Cross Site Scripting (XSS)

Overview

- What is XSS?
- Is XSS Important?
- Exploiting XSS
- Preventing XSS
- BeEF Usage
- Conclusion

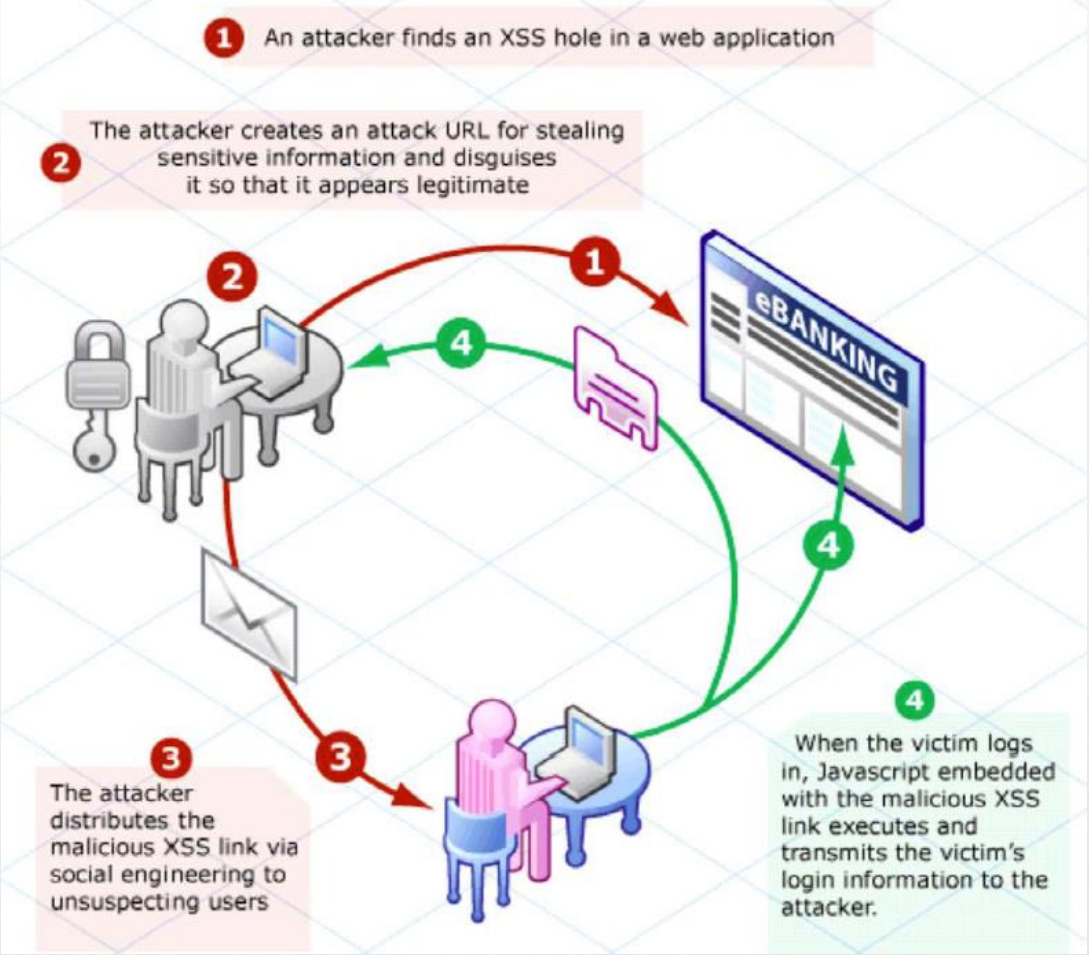
What is XSS?

- XSS is a vulnerability that allows an attacker to run arbitrary JavaScript in the context of the vulnerable website.
- XSS bypasses same-origin policy protection
  - "The policy permits scripts running on pages originating from the same site to access each other's methods and properties with no specific restrictions, but prevents access to most methods and properties across pages on different sites."
  - "The term 'origin' is defined using the domain name, application layer protocol, and (in most browsers) TCP port"
- Requires some sort of social engineering to exploit.

Types of XSS

- Reflected XSS
- Stored XSS (a.k.a. “Persistent XSS”)
- DOM Based XSS

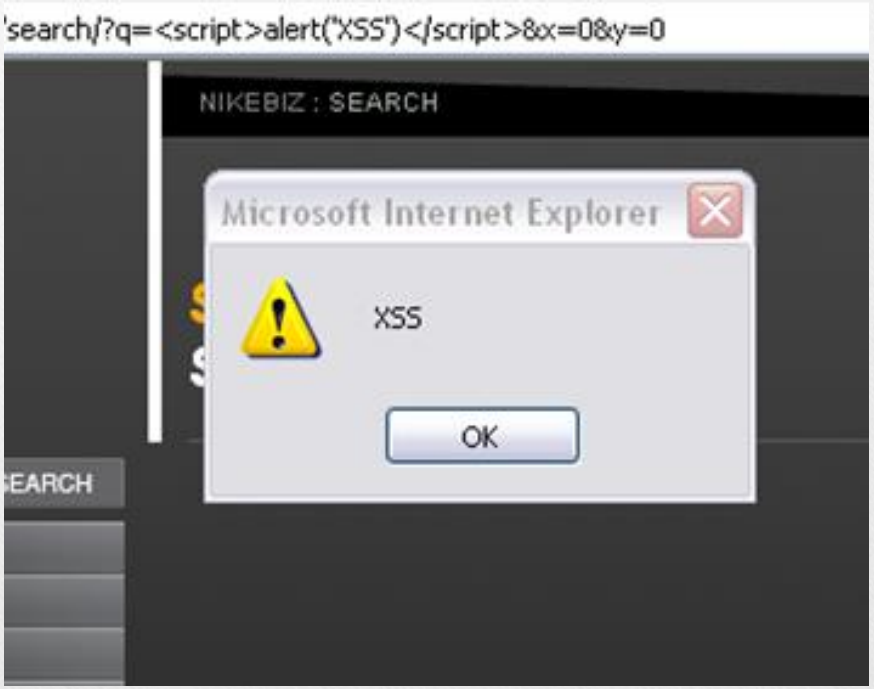
Reflected XSS



Reflected XSS Example

- Exploit URL:
  - http://www.nikebiz.com/search/?q=<script>alert('XSS')</script>&x=0&y=0
- HTML returned to victim:
  - <h2>Search Results
 Search: "<script>alert('XSS')</script>"</h2>

Reflected XSS Example



Stored XSS

- JavaScript supplied by the attacker is stored by the website (e.g. in a database)
- Doesn't require the victim to supply the JavaScript somehow, just visit the exploited web page
- More dangerous than Reflected XSS
  - Has resulted in many XSS worms on high profile sites like MySpace and Twitter

DOM Based XSS

- Occur in the content processing stages performed by the client

```
<select><script>
document.write("<OPTION
value=1>"+document.location.href.substring(docum
ent.location.href.indexOf("default=")+8)+"</OPTI
ON>");
</script></select>
```

- http://www.some.site/page.html?default=ASP.NET
- /page.html?default=<script>alert(document.cookie)</script>

- Source: http://en.wikipedia.org/wiki/Cross-site_scripting
- Source: http://www.owasp.org/index.php/DOM_Based_XSS

Is XSS Dangerous?

- Yes
- OWASP Top 2
- Defeats Same Origin Policy
- Just think, any JavaScript you want will be run in the victim's browser in the context of the vulnerable web page
- Hmmm, what can you do with JavaScript?

What can you do with JavaScript?

- Pop-up alerts and prompts
- Access/Modify DOM (Document Object Model)
  - Access cookies/session tokens
  - "Circumvent" same-origin policy
  - Virtually deface web page
- Detect installed programs
- Detect browser history
- Capture keystrokes
- Port scan the local network

What can you do with JavaScript?

- Induce user actions
- Redirect to a different web site
- Determine if they are logged on to a particular site
- Capture clipboard content
- Detect if the browser is being run in a virtual machine
- Rewrite the status bar
- Exploit browser vulnerabilities
- Launch executable files (in some cases)

XSS Cases

- Form Injection
- Pop-Up Alert
- Cookie Stealing
- XSS Worms (https://www.youtube.com/watch?v=DtnuaHl378M)
- Resource:
  - https://pentest-tools.com/blog/xss-attacks-practical-scenarios/
  - https://brutelogic.com.br/blog/the-7-main-xss-cases-everyone-should-know/

Cause of Injection Vulnerabilities: Improper Handling of User-Supplied Data

- >= 80% of web security issues caused by this!
- **NEVER Trust User/Client Input!**
  - Client-side checks/controls have to be invoked on the server too.
- Improper Input Validation
- **Improper Output Validation**

Preventing Injection Vulnerabilities In Your Apps

- **Validate** Input
  - Letters in a number field?
  - 10 digits for 4 digit year field?
  - Often only need alphanumeric
  - Careful with < > " ' and =
  - Whitelist (e.g. /[a-zA-Z0-9]{0,20}/)
  - Reject, don't try and sanitize

Preventing XSS In Your Applications

- **Validate** Output
  - Encode HTML Output
    - If data came from user input, a database, or a file
    - Response.Write(HttpUtility.HtmlEncode(Request.Form["name"]));
    - Not 100% effective but prevents most vulnerabilities
  - Encode URL Output
    - If returning URL strings
    - Response.Write(HttpUtility.UrlEncode(urlString));
- How To: Prevent Cross-Site Scripting in ASP.NET
  - https://msdn.microsoft.com/en-us/library/ms998274.aspx
- XSS Prevention Cheat Sheet:
  - https://cheatsheetseries.owasp.org/cheatsheets/SQL_Injection_Prevention_Cheat_Sheet.html

RULE #0 - Never Insert Untrusted Data

- <script>**...NEVER PUT UNTRUSTED DATA HERE...**</script> directly in a script
- <!--**...NEVER PUT UNTRUSTED DATA HERE...**--> inside an HTML comment
- <div **...NEVER PUT UNTRUSTED DATA HERE...**=test /> in an attribute name
- <**...NEVER PUT UNTRUSTED DATA HERE...** href="/test" /> in a tag name

RULE #1 - HTML Escape Before Inserting Untrusted Data

- <body>**...ESCAPE UNTRUSTED DATA BEFORE PUTTING HERE…**</body>
- **…ESCAPE UNTRUSTED DATA BEFORE PUTTING HERE…**
- any other normal HTML elements

RULE #1 (continued)

- Escape these characters:
  - `&` --> `&`
  - `<` --> `<`
  - `>` --> `>`
  - `"` --> `&quot;`
  - `'` --> `&#x27;` `&apos;` is not recommended
  - `/` --> `&#x2F;`
    - forward slash is included as it helps end an HTML entity
- Remember *`HttpUtility.HtmlEncode()`*

RULE #2 - Attribute Escape Before Inserting Untrusted Data

- <div attr=**…ESCAPE UNTRUSTED DATA BEFORE PUTTING HERE…**>content
- content
- content
- Except for alphanumeric characters, escape all characters with ASCII values less than 256 with the &#xHH; format or named entity if available. Examples: " '

RULE #3 - JavaScript Escape Before Inserting Untrusted Data Values

- The only safe place to put untrusted data into these event handlers as a quoted "data value.“
- <script>alert('**...ESCAPE UNTRUSTED DATA BEFORE PUTTING HERE...**')</script> inside a quoted string
- <script>x='**...ESCAPE UNTRUSTED DATA BEFORE PUTTING HERE...**'</script> one side of a quoted expression
- <div onmouseover="x='**...ESCAPE UNTRUSTED DATA BEFORE PUTTING HERE...**'" inside quoted event handler

RULE #3 (continued)

- Except for alphanumeric characters, escape all characters less than 256 with the \xHH format. Example: \x22 not \”
- But be careful!

<script> window.setInterval('**...EVEN IF YOU ESCAPE UNTRUSTED DATA YOU ARE XSSED HERE...**'); </script>

RULE #4 - CSS Escape Before Inserting Untrusted Data

- <style>selector { property : **...ESCAPE UNTRUSTED DATA BEFORE PUTTING HERE...**; } </style> property value
- <span style=property : **...ESCAPE UNTRUSTED DATA BEFORE PUTTING HERE...**;>text</style> property value
- Except for alphanumeric characters, escape all characters with ASCII values less than 256 with the \HH escaping format. Example: \22 not \”

RULE #5 - URL Escape into HTML URL Parameter Values

- <a href="http://www.somesite.com?test=**...URL ESCAPE UNTRUSTED DATA BEFORE PUTTING HERE...**">link
- Except for alphanumeric characters, escape all characters with ASCII values less than 256 with the %HH escaping format. Example: %22
- Remember HttpUtility.UrlEncode()

Reduce Impact of XSS Vulnerabilities

- If Cookies Are Used:
  - Scope as strict as possible
  - Set 'secure' flag
  - **Set 'HttpOnly' flag**

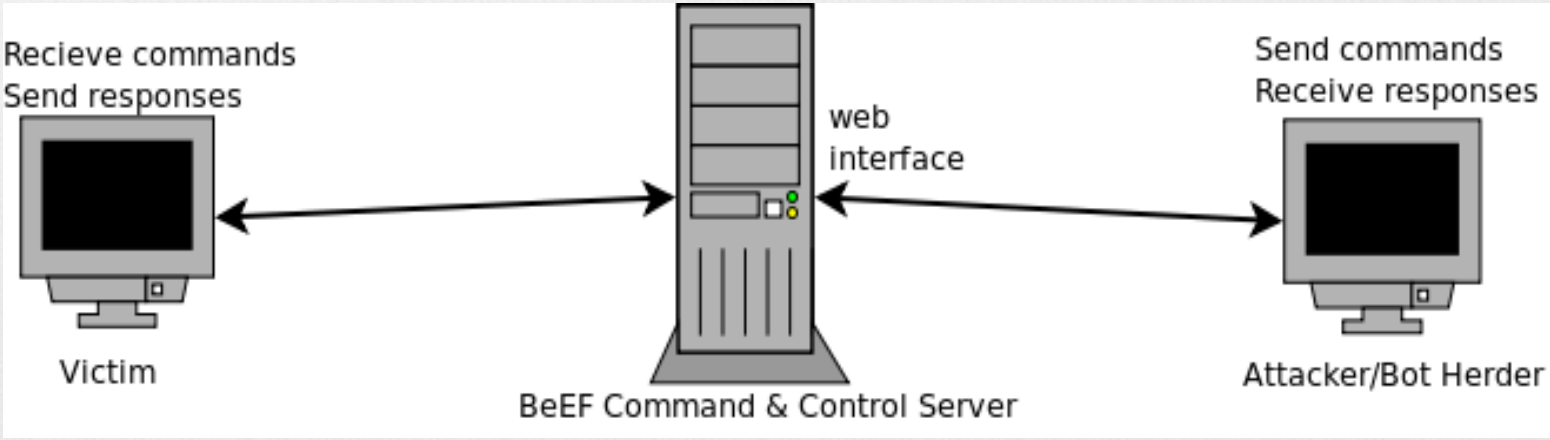
- On the client, consider disabling JavaScript (if possible) or use something like the NoScript Firefox extension.

Further Resources

- Damn Vulnerable Web Application (DVWA)
  - http://www.dvwa.co.uk/
- 5 Practical Scenarios for XSS Attacks
  - https://pentest-tools.com/blog/xss-attacks-practical-scenarios/
- BeEFproject
  - https://beefproject.com/
  - https://tools.kali.org/exploitation-tools/beef-xss

Demo: BeEF

- Browser Exploitation Framework
- Written by Wade Alcorn
- http://www.bindshell.net/tools/beef/
- Architecture:



Conclusion

- XSS vulnerabilities are bad (?)
- Avoid introducing XSS vulnerabilities in your code.
  - Please. They will only cause delays in getting your apps into production.
- Give me your email, I have a link you *really* need to see. ☺