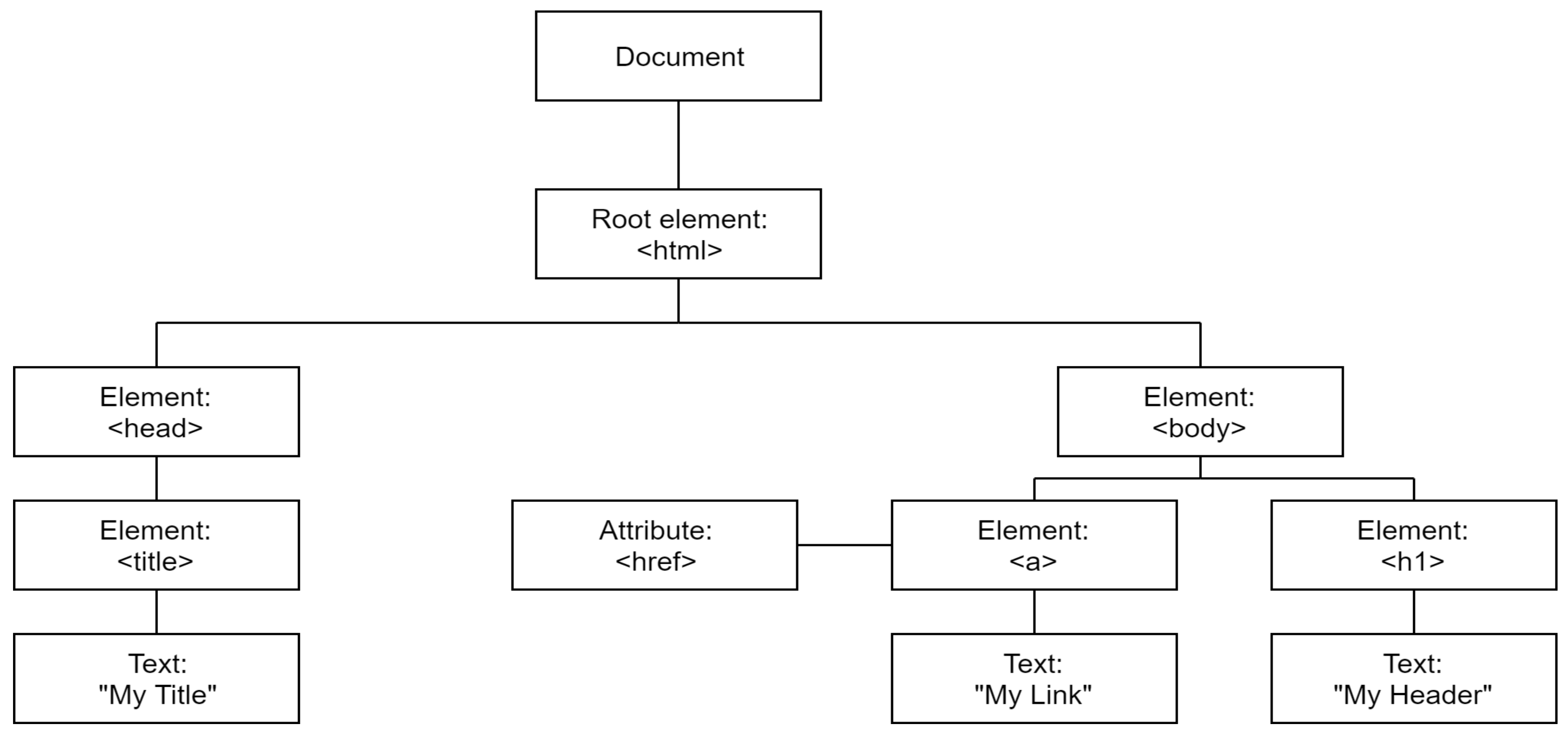


Figure 1. Document Object Model (DOM) of a simple webpage

DOM-based XSS

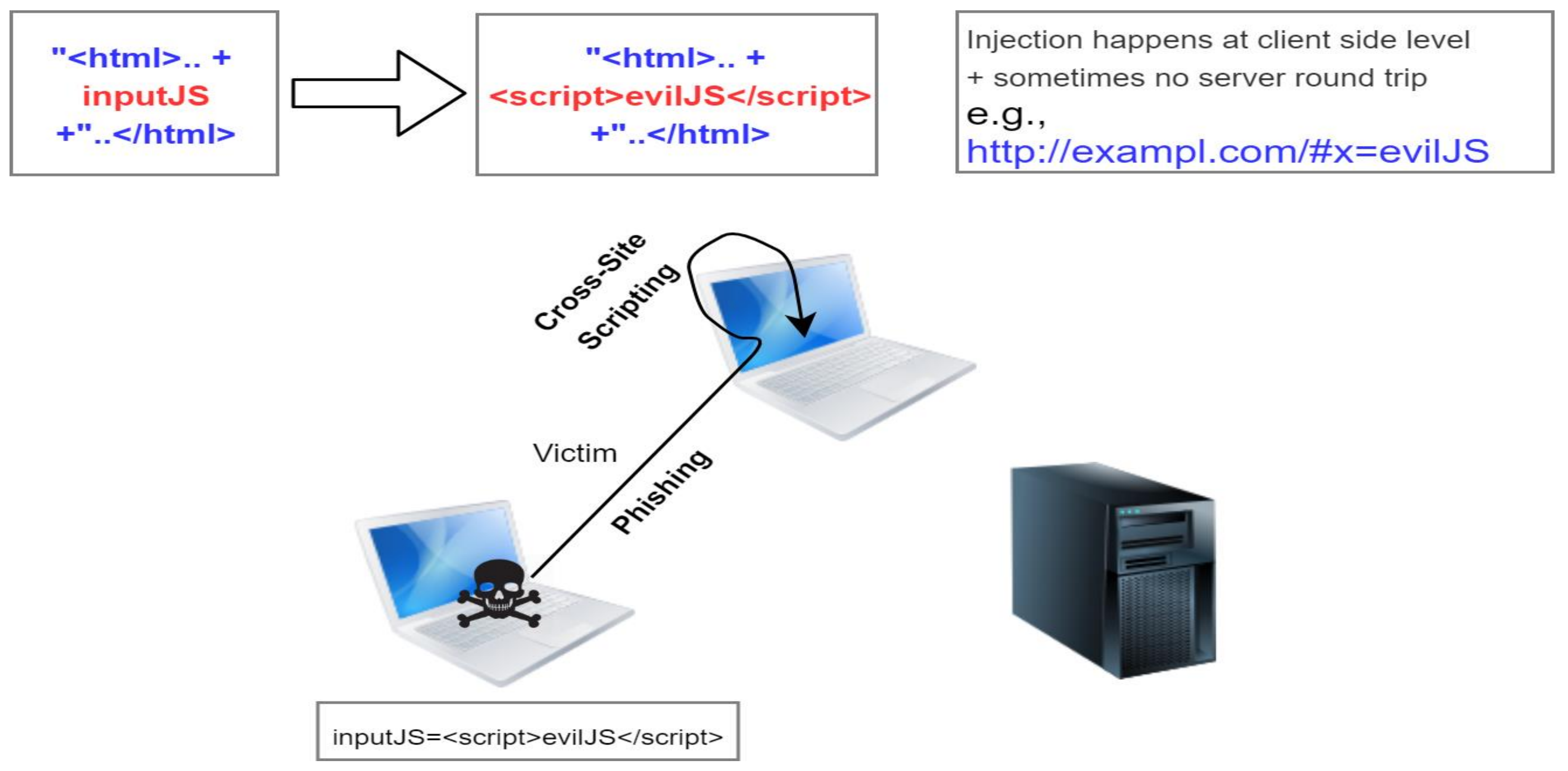
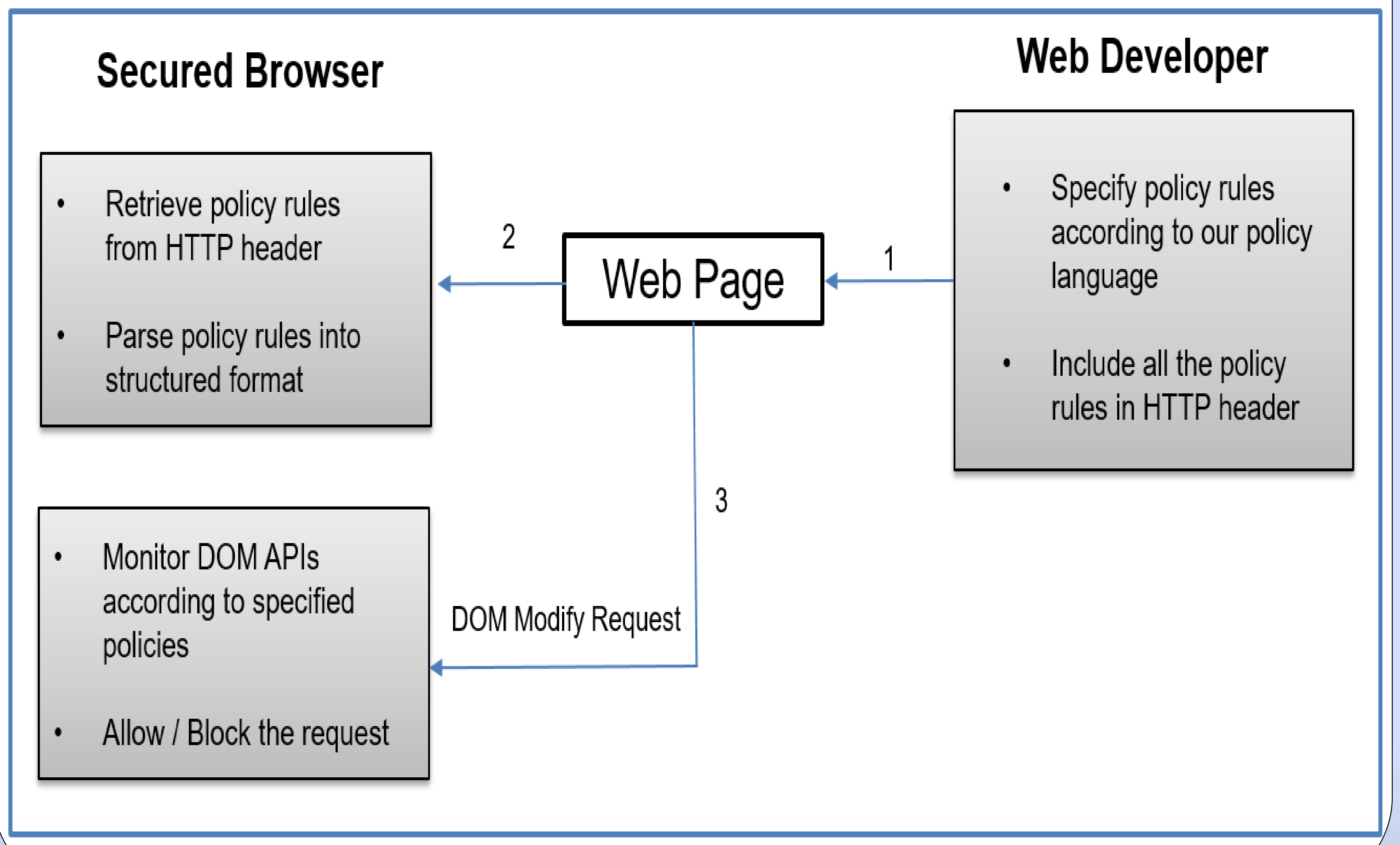

"<html>.. + inputJS +"..</html>
"<html>.. + <script>evilJS</script> +"..</html>
Injection happens at client side level
+ sometimes no server round trip
e.g.,
http://exampl.com/#x=evilJS
Cross-Site Scripting
Victim
Phishing
inputJS=<script>evilJS</script>

Figure 2. Example of a DOM-based XSS attack

Proposed Solution



- Retrieve policy rules from HTTP header
- Parse policy rules into structured format
- Monitor DOM APIs according to specified policies
- Allow / Block the request
- Specify policy rules according to our policy language
- Include all the policy rules in HTTP header

Figure 3. Proposed solution implemented in an open source browser Chromium

Policy Language

Selectors

1. Tag Selector

```
p {
    Policy directives that applies to <p>
}
```

2. ID Selector

```
#some-id {
    Policy directives that applies to <h1 id="some-id">
}
```

3. Class Selector

```
.note {
   Policy directives that applies to <div class="note">
}
```

Policy Example

```
p {
    attribute-whitelist: class, name, id;
    event-blacklist: click;
} #LoginForm {
    protected: true;
}
.note {
   style-modifications: false;
}
```

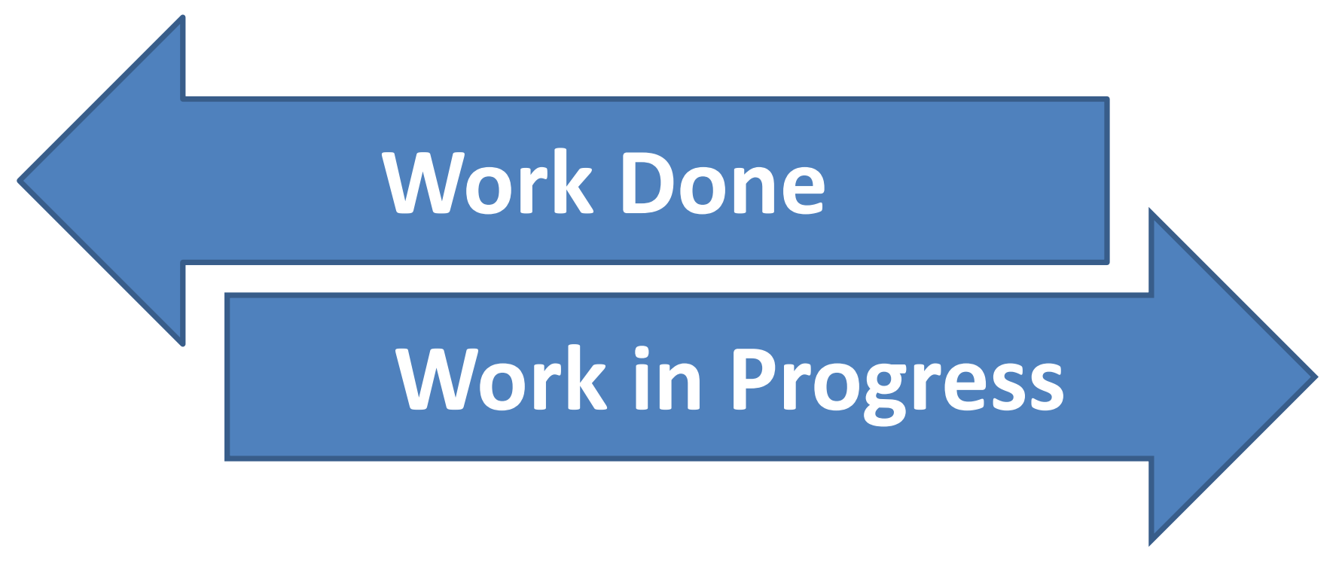
Directives List

Directive Name	Accepted Value	Description	Default value
attribute-attachment	true | false	Controls the attachment of attributes	true
attribute-whitelist	List of attributes	Contains white-list of attributes	
attribute-blacklist	List of attributes	Contains black-list of attributes	
event-attachment	true | false	Controls the addition of events	true
event-whitelist	List of events	Contains white-list of events	
event-blacklist	List of events	Contains black-list of events	
style-modifications	true | false	Controls any type of CSS modifications	true
shadow-attachment	true | false	Controls the attachment of shadow DOM	false
protected	true | false	Controls any type of DOM modifications	Static or sensitive pages (login, about us, contact us) will be protected by default

Table 1. List of Directives that web developer can use to specify policies to protect the DOM

1. Retrieving policy string through HTTP header

2. Parsing policy string



3. Hooking DOM APIs

Conclusion

- We present an efficient client-side system to detect legit or malicious change in DOM of the webpage
- The web developer is able to control what part of the DOM can be modified or not using the policy language