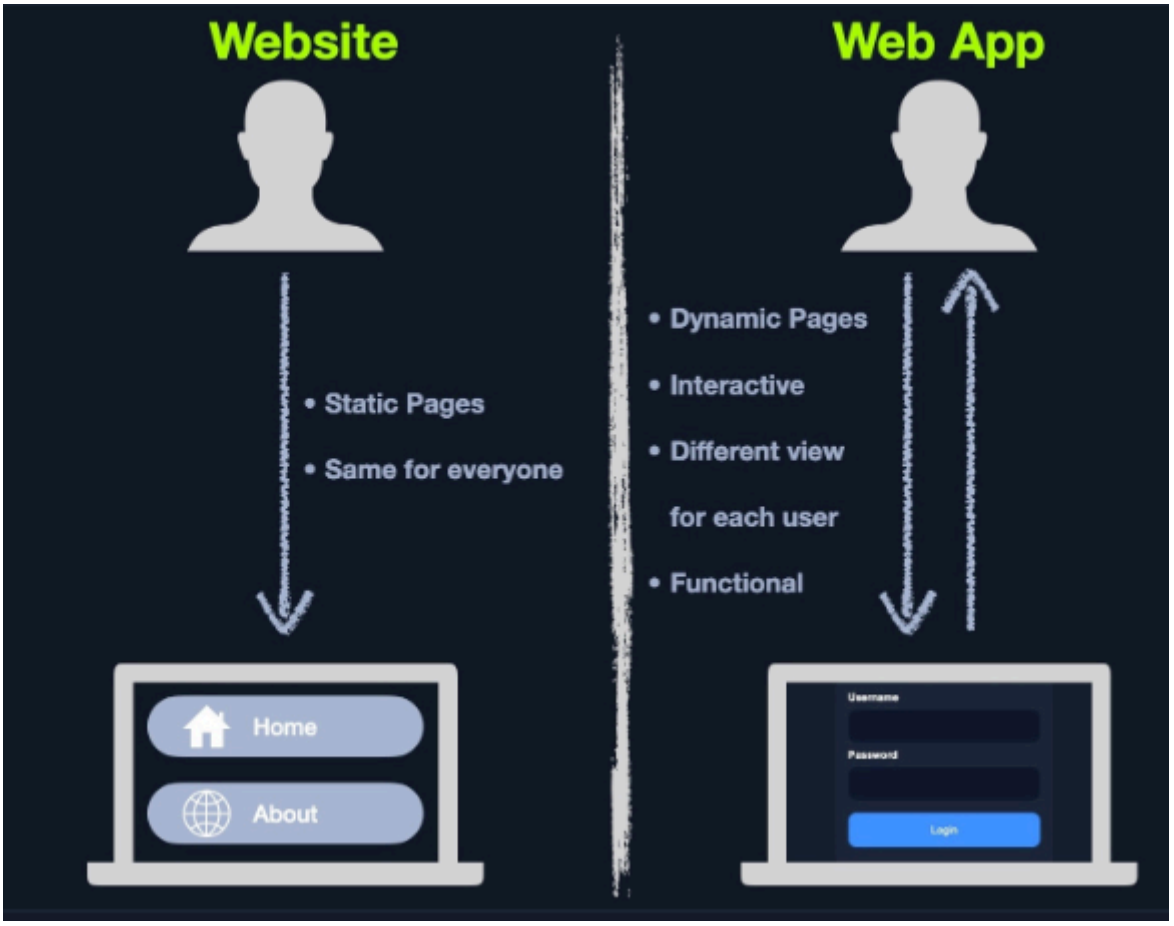
Introduction

Web Applications vs. Websites

AL principal diferencia se encuentra en que las Websites (Web 1.0) son estáticas, esto quiere decir que siempre se cargan los mismo recursos, y no proporcionan al cliente un servicio personalizado, sin embargo las Web Applications (Web 2.0) son dinámicas, esto quiere decir que son funcionales en el sentido que el usuario puede interactuar con ellas:



Web Application Distribution

Algunas de las más conocidas son:

(Open Source)

- WordPress
- OpenCart
- Joomla

(Closed Source)

- Wix
- Shopify
- DotNetNuke

Attacking Web Applications

Algunos de los posibles Ataques son:

Flaw	Real-world Scenario
SQL injection	Obtaining Active Directory usernames and performing a password spraying attack against a VPN or email portal.
File Inclusion	Reading source code to find a hidden page or directory which exposes additional functionality that can be used to gain remote code execution.
Unrestricted File Upload	A web application that allows a user to upload a profile picture that allows any file type to be uploaded (not just images). This can be leveraged to gain full control of the web application server by uploading malicious code.
Insecure Direct Object Referencing (IDOR)	When combined with a flaw such as broken access control, this can often be used to access another user's files or functionality. An example would be editing your user profile browsing to a page such as /user/701/edit-profile. If we can change the `701` to `702`, we may edit another user's profile!
Broken Access Control	Another example is an application that allows a user to register a new account. If the account registration functionality is designed poorly, a user may perform privilege escalation when registering. Consider the `POST` request when registering a new user, which submits the data `username=bjones&password=Welcome1&email=bjones@inlanefreight.local&roleid=3`. What if we can manipulate the `roleid` parameter and change it to `0` or `1`. We have seen real-world applications where this was the case, and it was possible to quickly register an admin user and access many unintended features of the web application.

Web Application Layout

Los diseños de aplicaciones web constan de muchas capas diferentes que pueden resumirse en las siguientes tres categorías principales:

Category	Description
`Web Application Infrastructure`	Describes the structure of required components, such as the database, needed for the web application to function as intended. Since the web application can be set up to run on a separate server, it is essential to know which database server it needs to access.
`Web Application Components`	The components that make up a web application represent all the components that the web application interacts with. These are divided into the following three areas: `UI/UX`, `Client`, and `Server` components.
`Web Application Architecture`	Architecture comprises all the relationships between the various web application components.

Web Application Infrastructure

Los diferentes modelos son:

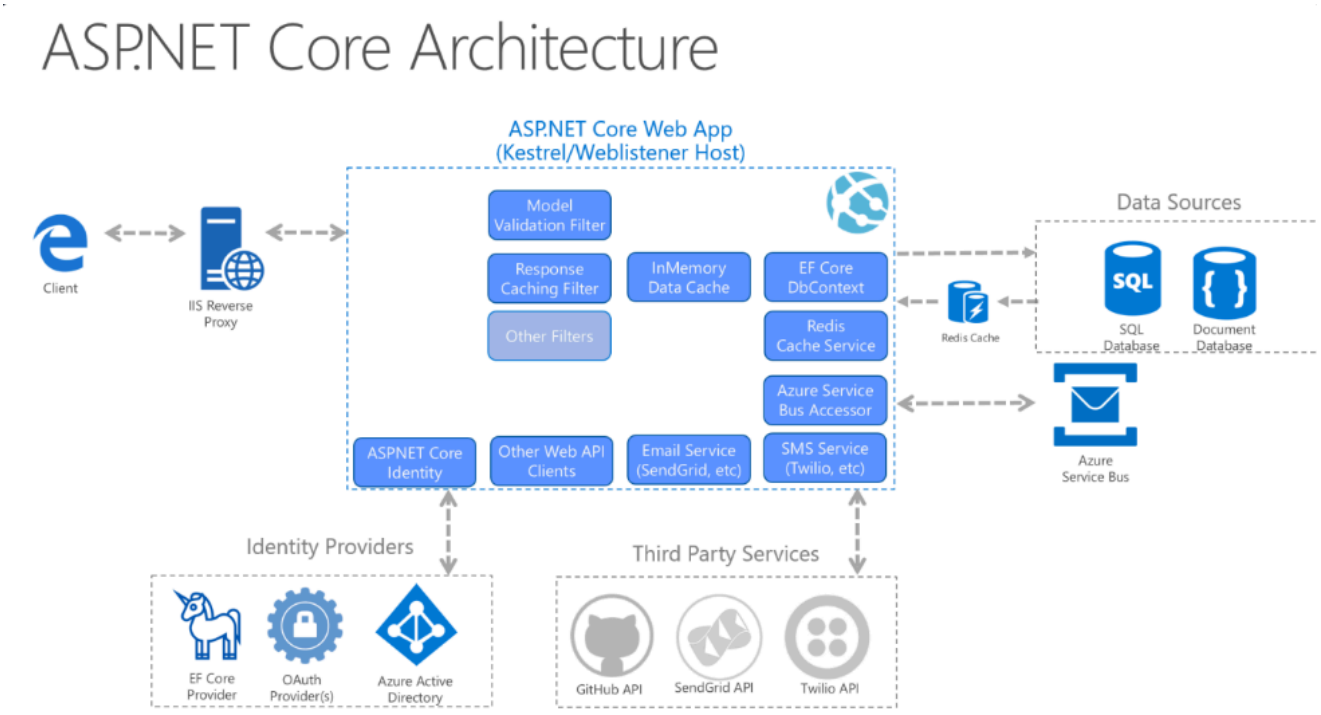
- Client-Server (la más común)
- One Server (La más peligrosa)
- Many Servers - One Database
- Many Servers - Many Databases (la opción más segura)

Web Application Components

1. Client
2. Server
    - Webserver
    - Web Application Logic
    - Database
3. Services (Microservices)
    - 3rd Party Integrations
    - Web Application Integrations
4. Functions (Serverless)

Web Application Architecture

Un ejemplo de como se vería:



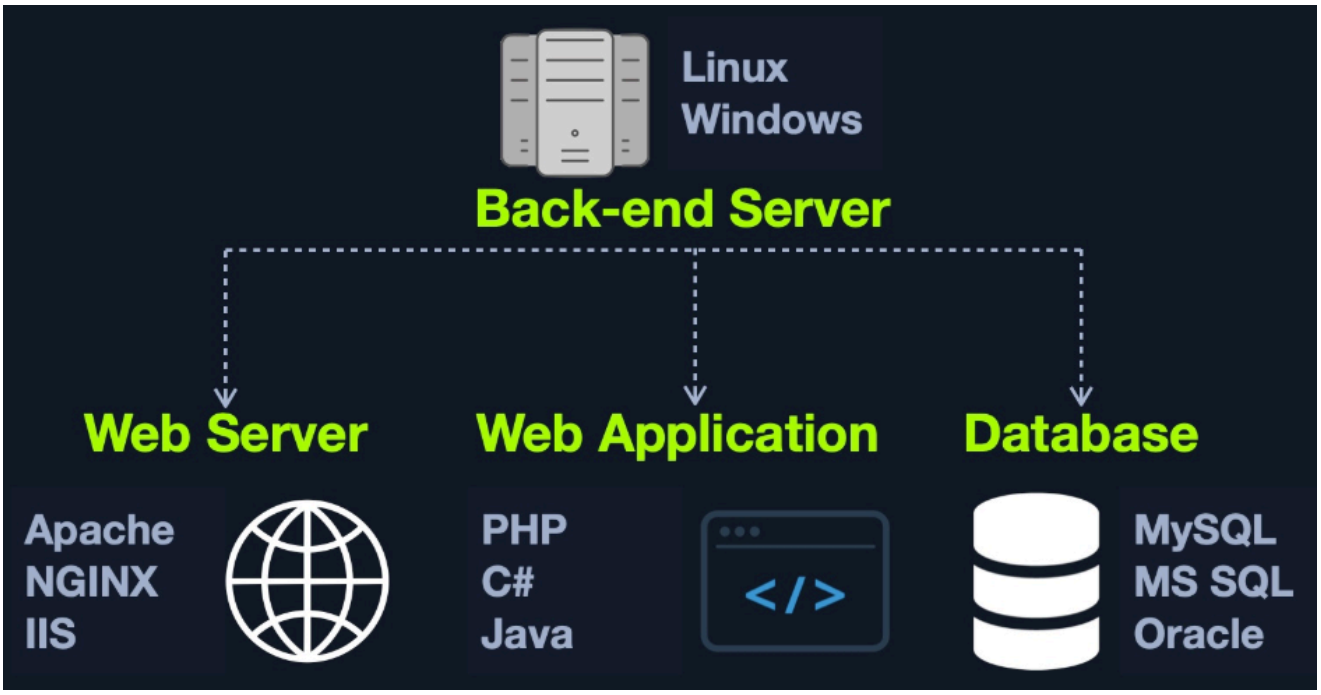
Front End vs. Back End

Front End

HTM, CSS, Java Script son los tres agentes principales de que la página se vea por el usuario , y de como se vea esta misma.

Back End

Component	Description
Back end Servers	The hardware and operating system that hosts all other components and are usually run on operating systems like Linux, Windows, or using Containers.
Web Servers	Web servers handle HTTP requests and connections. Some examples are Apache, NGINX, and IIS.
Databases	Databases (DBs) store and retrieve the web application data. Some examples of relational databases are MySQL, MSSQL, Oracle, PostgreSQL, while examples of non-relational databases include NoSQL and MongoDB.
Development Frameworks	Development Frameworks are used to develop the core Web Application. Some well-known frameworks include Laravel (PHP), ASP.NET (C#), Spring (Java), Django (Python), and Express (NodeJS JavaScript).



Sensitive Data Exposure

La exposición de datos sensibles se refiere a la disponibilidad de datos sensibles en texto plano para el usuario final. Esto suele encontrarse en el código fuente de la página web o en el código fuente de la página en el front-end de las aplicaciones web. Este es el código fuente HTML de la aplicación, que no debe confundirse con el código back-end, que normalmente solo es accesible en el servidor. Podemos ver el código fuente de cualquier sitio web en nuestro navegador haciendo clic derecho en cualquier parte de la página y seleccionando "Ver código fuente" en el menú emergente. En ocasiones, un desarrollador puede desactivar el clic derecho en una aplicación web, pero esto no nos impide ver el código fuente, ya que podemos simplemente presionar **Ctrl + U** o verlo a través de un proxy web como Burp Suite.

Algunas veces (pocas) vemos credenciales en el front-end:

```
<form action="action_page.php" method="post">

    <div class="container">
        <label for="uname"><b>Username</b></label>
        <input type="text" required>

        <label for="psw"><b>Password</b></label>
        <input type="password" required>

        <!-- TODO: remove test credentials test:test -->

        <button type="submit">Login</button>
    </div>
</form>

</html>
```

HTML Injection

Cuando un usuario tiene control total sobre cómo se mostrará su entrada, puede enviar código HTML y el navegador podría mostrarlo como parte de la página. Esto puede incluir código HTML malicioso, como un formulario de inicio de sesión externo, que puede usarse para engañar a los usuarios para que inicien sesión, mientras que en realidad envía sus credenciales de inicio de sesión a un servidor malicioso para su recopilación con fines de otros ataques.

Podemos observar que esta entrada no está sanitizada:

```html
<!DOCTYPE html>
<html>

<body>
    <button onclick="inputFunction()">Click to enter your name</button>
    <p id="output"></p>

    <script>
        function inputFunction() {
            var input = prompt("Please enter your name", "");

            if (input != null) {
                document.getElementById("output").innerHTML = "Your name is " + input;
            }
        }
    </script>
</body>

</html>
```

Por lo que podríamos utilizar esa entra a nuestro antojo, inyectando HTM que nos permita inyectar código malicioso.

Cross-Site Scripting (XSS)

Este ataque se basa en la inyección de código Java Script en el lado del cliente.

Type	Description
Reflected XSS	Occurs when user input is displayed on the page after processing (e.g., search result or error message).
Stored XSS	Occurs when user input is stored in the back end database and then displayed upon retrieval (e.g., posts or comments).
DOM XSS	Occurs when user input is directly shown in the browser and is written to an HTML DOM object (e.g., vulnerable username or page title).

Con payloads como el suiguiente podemos obtener cookies de sesión:

```
#"><img src=/ onerror=alert(document.cookie)>
```

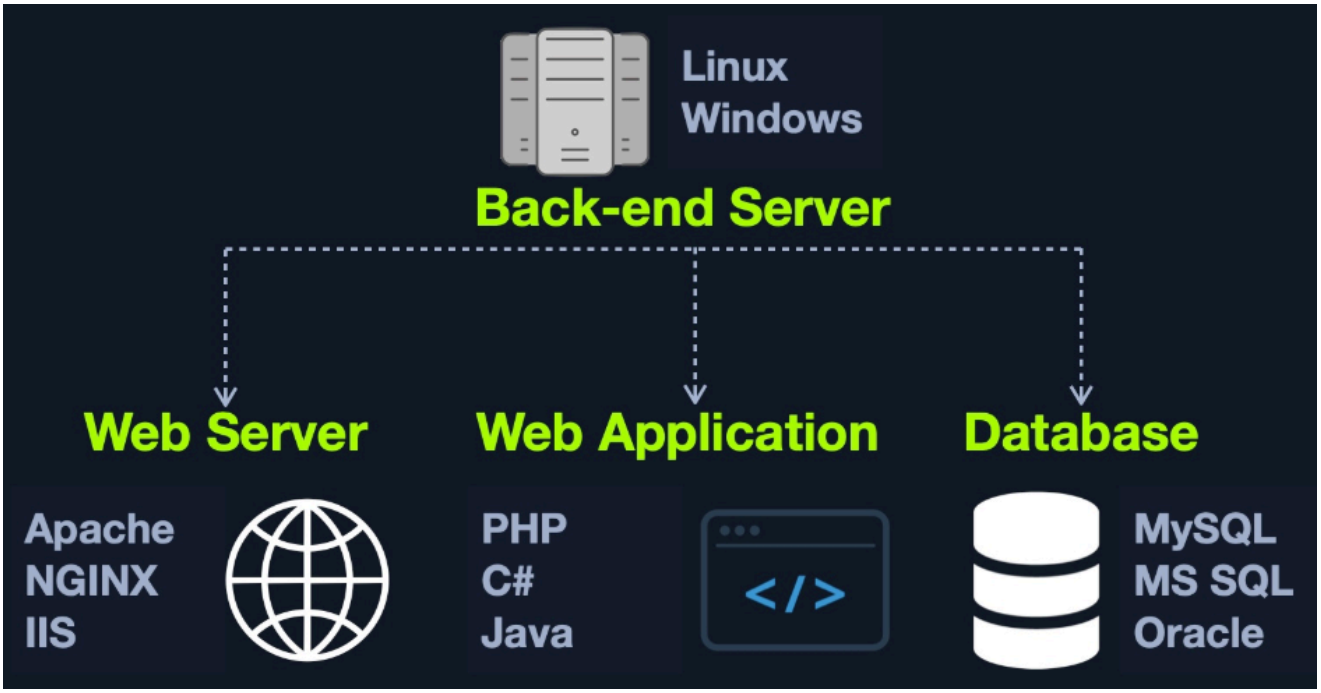
Prevention

Type	Description
Sanitization	Removing special characters and non-standard characters from user input before displaying it or storing it.
Validation	Ensuring that submitted user input matches the expected format (i.e., submitted email matched email format)

Back End Servers

Un servidor back-end es el hardware y el sistema operativo que aloja todas las aplicaciones necesarias para ejecutar la aplicación web. Es el sistema real que ejecuta todos los procesos y realiza todas las tareas que conforman la aplicación web. El servidor back-end se ubicaría en la capa de acceso a datos.

Combinations	Components
LAMP	Linux, Apache, MySQL, and PHP.
WAMP	Windows, Apache, MySQL, and PHP.
WINS	Windows, IIS, .NET, and SQL Server
MAMP	macOS, Apache, MySQL, and PHP.
XAMPP	Cross-Platform, Apache, MySQL, and PHP/PERL.



Web Servers

Un servidor web es una aplicación que se ejecuta en el servidor backend y gestiona todo el tráfico HTTP del navegador del cliente, lo enruta a las páginas solicitadas y, finalmente, responde al navegador del cliente. Los servidores web suelen ejecutarse en los puertos TCP 80 o 443 y son responsables de conectar a los usuarios finales con las distintas partes de la aplicación web, además de gestionar sus diversas respuestas.

Algunos de los ejemplos son AACHE, NGINX, IIS.

Databases

Las aplicaciones web utilizan bases de datos back-end para almacenar contenido e información relacionados con ellas. Estos pueden ser recursos esenciales de la aplicación, como imágenes y archivos; contenido de la aplicación, como publicaciones y actualizaciones; o datos de usuario, como nombres de usuario y contraseñas. Esto permite que las aplicaciones web almacenen y recuperen datos de forma fácil y rápida, y habilitan contenido dinámico diferente para cada usuario.

Development Frameworks & APIs

Dado que la mayoría de las aplicaciones web comparten una funcionalidad común, como el registro de usuarios, los frameworks de desarrollo web facilitan la rápida implementación de esta funcionalidad y su vinculación con los componentes front-end, creando una aplicación web completamente funcional. Algunos de los frameworks de desarrollo web más comunes incluyen:

```
• Laravel (PHP): usually used by startups and smaller companies, as it is powerful
  yet easy to develop for.
• Express (Node.JS): used by PayPal, Yahoo, Uber, IBM, and MySpace.
• Django (Python): used by Google, YouTube, Instagram, Mozilla, and Pinterest.
• Rails (Ruby): used by GitHub, Hulu, Twitch, Airbnb, and even Twitter in the past.
```

APIs

Un aspecto importante del desarrollo de aplicaciones web back end es el uso de API web y parámetros de solicitud HTTP para conectar el front end y el back end para poder enviar datos de ida y vuelta entre los componentes front end y back end y llevar a cabo diversas funciones dentro de la aplicación web.

Web APIs

Una API (Interfaz de Programación de Aplicaciones) es una interfaz dentro de una aplicación que especifica cómo esta puede interactuar con otras aplicaciones. En el caso de las aplicaciones web, permite el acceso remoto a la funcionalidad de los componentes del backend.

SOAP

El estándar SOAP (Acceso Simple a Objetos) comparte datos mediante XML. La solicitud se realiza en XML mediante una solicitud HTTP y la respuesta también se devuelve en XML. Los componentes front-end están diseñados para analizar correctamente esta salida XML.

REST

El estándar REST (Transferencia de Estado Representacional) comparte datos a través de la ruta URL, por ejemplo, `search/users/1`, y generalmente devuelve la salida en formato JSON, por ejemplo, `userid 1`. A diferencia de los parámetros de consulta, las API REST suelen centrarse en páginas que esperan un tipo de entrada que se pasa directamente a través de la ruta URL, sin especificar su nombre ni tipo. Esto suele ser útil para consultas como buscar, ordenar o filtrar. Por ello, las API REST suelen dividir la funcionalidad de las aplicaciones web en API más pequeñas y utilizar estas solicitudes para permitir que la aplicación web realice acciones más avanzadas, lo que la hace más modular y escalable.

Common Web Vulnerabilities

Broken Authentication/Access Control

Broken Authentication se refiere a vulnerabilidades que permiten a los atacantes eludir las funciones de autenticación. Por ejemplo, esto podría permitir que un atacante inicie sesión sin credenciales válidas o que un usuario normal se convierta en administrador sin los privilegios necesarios.

Broken Access Control se refiere a vulnerabilidades que permiten a los atacantes acceder a páginas y funciones a las que no deberían tener acceso. Por ejemplo, un usuario normal podría acceder al panel de administración.

Malicious File Upload

Si la aplicación web tiene una función de carga de archivos y no valida correctamente los archivos cargados, podemos cargar un script malicioso (es decir, un script PHP), que nos permitirá ejecutar comandos en el servidor remoto.

Por ejemplo, el plugin de WordPress **Responsive Thumbnail Slider 1.0** puede explotarse para cargar cualquier archivo arbitrario, incluyendo scripts maliciosos, subiendo un archivo con doble extensión (p. ej., shell.php.jpg). Incluso existe un módulo de Metasploit que permite explotar esta vulnerabilidad fácilmente.

Command Injection

Si no se filtra y sanitiza adecuadamente, los atacantes podrían inyectar otro comando que se ejecute junto con el comando original (es decir, como el nombre del complemento), lo que les permite ejecutar comandos directamente en el servidor backend y obtener control sobre él.

Por ejemplo, el complemento de WordPress Plainview Activity Monitor 20161228 tiene una vulnerabilidad que permite a los atacantes inyectar su comando en el valor de IP, simplemente agregando | COMMAND... después del valor de IP.

SQL Injection (SQLi)

Otra vulnerabilidad muy común en las aplicaciones web es la inyección SQL. Al igual que la inyección de comandos, esta vulnerabilidad puede ocurrir cuando la aplicación web ejecuta una consulta SQL que incluye un valor obtenido de la entrada del usuario.

Public Vulnerabilities

Dado que muchas organizaciones implementan aplicaciones web de uso público, como aplicaciones web de código abierto y propietarias, estas suelen ser probadas por numerosas organizaciones y expertos de todo el mundo. Esto lleva al descubrimiento frecuente de un gran número de vulnerabilidades, la mayoría de las cuales se parchean y se comparten públicamente, asignándoseles un registro y una puntuación CVE (vulnerabilidades y exposiciones comunes).

Base Score Metrics

Exploitability Metrics

Attack Vector (AV)*

Network (AV:N) | Adjacent Network (AV:A) | Local (AV:L) | Physical (AV:P)

Attack Complexity (AC)*

Low (AC:L) | High (AC:H)

Privileges Required (PR)*

None (PR:N) | Low (PR:L) | High (PR:H)

User Interaction (UI)*

None (UI:N) | Required (UI:R)

Scope (S)*

Unchanged (S:U) | Changed (S:C)

Impact Metrics

Confidentiality Impact (C)*

None (C:N) | Low (C:L) | High (C:H)

Integrity Impact (I)*

None (I:N) | Low (I:L) | High (I:H)

Availability Impact (A)*

None (A:N) | Low (A:L) | High (A:H)

* - All base metrics are required to generate a base score.

Temporal Score Metrics

Exploit Code Maturity (E)

Not Defined (E:X) | Unproven that exploit exists (E:U) | Proof of concept code (E:P) | Functional exploit exists (E:F) | High (E:H)

Remediation Level (RL)

Not Defined (RL:X) | Official fix (RL:O) | Temporary fix (RL:T) | Workaround (RL:W) | Unavailable (RL:U)

Report Confidence (RC)

Not Defined (RC:X) | Unknown (RC:U) | Reasonable (RC:R) | Confirmed (RC:C)

Environmental Score Metrics

Exploitability Metrics

Attack Vector (MAV)

Not Defined (MAV:X) | Network (MAV:N) | Adjacent Network (MAV:A) | Local (MAV:L) | Physical (MAV:P)

Attack Complexity (MAC)

Not Defined (MAC:X) | Low (MAC:L) | High (MAC:H)

Privileges Required (MPR)

Not Defined (MPR:X) | None (MPR:N) | Low (MPR:L) | High (MPR:H)

User Interaction (MUI)

Not Defined (MUI:X) | None (MUI:N) | Required (MUI:R)

Scope (MS)

Not Defined (MS:X) | Unchanged (MS:U) | Changed (MS:C)

Impact Metrics

Confidentiality Impact (MC)

Not Defined (MC:X) | None (MC:N) | Low (MC:L) | High (MC:H)

Integrity Impact (MI)

Not Defined (MI:X) | None (MI:N) | Low (MI:L) | High (MI:H)

Availability Impact (MA)

Not Defined (MA:X) | None (MA:N) | Low (MA:L) | High (MA:H)

Impact Subscore Modifiers

Confidentiality Requirement (CR)

Not Defined (CR:X) | Low (CR:L) | Medium (CR:M) | High (CR:H)

Integrity Requirement (IR)

Not Defined (IR:X) | Low (IR:L) | Medium (IR:M) | High (IR:H)

Availability Requirement (AR)

Not Defined (AR:X) | Low (AR:L) | Medium (AR:M) | High (AR:H)