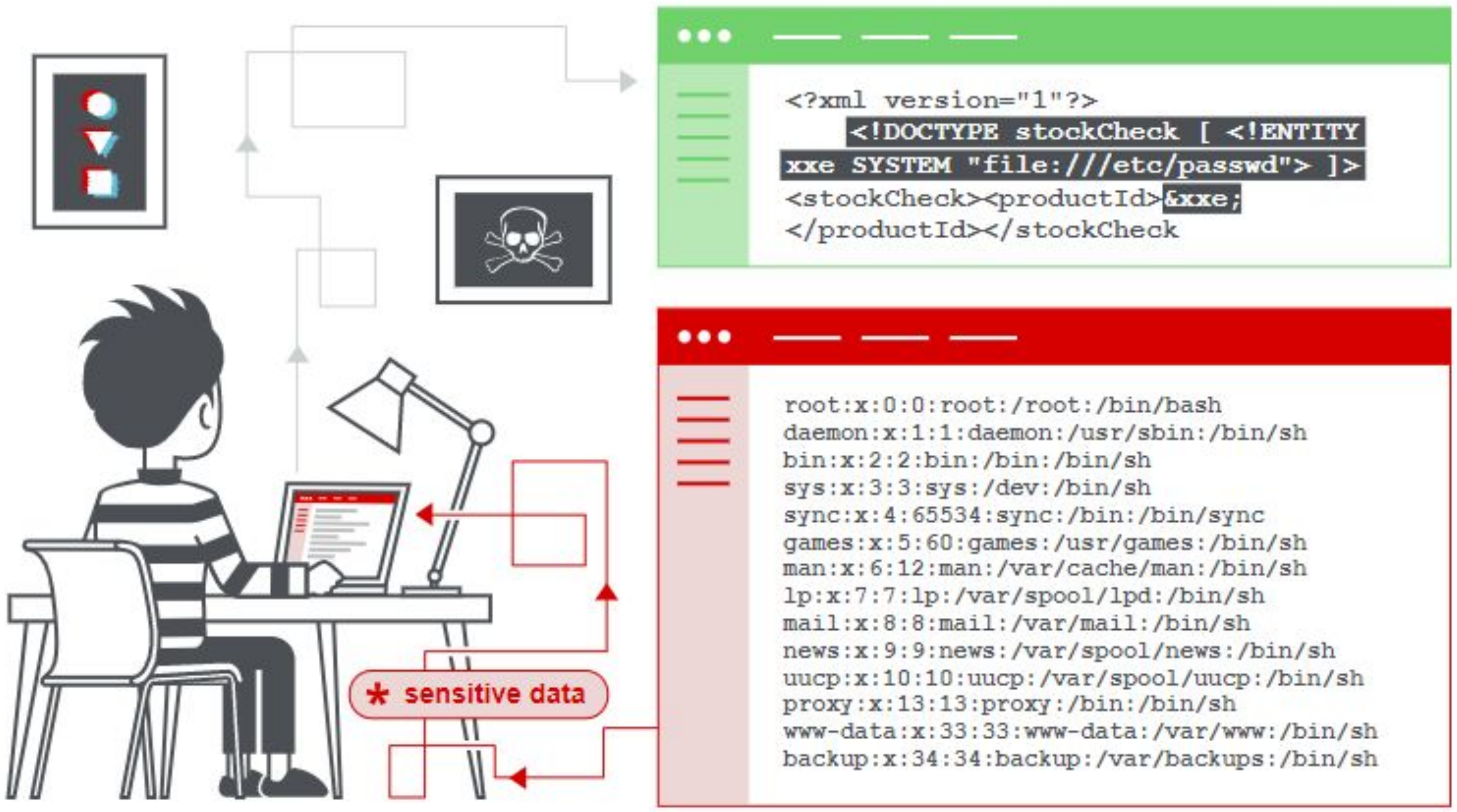
UNIT 4
Attack Landscape – Web Application Security

Subject Faculty – Ms. Hepi Suthar

External Entities

- XML external entity injection (also known as XXE) is a web security vulnerability that allows an attacker to interfere with an application's processing of XML data.
- It often allows an attacker to view files on the application server filesystem, and to interact with any back-end or external systems that the application itself can access.
- In some situations, an attacker can escalate an XXE attack to compromise the underlying server or other back-end infrastructure, by leveraging the XXE vulnerability to perform server-side request forgery (SSRF) attacks.

https://portswigger.net/web-security/xxe#:~:text=What%20is%20XML%20external%20entity,application's%20processing%20of%20XML%20data.



```
<?xml version="1"?>
    <!DOCTYPE stockCheck [ <!ENTITY
xxe SYSTEM "file:///etc/passwd"> ]>
<stockCheck><productId>&xxe;
</productId></stockCheck
```

```
root:x:0:0:root:/root:/bin/bash
daemon:x:1:1:daemon:/usr/sbin:/bin/sh
bin:x:2:2:bin:/bin:/bin/sh
sys:x:3:3:sys:/dev:/bin/sh
sync:x:4:65534:sync:/bin:/bin/sync
games:x:5:60:games:/usr/games:/bin/sh
man:x:6:12:man:/var/cache/man:/bin/sh
lp:x:7:7:lp:/var/spool/lpd:/bin/sh
mail:x:8:8:mail:/var/mail:/bin/sh
news:x:9:9:news:/var/spool/news:/bin/sh
uucp:x:10:10:uucp:/var/spool/uucp:/bin/sh
proxy:x:13:13:proxy:/bin:/bin/sh
www-data:x:33:33:www-data:/var/www:/bin/sh
backup:x:34:34:backup:/var/backups:/bin/sh
```

Broken Authentication Control

- Broken access control vulnerabilities exist **when a user can in fact access some resource or perform some action that they are not supposed to be able to access**.
- Access control, sometimes called authorization, is how a web application grants access to content and functions to some users and not others. These checks are performed after authentication, and govern what 'authorized' users are allowed to do.
- Access control sounds like a simple problem but is insidiously difficult to implement correctly. A web application's access control model is closely tied to the content and functions that the site provides. In addition, the users may fall into a number of groups or roles with different abilities or privileges.
- Developers frequently underestimate the difficulty of implementing a reliable access control mechanism.
- Many of these schemes were not deliberately designed, but have simply evolved along with the web site. In these cases, access control rules are inserted in various locations all over the code. As the site nears deployment, the ad-hoc collection of rules becomes so unwieldy that it is almost impossible to understand.

https://owasp.org/www-community/Broken_Access_Control

Security Misconfiguration

- The application might be vulnerable if the application is:
- Missing appropriate security hardening across any part of the application stack or improperly configured permissions on cloud services.
- Unnecessary features are enabled or installed (e.g., unnecessary ports, services, pages, accounts, or privileges).
- Default accounts and their passwords are still enabled and unchanged.
- Error handling reveals stack traces or other overly informative error messages to users.
- For upgraded systems, the latest security features are disabled or not configured securely.
- The security settings in the application servers, application frameworks (e.g., Struts, Spring, ASP.NET), libraries, databases, etc., are not set to secure values.
- The server does not send security headers or directives, or they are not set to secure values.
- The software is out of date or vulnerable (see A06:2021-Vulnerable and Outdated Components).

How to Prevent SM

- Secure installation processes should be implemented, including:
- A repeatable hardening process makes it fast and easy to deploy another environment that is appropriately locked down. Development, QA, and production environments should all be configured identically, with different credentials used in each environment. This process should be automated to minimize the effort required to set up a new secure environment.
- A minimal platform without any unnecessary features, components, documentation, and samples. Remove or do not install unused features and frameworks.
- A task to review and update the configurations appropriate to all security notes, updates, and patches as part of the patch management process (see A06:2021-Vulnerable and Outdated Components). Review cloud storage permissions (e.g., S3 bucket permissions).
- A segmented application architecture provides effective and secure separation between components or tenants, with segmentation, containerization, or cloud security groups (ACLs).
- Sending security directives to clients, e.g., Security Headers.
- An automated process to verify the effectiveness of the configurations and settings in all environments.

Cross site Scripting

- Cross-Site Scripting (XSS) attacks are a type of injection, in which malicious scripts are injected into otherwise benign and trusted websites. XSS attacks occur when an attacker uses a web application to send malicious code, generally in the form of a browser side script, to a different end user. Flaws that allow these attacks to succeed are quite widespread and occur anywhere a web application uses input from a user within the output it generates without validating or encoding it.
- An attacker can use XSS to send a malicious script to an unsuspecting user. The end user's browser has no way to know that the script should not be trusted, and will execute the script. Because it thinks the script came from a trusted source, the malicious script can access any cookies, session tokens, or other sensitive information retained by the browser and used with that site. These scripts can even rewrite the content of the HTML page.

When it Occur

- Data enters a Web application through an untrusted source, most frequently a web request.
- The data is included in dynamic content that is sent to a web user without being validated for malicious content.
- The malicious content sent to the web browser often takes the form of a segment of JavaScript, but may also include HTML, Flash, or any other type of code that the browser may execute.
- The variety of attacks based on XSS is almost limitless, but they commonly include transmitting private data, like cookies or other session information, to the attacker, redirecting the victim to web content controlled by the attacker, or performing other malicious operations on the user’s machine under the guise of the vulnerable site.

How to Protect Yourself

- The primary defenses against XSS are described in the OWASP XSS Prevention Cheat Sheet.
- Also, it’s crucial that you turn off HTTP TRACE support on all web servers. An attacker can steal cookie data via Java-script even when document.
- cookie is disabled or not supported by the client. This attack is mounted when a user posts a malicious script to a forum so when another user clicks the link, an asynchronous HTTP Trace call is triggered which collects the user’s cookie information from the server, and then sends it over to another malicious server that collects the cookie information so the attacker can mount a session hijack attack.
- This is easily mitigated by removing support for HTTP TRACE on all web servers.

Insecure Deserialization

- Insecure deserialization is a vulnerability in which an untrusted or unknown data is used to either inflict a denial of service attack (DoS attack), execute code, bypass authentication or further abuse the logic behind an application.
- Insecure deserialization is **when user-controllable data is deserialized by a website**.
- This potentially enables an attacker to manipulate serialized objects in order to pass harmful data into the application code.
- It is even possible to replace a serialized object with an object of an entirely different class.

https://portswigger.net/web-security/deserialization

Components with known Vulnerabilities

- **According to OWASP:** Using Components with Known Vulnerabilities. Components such as libraries, frameworks, and other software modules run with the same privileges as the application. If a vulnerable component is exploited, an attack can facilitate severe data loss or server takeover.
- These attacks have become commonplace because it is far easier for an attacker to use a known weakness than create a specific attack method to search out vulnerabilities themselves. This fact should put known component vulnerabilities high on your security priority list to mitigate.

Examples of using components with known vulnerabilities

- The following example demonstrates how you might inadvertently start using a component with a known vulnerability:
- You are building a website on WordPress and require a template and various plugins (tools) to create your fully functioning website. The plugins & templates include those made and provided by other users. These could have their own vulnerability issues, and using the plugin compromises your website's security.
- WordPress creates general updates regularly to tackle security issues. However, due to its popularity, it can still be a target.
- The next example demonstrates the potential risks and the magnitude of an attack facilitated by using a component with a known vulnerability:
- The Equifax breach allowed an attacker to gain remote code execution and pivot inside the Equifax network, allowing the attacker to steal the personal information of more than 140 million customers. The entry point was a vulnerability in a version of Struts (CVE-2017-5638)

How to prevent using components with a known vulnerability

- The best way to prevent using components with known vulnerabilities would be to never use third-party components.
- However, this is not always possible in the real world. This means that you need to take precautions when choosing which 3rd party tools to use or work with.
- Always upgrade components to the latest version (patching)
- Use penetration testing. It is best to use an Iasme certified and CREST accredited penetration testing company to identify cybersecurity vulnerabilities such as insecure deserialisation or insufficient logging and monitoring.

Insufficient Logging & Monitoring

- https://owasp.org/www-project-top-ten/2017/A10_2017-Insufficient_Logging%2526Monitoring

Cross site Scripting Forgery

- Cross-site request forgery (also known as CSRF) is a web security vulnerability that allows an attacker to induce users to perform actions that they do not intend to perform.
- It allows an attacker to partly circumvent the same origin policy, which is designed to prevent different websites from interfering with each other.

- https://portswigger.net/web-security/csrf

File Inclusion

- Local file inclusion (also known as LFI) is the process of including files, that are already locally present on the server, through the exploiting of vulnerable inclusion procedures implemented in the application.
- This vulnerability occurs, for example, when a page receives, as input, the path to the file that has to be included and this input is not properly sanitized, allowing directory traversal characters (such as dot-dot-slash) to be injected.
- Although most examples point to vulnerable PHP scripts, we should keep in mind that it is also common in other technologies such as JSP, ASP and others.

https://owasp.org/www-project-web-security-testing-guide/v42/4-Web_Application_Security_Testing/07-Input_Validation_Testing/11.1-Testing_for_Local_File_Inclusion#:~:text=Local%20file%20inclusion%20(also%20known,procedures%20implemented%20in%20the%20application.

Click Jacking

- Clickjacking is an attack that tricks a user into clicking a webpage element which is invisible or disguised as another element. This can cause users to unwittingly download malware, visit malicious web pages, provide credentials or sensitive information, transfer money, or purchase products online.
- Typically, clickjacking is performed by displaying an invisible page or HTML element, inside an iframe, on top of the page the user sees. The user believes they are clicking the visible page but in fact they are clicking an invisible element in the additional page transposed on top of it.
- The invisible page could be a malicious page, or a legitimate page the user did not intend to visit – for example, a page on the user's banking site that authorizes the transfer of money.

https://www.imperva.com/learn/application-security/clickjacking/

File Inclusion / File Upload

- https://www.offensive-security.com/metasploit-unleashed/file-inclusion-vulnerabilities/#:~:text=Remote%20File%20Inclusion%20(RFI)%20and,upload%20files%20to%20the%20server.

Insecure Captcha

- https://christovitohidajat.medium.com/insecure-captcha-dvwa-hacking-for-unsafe-verification-process-8d0d938eb9e9

SSRF / XSPA

- SSRF flaws occur whenever a web application is fetching a remote resource without validating the user-supplied URL.
- It allows an attacker to coerce the application to send a crafted request to an unexpected destination, even when protected by a firewall, VPN, or another type of network access control list (ACL).
- As modern web applications provide end-users with convenient features, fetching a URL becomes a common scenario.
- As a result, the incidence of SSRF is increasing. Also, the severity of SSRF is becoming higher due to cloud services and the complexity of architectures.

https://owasp.org/Top10/A10_2021-Server-Side_Request_Forgery_%28SSRF%29/

How to Prevent SSRF

- Developers can prevent SSRF by implementing some or all the following defense in depth controls:
- **From Network layer**
- Segment remote resource access functionality in separate networks to reduce the impact of SSRF
- Enforce "deny by default" firewall policies or network access control rules to block all but essential intranet traffic.
  ~ Establish an ownership and a lifecycle for firewall rules based on applications.
  ~ Log all accepted *and* blocked network flows on firewalls (see A09:2021-Security Logging and Monitoring Failures).
- **From Application layer:**
- Sanitize and validate all client-supplied input data
- Enforce the URL schema, port, and destination with a positive allow list
- Do not send raw responses to clients
- Disable HTTP redirections
- Be aware of the URL consistency to avoid attacks such as DNS rebinding and "time of check, time of use" (TOCTOU) race conditions
- Do not mitigate SSRF via the use of a deny list or regular expression. Attackers have payload lists, tools, and skills to bypass deny lists.
- **Additional Measures to consider:**
- Don't deploy other security relevant services on front systems (e.g. OpenID). Control local traffic on these systems (e.g. localhost)
- For frontends with dedicated and manageable user groups use network encryption (e.g. VPNs) on independent systems to consider very high protection needs.