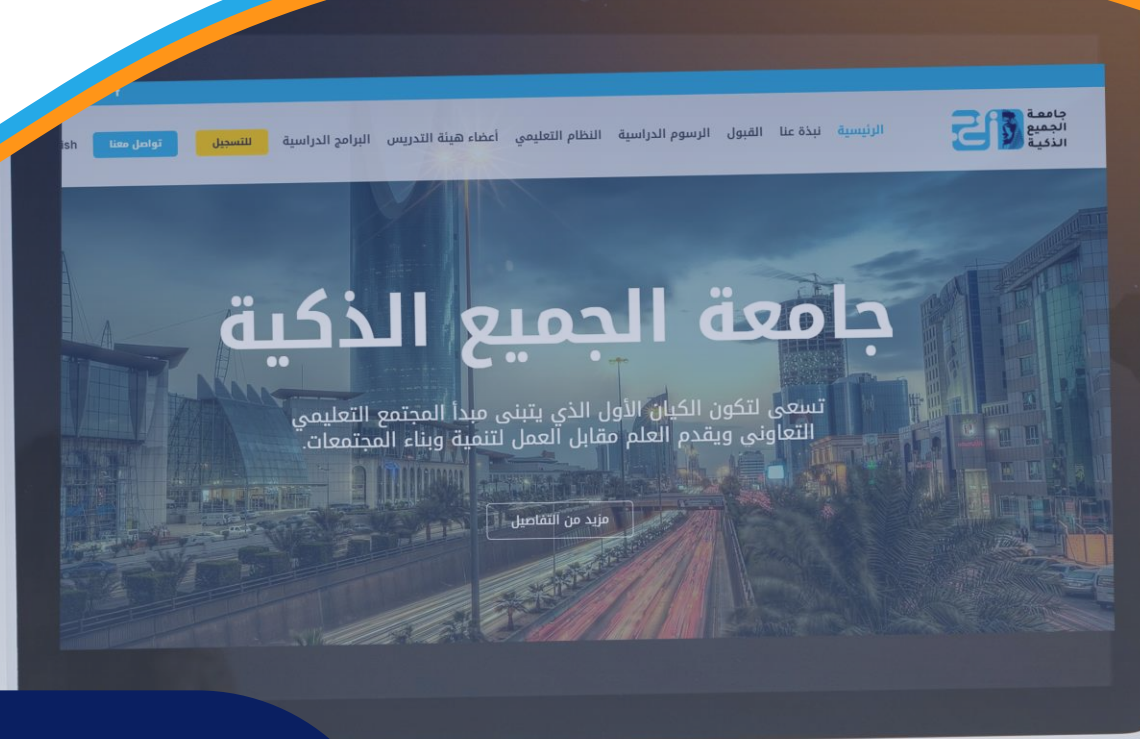




جامعة الجميع الذكية
EVERYONE'S SMART UNIVERSITY



IoT Security
Prepared by Dr. Abdelrhman Ahmed
Lecture 8

IoT Risk & Web/Cloud Interface
Security

IoT devices bring new functionality and efficiency to many business processes. However, many of their technological attributes (some of which are what make them useful) can also make them vulnerable to cyberattacks.

Those attributes include:

- **Centralized Architecture**

IoT devices are often operated under a centralized architecture. Consider, for example, an enterprise environment where a single database houses the information collected by all IoT devices across the organization. That's a massive amount of valuable data in one place. In the event it is compromised, this would mean a catastrophic loss.

In order to function properly, IoT devices including sensors must gather large amounts of telemetry and data. While it may be less expensive to build the framework with a centralized architecture than to use multiple databases, doing so also results in a wider attack surface.

- **Marriage of Virtual and Tangible Environments**

IoT devices are useful because they connect the tangible and virtual environments. The data they collect from their physical surroundings can be read and used accordingly via a virtual interface.

For example, based on the data collected by a home thermostat, a user 15 miles away can adjust the home's temperature up or down without being on the premises. While this is convenient, this technology also means that attacks can bring immediate real-world consequences, such as outages.



- Ecosystem Complexity

As IoT devices become more available and diverse, complex IoT ecosystems can be built with a host of different devices on a network. This makes the connectivity between them more dynamic and difficult to manage. In a complex environment, IoT devices have expanded capabilities and provide a broader attack surface.

- The IT, OT, and IoT Connection

IoT devices have become a staple of operational technology (OT) in recent years. Machines that control physical processes, such as a Programmable Logic Controller used to open and close valves in a wastewater system, now receive data from connected IoT sensors and gauges.

Within enterprise systems, OT systems used to work independently, meaning they weren't connected to IT systems, nor did they have an external internet connection. Now, however, IoT devices utilized in OT are regularly accessible not only inside the corporate network but outside it. Newly discovered vulnerabilities for hard-to-access IoT devices present a massive security challenge.

- **Lack of Network Visibility**

For many IoT devices, it's difficult for network security to detect them and their network connections. This also means the system can't easily identify threats to these devices as they arise. This is why consistent monitoring for new IoT devices is so important.

While large-scale IoT deployments provide a number of benefits, more IoT devices — and consequently, a more complex IoT ecosystem — also mean more security vulnerabilities from the edge to the cloud. When trying to assess risks, most organizations think in terms of the devices themselves instead of the services offered through the cloud as probable attack vectors. But with all the data being amassed through the edge, organizations turn to cloud services to ease the strain on their own IT infrastructure, which then creates a whole new layer of cloud security issues. Given this, many companies, unfortunately, the urgency of implementing a cybersecurity strategy for the IoT until it's too late.

Prevention in this instance is infinitely better than a pound of cure; a measured approach can help security teams mitigate risks now as threats multiply and continually evolve in the future.

Enterprises that have just started their IoT adoption or are looking to expand their established IoT networks are pretty much in the same boat when it comes to managing, monitoring, and securing their connected IoT environments. The IoT relies on cloud computing to **integrate servers, analyze information gathered from sensors, boost processing power, and increase storage capacity**. The security risks to IoT herein discussed are specific to the cloud.



Major cloud-related threats facing the IoT:-

- **API gateway misconfigurations**

Application programming interfaces (APIs) make fast and efficient communication with IoT cloud servers possible for device manufacturers and developers. An IoT device can request specific actions from the cloud server using an API gateway and vice versa. The gateway serves as a doorway to the cloud as it limits IoT device traffic. It also facilitates specific commands such as turning the device on or off, checking the device's status, upgrading the firmware, downloading or uploading screenshots or videos, and accessing data. Threat actors can use a misconfigured device or a cloud service for malicious activities such as faking a command sequence by changing the logic between the APIs, thus causing more vulnerabilities. Other possible activities include user spoofing, man-in-the-middle (MiTM) attacks, denial of service (DoS) attacks, and session replays.

- **Cloud misconfigurations:-**

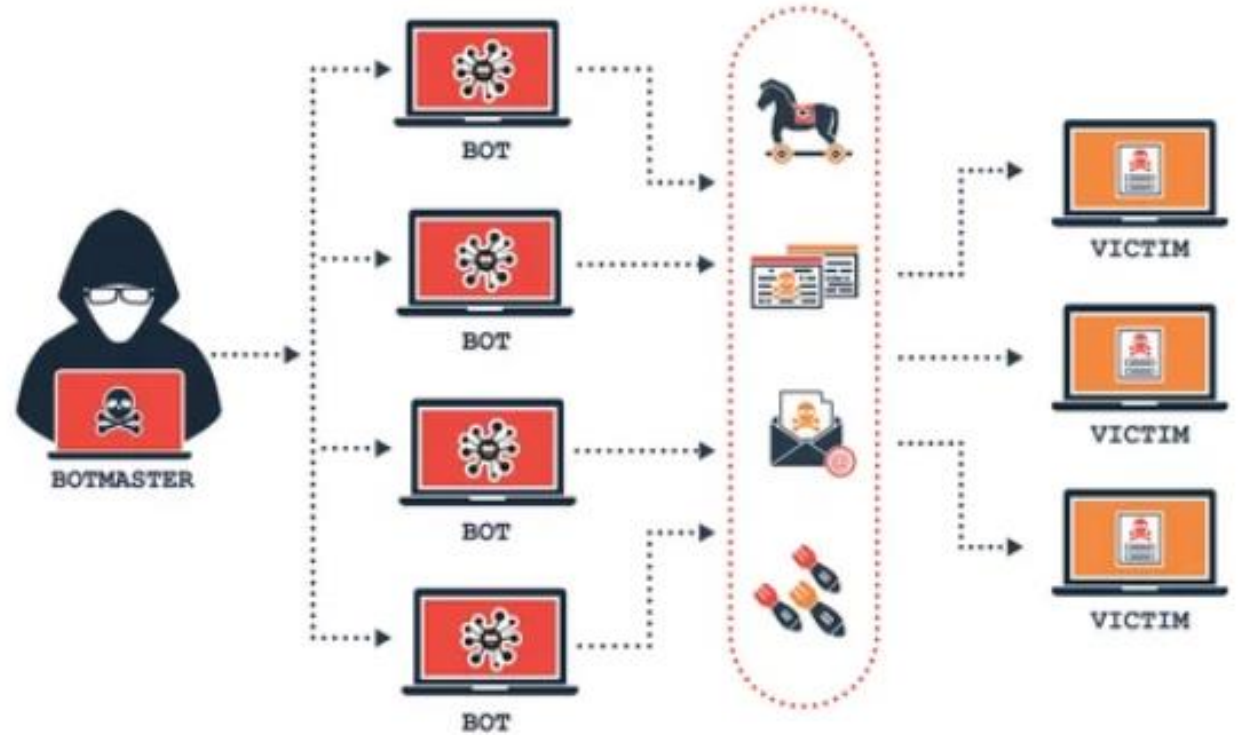
Misconfiguration is a common problem in cloud computing, as it can provide a path for attacks like data exfiltration. Misconfigurations and similar security flaws in the cloud can cause serious damage to the IoT ecosystem to which it belongs.

- **Unsecure communication and data flow between the edge and the cloud:-**

Access controls and the integrity of the data sent between the endpoints is endangered if the cloud lacks security features such as authentication and encryption. Weaknesses in the security of data traffic or path exposes the device or the cloud server.

- **Misuse of cloud services:-**

Threat actors can willfully violate their contracts with the cloud platform by launching attacks on the networks such as brute force attacks, trojans, SQL injections, botnets, phishing, and denial of service (DoS) attacks. Some cloud service providers may not always have the ability to detect the attacks on their networks because they can neither generate nor block the attacks.

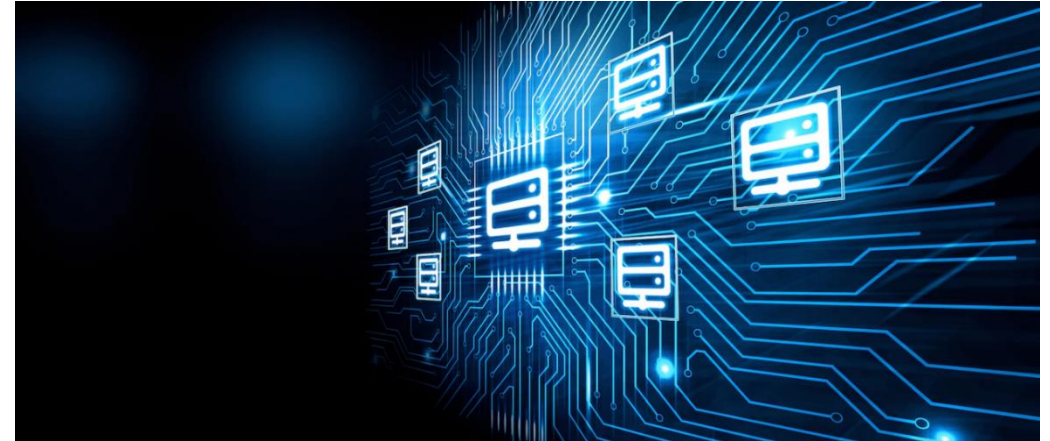


Since Hypertext Transfer Protocol (HTTP) is stateless, special provisions must be made outside of the protocol for the server to remember previous interactions with a user. In web applications, a “session” refers to a data structure stored on the server that is associated with a specific user during a limited time period. A session is typically initiated by user authentication and is terminated when they log out (or otherwise terminate the session). The session associated with the user is identified through a “session token” that is originally generated by the server and is delivered to the browser as a cookie. The browser then returns the session token with subsequent requests, allowing the server to retrieve the corresponding session object and thus maintain context with that user.



Avoiding or remediating Session Management vulnerabilities is straightforward if you observe the following guidelines:

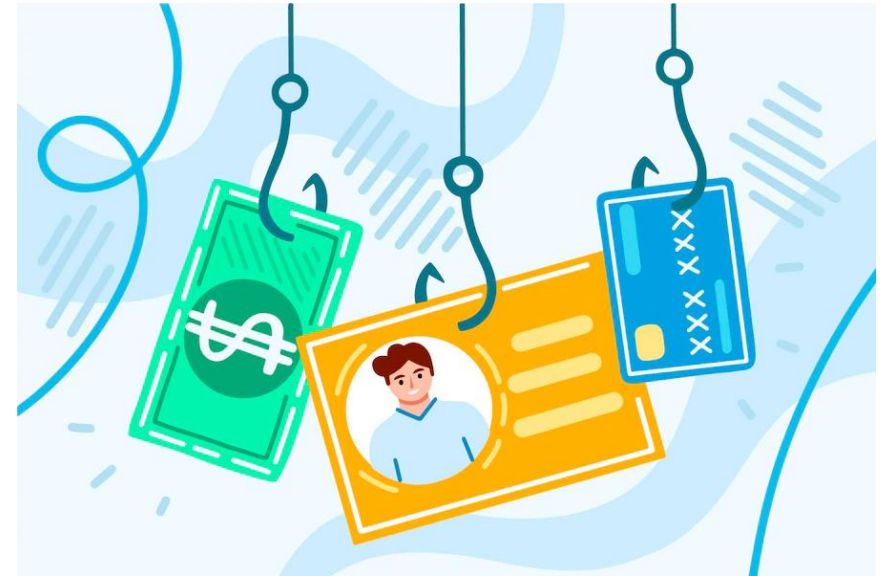
- Use an up-to-date web-server framework to generate and manage the session identifier token, as this will guarantee values that defy prediction.
1. Note that the default implementation will be to represent the session identifier token as a cookie
 2. It is particularly important to never design your own session-management solution.



- Take every precaution to ensure that the session identifier remains confidential to the application. This means you must:
 1. Exclusively use HTTPS to transfer the session identifier token.
 2. Never expose the session identifier in a URL, in the contents of a page, or any other insecure location or communication.
 3. Set the “secure” and “httpOnly” flags on the session identifier token cookie. This directs the browser to restrict transfer of the cookie to HTTPS and puts the cookie off-limits to Javascript.



- Ensure that the session identifier token (cookie) has a browser session lifetime. That is, the browser should store the cookie in memory (not in a file) and it should expire and disappear when the browser closes. This is the default lifetime for cookies, and changing it is typically a mistake.
- Implement an “inactivity timeout” for every session. This is an application configuration setting or programmatic setting that should be consistent with documented requirements.



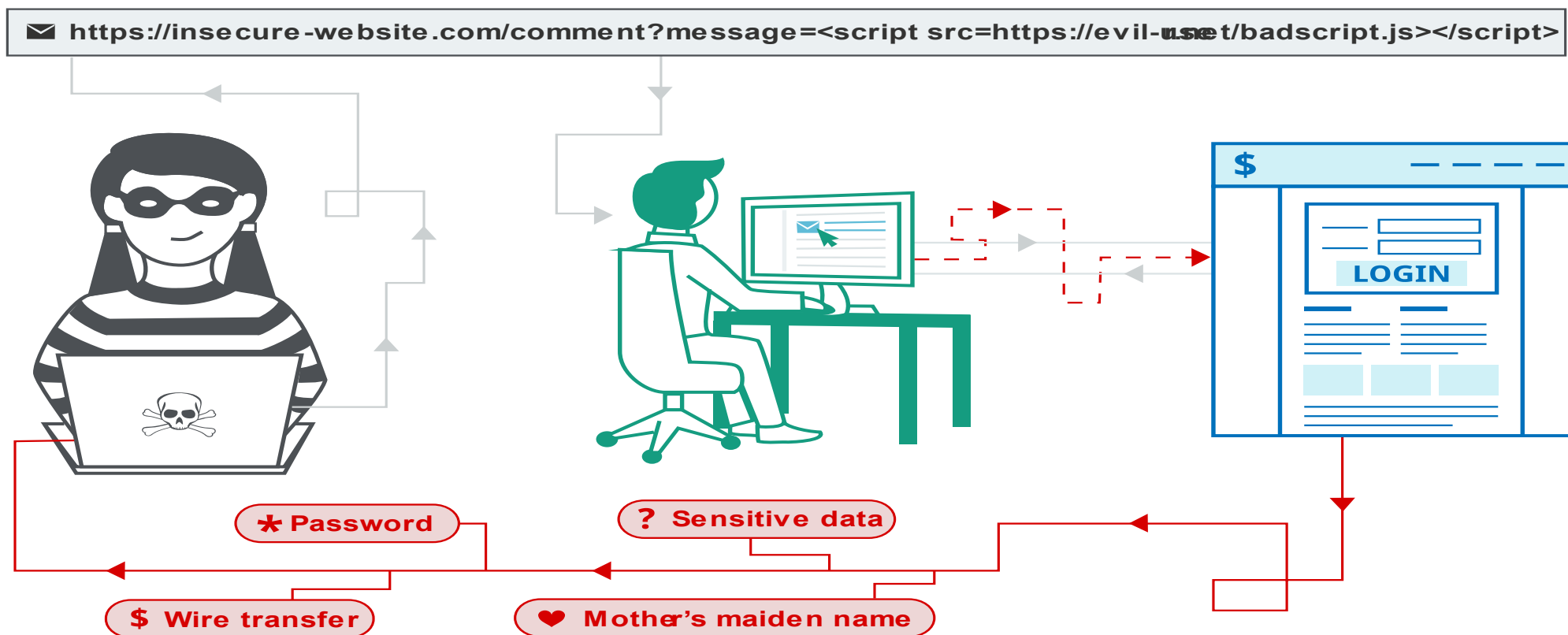
- Ensure that the session on the server is terminated (a.k.a. “invalidated”) when the user logs out.
1. It is not sufficient to simply forward to the login page with a message that the user is “logged out”.
- Verify that a new session identifier value is generated in response to successful authentication, even if one pre-exists login.
1. ASP .NET is an exception to this, as it complements the session identifier with an authentication token.



Cross-site scripting (also known as XSS) is a web security vulnerability that allows an attacker to compromise the interactions that users have with a vulnerable application. It allows an attacker to circumvent the same origin policy, which is designed to segregate different websites from each other. Cross-site scripting vulnerabilities normally allow an attacker to masquerade as a victim user, to carry out any actions that the user is able to perform, and to access any of the user's data. If the victim user has privileged access within the application, then the attacker might be able to gain full control over all of the application's functionality and data. Cross-site scripting works by manipulating a vulnerable web site so that it returns malicious JavaScript to users. When the malicious code executes inside a victim's browser, the attacker can fully compromise their interaction with the application.

Chapter 08

What is Cross-Site Scripting

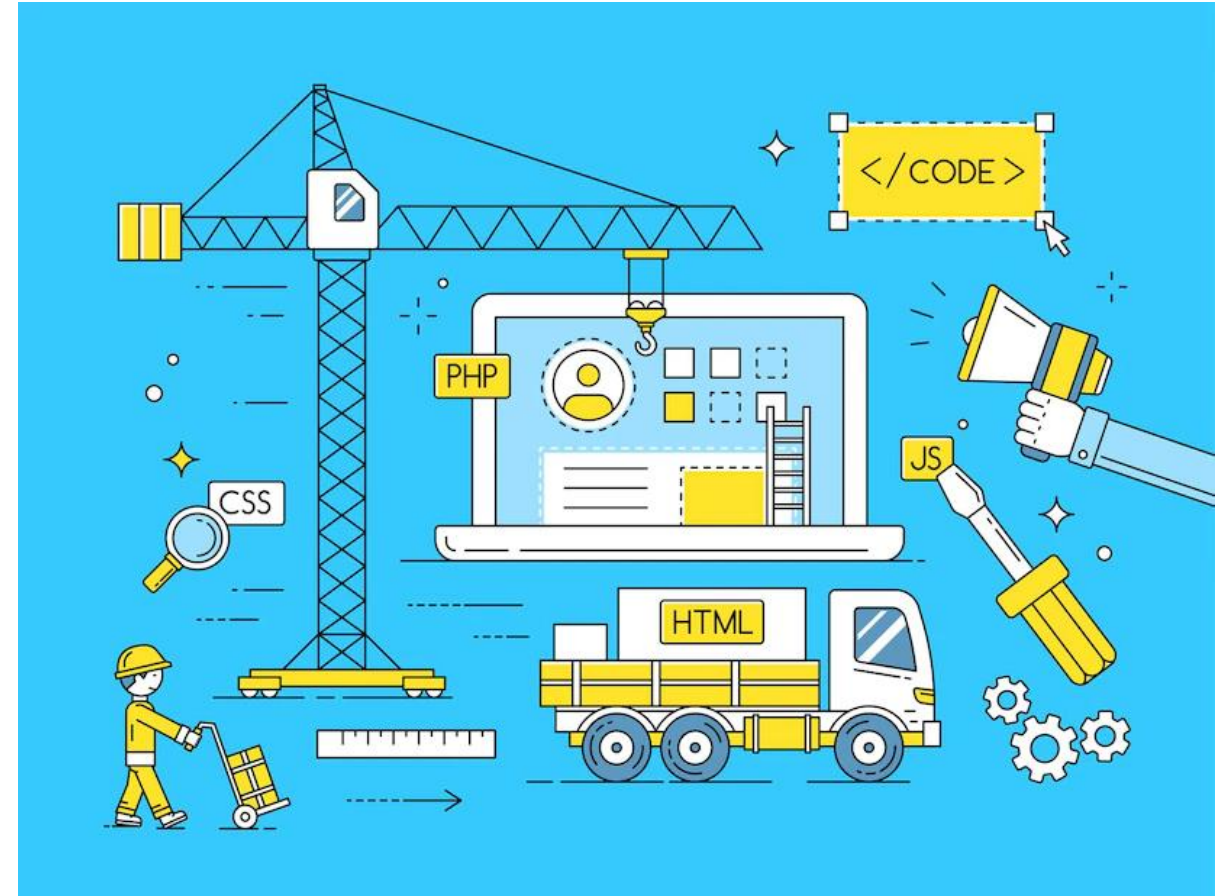


Preventing cross-site scripting is trivial in some cases but can be much harder depending on the complexity of the application and the ways it handles user-controllable data.

In general, effectively preventing XSS vulnerabilities is likely to involve a combination of the following measures:

- **Filter input on arrival:-** At the point where user input is received, filter as strictly as possible based on what is expected or valid input.
- **Encode data on output:-** At the point where user-controllable data is output in HTTP responses, encode the output to prevent it from being interpreted as active content. Depending on the output context, this might require applying combinations of HTML, URL, JavaScript, and CSS encoding.

- **Use appropriate response headers:-** To prevent XSS in HTTP responses that aren't intended to contain any HTML or JavaScript, you can use the Content-Type and X-Content-Type-Options headers to ensure that browsers interpret the responses in the way you intend.
- **Content Security Policy:-** As a last line of defense, you can use Content Security Policy (CSP) to reduce the severity of any XSS vulnerabilities that still occur.



A redirect is a situation when your website or web application causes the user's browser to open another URL (external). A forward is a situation when instead of an external URL, your website or web application causes the browser to go to different parts of the site. Redirects and forwards are technically identical, the only difference is the type of destination: external URLs vs. internal pages.

From the technical point of view, the most common way to create redirects and forwards is for the web server to send a 3xx status code to the client (for example, **302 Found**). This is done using back-end technologies, either in server-side programs or web server configuration. However, you can also create redirects and forwards directly in HTML using **META** tags (**meta http-equiv="Refresh"**) or using client-side JavaScript.

An unvalidated redirect or forward happens if your application uses a URL or a page name that is supplied directly from untrusted input. This makes it possible for an attacker to redirect the browser to a malicious site and use your domain name to gain the victim's trust. For example, if your domain is example.com, the attacker may create the following URL:

```
http://www.example.com/redirect.php?url=http://evil.com
```

The attacker may then send this URL as part of a phishing attempt hoping that example.com at the beginning will have a trustworthy appearance, gain user's trust, and make them fall for the phishing scam.

Note that exactly the same may happen if the intent of your application is to forward users to different pages within your site. If the user input parameters are unvalidated, a forwarding mechanism may be used to create a redirect.

An Example of an Unvalidated Redirect

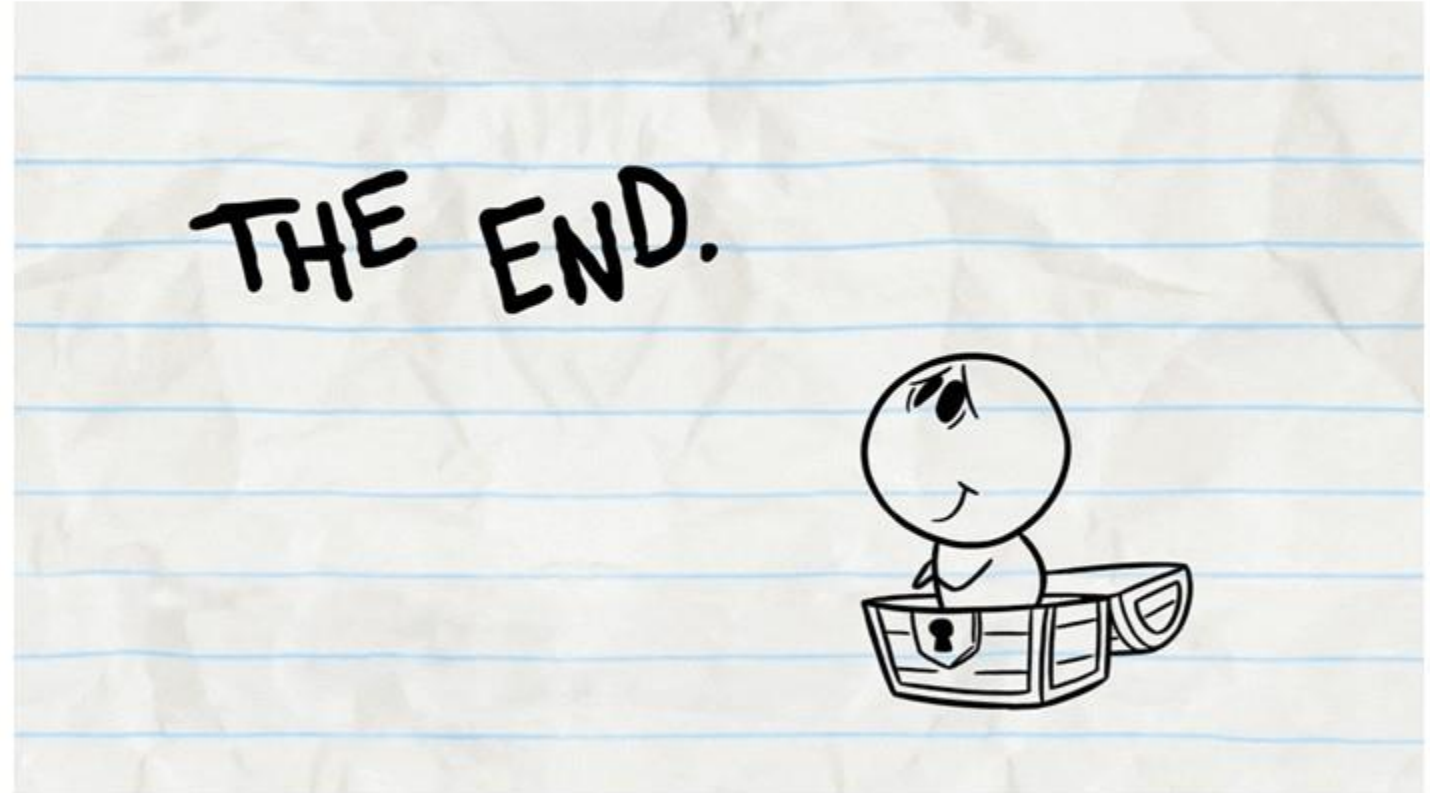
If for some reason your application needs to redirect the user to another URL, some developers might use the following simple construct (example in PHP):

```
$new_url = $_GET['url'];  
header("Location: " . $new_url);
```

This is an unvalidated redirect (in this case, a completely open redirect) because the attacker may supply a malicious URL in the url parameter of the GET request and this target URL will be used in the Location: header.

The End

Please feel free to contact me anytime if you have any questions



جامعة زايد
EVERYONE'S SMART UNIVERSITY