Cross Site Scripting

COMP620

“If what you are getting online is for free, you are not the customer, you are the product.”

Jonathan Zittrain

professor of Internet law

Typical Web Application Design

- Runs on a Web server and the client's browser
  - HTML and JavaScript on the browser
  - PHP and SQL on the server
- The user enters data on the web form
- JavaScript sends the data to the server as a request to execute a program on the server
- The program executes on the server possibly accessing data bases or other server data
- Data is returned to the client browser, often in XML format
- JavaScript receives the data and displays it in the browser

Displaying Data

- Consider an element in HTML

```
<p id="thing">Mom and apple pie</p>
```

- The text can be changed by JavaScript

```
<script>
document.getElementById("thing").innerHTML =
          "Evil and corruption";
</script>
```

- Any JavaScript in the replacement text will be executed

Threats along the way

- Input data with JavaScript might be displayed
  - Cross Site Scripting
- HTTP sent to and from the server might be modified
  - Parameter modification
- Input data used to create an SQL request could create an unintentional command
  - SQL Injection

XSS Example

- Client browser script asks the user for their name
- It sends a message to the web server

https://example.com/myapp.php?user=Ken

XSS Example

- The name is “Reflected” back from the Web server to the client in a web page

Hello Ken.

XSS Example

- Instead of entering their name at the prompt, the user could enter

<script>alert('Got you');</script>

- When displayed on a web page

XSS vulnerability with JavaScript

```
<html>
<head>
<script type="text/javascript">
function showName() {
  var userin = document.getElementById("instuff").value;
  document.write("Hello " + userin);
}
</script>
</head>
<body>
<p>Example of XSS</p>
Username: <input type="text" onchange="showName();" id="instuff"
    maxlength="80" size="50" />
Password: <input type="text"/>
</body>
</html>
```

Top 10 Vulnerabilities

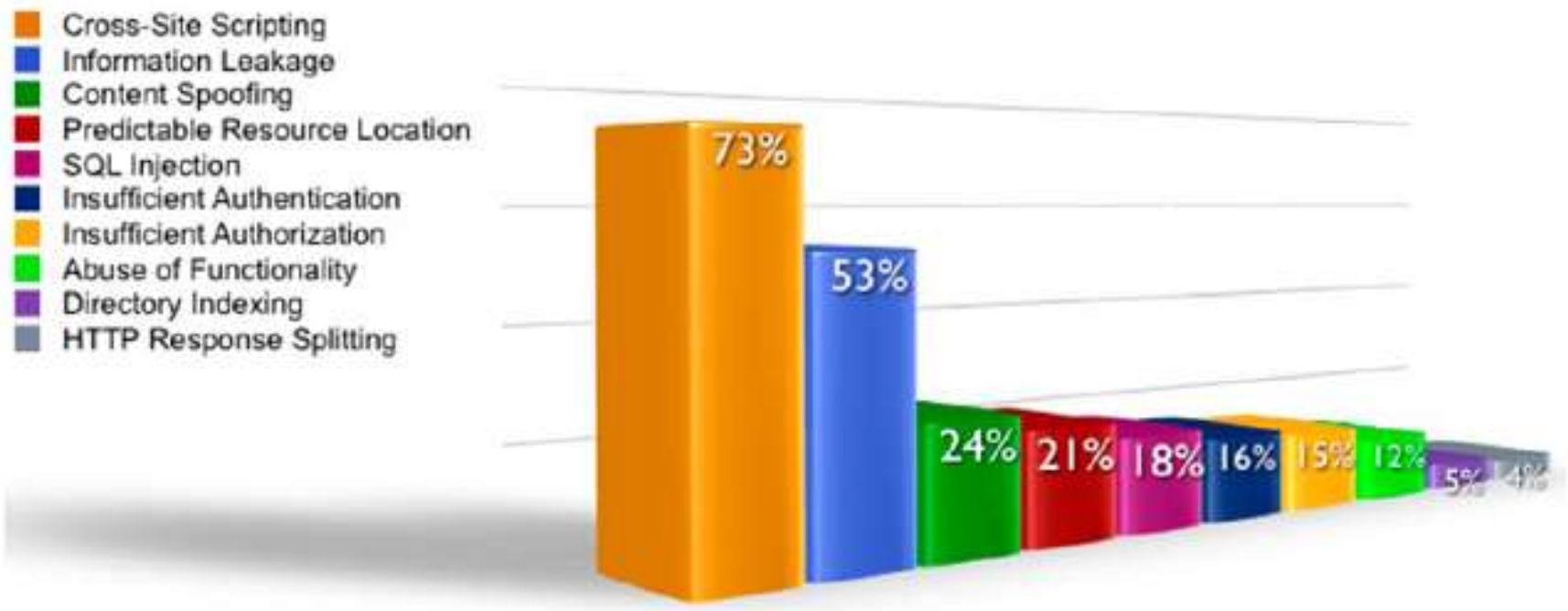
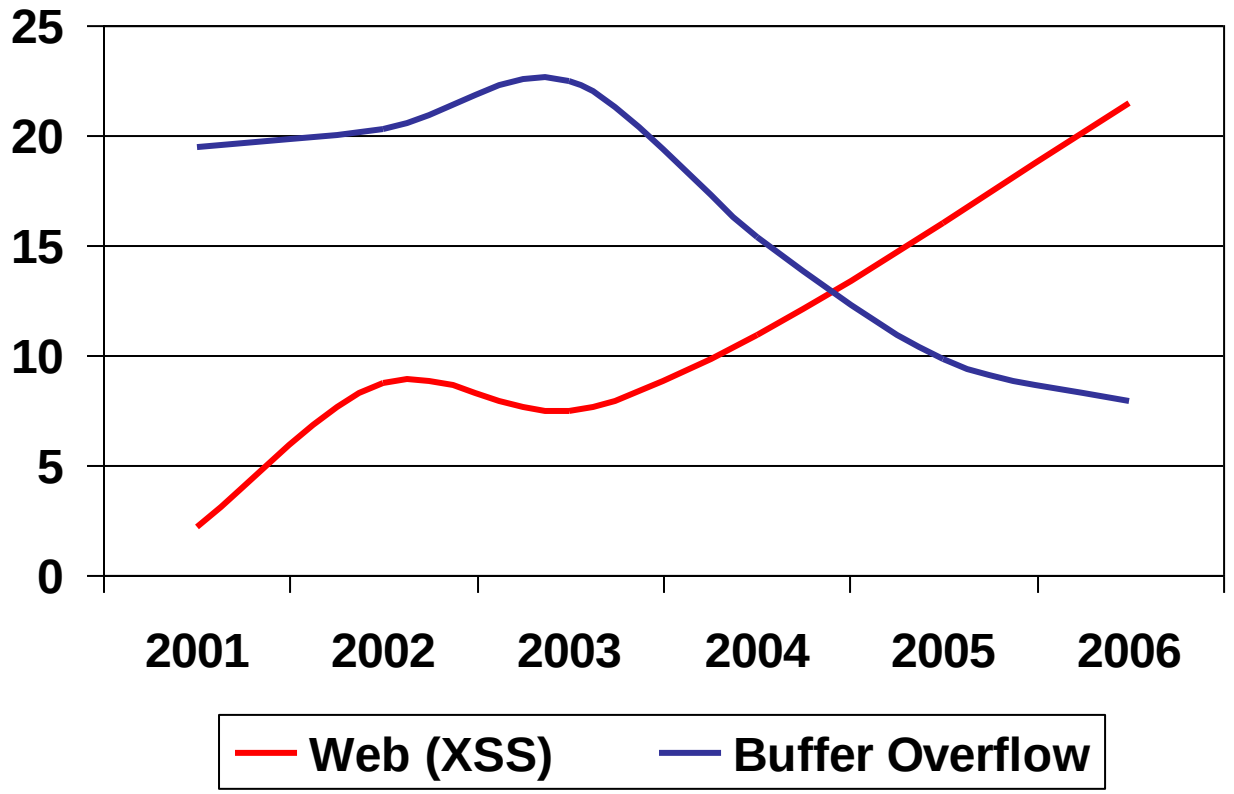


Figure 2. Top 10 vulnerability classes by percentage likelihood.

Vulnerability Stats

Source: MITRE CVE trends

Majority of vulnerabilities now found in web software



Attacking Yourself

- If you enter script in an input field that is then displayed in your browser, aren’t you attacking yourself?

Non-Persistent (Reflected) XSS Attacks

- Most common type
- With invalidated user-supplied data in a resulting webpage without html encoding, client-side code can be injected into the dynamic page
- Then with some social engineering
  - Manipulating someone to perform actions
- An attacker convinces a user to follow a malicious URL which injects code into the resulting page
- Now the attacker has full access to that pages content

Persistent (Stored) XSS Attacks

- Allows the most powerful kinds of attacks
- First data is stored in a server provided by a web application
- It is later shown to a user on a webpage without any html encoding
  - Ex: Online message board that allows users to post messages for other users to read
- With this method, malicious scripts can be provided more than once
- An attack can affect a large amount of users and the application can also be infected by a XSS Virus or Worm

DOM-Based (Local) XSS Attacks

- Document Object Model
  - Standard object model for representing html
- Problem exists within the page's client side script
- If an attacker hosts a malicious site, which contains a vulnerable website on a clients local system, a script can be injected
- Now the attacker can run the privileges of that users browser on their system "Local Zone"
- Can be either persistent or non-persistent

OWASP Guidelines

- XSS (Cross Site Scripting) Prevention Cheat Sheet

- An HTML document has slots where a developer is allowed to put untrusted data
- Putting untrusted data in other places in the HTML is not allowed
- You need to take certain steps to make sure that the data does not break out of that slot

HTML Entity Encode

- HTML Entity Encode converts certain characters (particularly those used in scripts) to escape sequences

  & --> &

  < --> <

  > --> >

  " --> "

- This can help prevent XSS in certain cases

Using User Input

- Use HTML Entity Encode for all data displayed as normal text
- Use HTML Entity Encode for untrusted data inserted into html common attributes
- The only safe place to put untrusted data into JavaScript is inside a quoted "data value"

Sanitizing Input

- A good way to avoid problems is to not allow characters that should not be in the input
  - “White Listing” is allowing only valid characters
  - “Black Listing” is preventing invalid characters
- A user’s name should not include brackets
- The sanitizing should be aware of encoded data
- There are many libraries to provide sanitation

What names will cause input problems?

A. Susan O’Malley
B. Kurt Gödel
C. Ségolène Royal
D. All the above
E. None of the above

XSS Problem

- XSS is a complex problem that is not going away anytime soon
- The browser is insecure by design
- It understands JavaScript
- Disabling scripting seriously dampens the user’s browsing experience