|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **MATERIA** | Desarrollo Web Avanzado | **NRC** | 2296 | **TRABAJO No.** | 2 |
| **CARRERA** | Ing. de Software | **DOCENTE** | | Edison Lascano | |
| **PERIODO ACADÉMICO** | OCT 24 - MAR 25 | **FECHA** | | 23/12/2024 | |
| **TEMA** | Investigación de RESTful | | | | |
| **ESTUDIANTE(S)** | Matías Suárez | | | | |

|  |
| --- |
| **A.** **INTRODUCCION** |

En la creación de aplicaciones web contemporáneas, la seguridad y la eficacia son elementos clave. La urgencia de salvaguardar la información del usuario y asegurar una comunicación fiable entre los sistemas ha impulsado la implementación de herramientas y métodos como OAuth, Credentials de Seguridad y RESTful. Estas tecnologías no solo potencian la experiencia del usuario, sino que también brindan a los programadores la posibilidad de desarrollar aplicaciones más escalables, seguras y de fácil integración. Este reporte se enfoca en la aplicación de estas tecnologías en el ámbito de las aplicaciones web, resaltando sus características principales, beneficios y vínculos.

La seguridad cobra particular importancia en un contexto donde las vulnerabilidades de datos y los ataques informáticos se vuelven cada vez más habituales. OAuth responde a la exigencia de regular el acceso a los recursos del usuario sin poner en riesgo su privacidad. Las Credentials de Seguridad actúan como fundamento para la autenticación y autorización, mientras que RESTful ofrece un marco eficaz y adaptable para la comunicación entre sistemas. Entender estas herramientas y su incorporación es fundamental para desarrollar aplicaciones web sólidas y fiables.

|  |
| --- |
| **B.** **DESARROLLO** |

**OAuth**

OAuth (Authorization Open) es una norma que facilita a las aplicaciones el acceso seguro a recursos en representación de un usuario, sin la necesidad de compartir sus credenciales. La versión más empleada es OAuth 2.0, la cual se fundamenta en la utilización de tokens de acceso para validar peticiones.

**Especificaciones de OAuth**

**División de obligaciones:** OAuth posibilita que el proveedor de identidad (como Google) administre las credenciales del usuario, mientras que la aplicación solo utiliza los recursos requeridos.

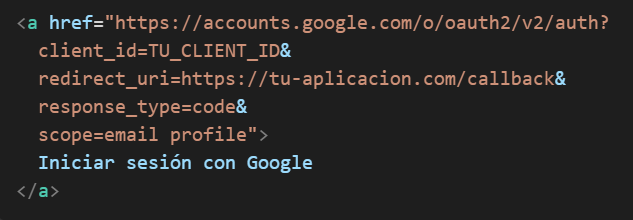
**Implementación de tokens de acceso:** Estos tokens tienen una duración limitada y restringen el alcance del acceso, lo que disminuye el efecto si se comprometen.

**Flujos a medida:** OAuth respalda diversos protocolos de autorización, tales como el Código de Autorización, Implicit y Credentials del Cliente, en función de las necesidades de la aplicación.

**Ejemplo de Implementación de OAuth 2.0**

Imaginemos que hemos creado una aplicación web que facilita a los usuarios la autenticación a través de Google. Para la implementación de OAuth 2.0, el método habitual es el de "Authorization Code".

1. **Solicitud de autorización:**

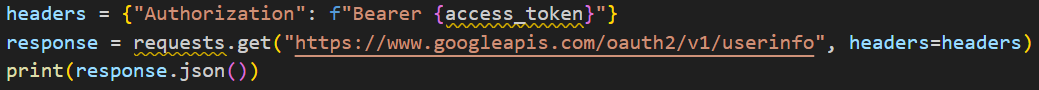


“Esto redirige al usuario a la página de autorización de Google.”

1. **Código intercambiado por token de acceso:**

En el backend, se efectúa la petición siguiente para conseguir el token:



1. **Utilización del token para obtener información del usuario:**

**Security Credentials**

Es fundamental contar con credenciales de seguridad para asegurar la autenticación y autorización en las aplicaciones. Estos comprenden claves de acceso, claves API, certificados y tokens.

**Tipos de Credenciales de Seguridad**

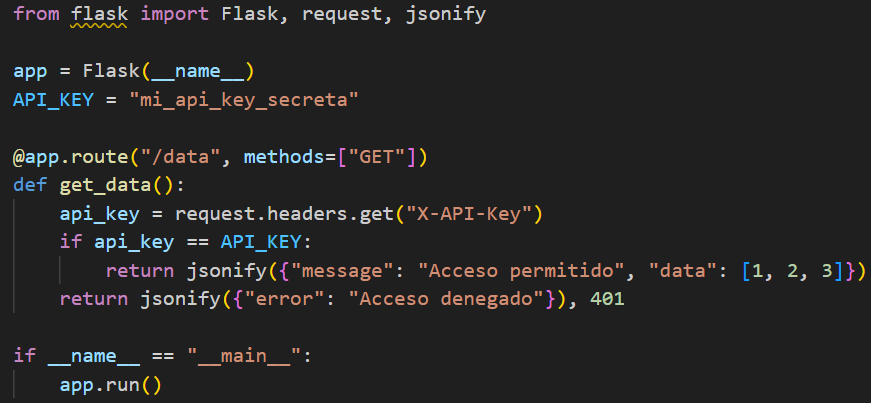
1. **Contraseñas:** Empleadas para verificar la autenticidad de los usuarios humanos.
2. **Claves API:** Creadas para verificar la autenticidad de las aplicaciones del cliente al relacionarse con APIs.
3. **Certificados:** Ofrecen autenticación cifrada para comunicaciones seguras.
4. **Tokens:** Facilitan un acceso limitado y temporal a recursos determinados.

**Ejemplo de Uso de Claves API**

Una API que requiere una clave para verificar peticiones puede ser configurada de la siguiente manera:

1. **Establecimiento de la API:**

El servidor corrobora la clave suministrada:



1. **Solicitud del cliente:**

****

Esto devolverá los datos solo si la clave proporcionada es válida.

**RESTful**

RESTful es una arquitectura orientada a la creación de servicios web. Establece fundamentos como la utilización de operaciones HTTP estándar y un diseño libre de estado.

**Principios de RESTful**

1. **Reconocimiento de recursos:** Cada recurso posee un identificador singular que se muestra por medio de una URL.
2. **Operaciones normalizadas:** Implementación de protocolos HTTP tales como GET, POST, PUT, DELETE para interactuar con los recursos.
3. **Sin condición:** Cada petición incluye toda la información requerida para su tratamiento.
4. **Diversas representaciones:** Los recursos se pueden mostrar en variados formatos como JSON, XML, entre otros.

**Ejemplo de Implementación de una API RESTful**

Una API básica para gestionar tareas puede ser implementada con Flask en Python:

1. **Definición del servidor RESTful:**

**Texto

Descripción generada automáticamente**

|  |
| --- |
| **C.** **CONCLUCION** |

1. La incorporación de OAuth, Credentials de Seguridad y RESTful en la creación de aplicaciones web asegura un balance entre la seguridad y la usabilidad. OAuth simplifica la autenticación delegada y salvaguarda la privacidad del usuario a través de la utilización de tokens de acceso.
2. Las claves de seguridad constituyen el núcleo de la autenticación y autorización, ofreciendo variados instrumentos como claves API y certificados, ajustados a diferentes contextos de uso, lo que facilita una administración de accesos más sólida y regulada.
3. RESTful proporciona un método arquitectónico eficaz para la creación de APIs que no solo son escalables y adaptables, sino que también son sencillas de incorporar en ambientes diversos. Además de OAuth y Credentials de Seguridad, se consigue un ecosistema seguro y operativo para aplicaciones contemporáneas.

|  |
| --- |
| **D.** **BIBLIOGRAFIA** |

Fotiou, N., Siris, V. A., & Polyzos, G. C. (2021). Capability-based access control for multi-tenant systems using OAuth 2.0 and Verifiable Credentials. arXiv preprint arXiv:2104.11515. Recuperado de <https://arxiv.org/abs/2104.11515>

Al Rahat, T., Feng, Y., & Tian, Y. (2021). Cerberus: Query-driven Scalable Vulnerability Detection in OAuth Service Provider Implementations. arXiv preprint arXiv:2110.01005. Recuperado de <https://arxiv.org/abs/2110.01005>

Autor(es) desconocido(s). (2023). Security measures implemented in RESTful API Development. ResearchGate. Recuperado de <https://www.researchgate.net/publication/384461158_Security_measures_implemented_in_RESTful_API_Development>

Parecki, A. (2022, 22 de diciembre). The complete guide to protecting your APIs with OAuth2 (part 1). Stack Overflow Blog. Recuperado de <https://stackoverflow.blog/2022/12/22/the-complete-guide-to-protecting-your-apis-with-oauth2/>

Google Developers. (s.f.). Using OAuth 2.0 for Web Server Applications. Google Identity. Recuperado de <https://developers.google.com/identity/protocols/oauth2/web-server>

HSC. (s.f.). Securing RESTful Web Services using Spring and OAuth 2.0. HSC Blog. Recuperado de <https://www.hsc.com/resources/blog/securing-restful-web-services-using-spring-and-oauth-2-0/>

OAuth.net. (s.f.). OAuth 2.0. Recuperado de <https://oauth.net/2/>

DreamFactory. (s.f.). How to Secure REST APIs: API Keys Vs OAuth. DreamFactory Blog. Recuperado de <https://blog.dreamfactory.com/how-to-secure-rest-apis-api-keys-vs-oauth>

Google Developers. (s.f.). Using OAuth 2.0 to Access Google APIs. Google Identity. Recuperado de <https://developers.google.com/identity/protocols/oauth2>