

PROGRAMMAZIONE ORIENTATA AI MICROSERVIZI

Sicurezza, autenticazione e autorizzazione

Tommaso Nanu

tommaso.nanu@alad.cloud



UNIVERSITÀ DI PARMA

DIPARTIMENTO DI SCIENZE MATEMATICHE, FISICHE E INFORMATICHE

Corso di Laurea in Informatica

Presentazione di Fabio Iotti

fabio.iotti@alad.cloud

Sicurezza nei web-service

In ASP.NET Core

Argomenti

- Tipi di API esposte in web-services
- Possibili attacchi verso i web-services
- Come proteggere le password degli utenti
- Meccanismi standard di autenticazione
- Autenticazione in ASP.NET Core

Web-services

- Servizi web espongono API
- API pubbliche (usufruibili liberamente da servizi esterni)
- API private (solo per uso interno del servizio stesso)
- API interne (usufruibili solo da servizi autorizzati)

Attacchi

- Denial of Service (DOS)
- Distributed Denial of Service (DDOS)
- Man in the Middle (MITM)
- Cross-site Request Forgery (CSRF)
- Abuso di API pubbliche
- Abuso di API private
- Abuso di API interne
- SQL-Injection
- Cross-site Scripting (XSS)
- Accesso fisico al server
- Furto di credenziali (social engineering, phishing)
- Altro
- Load-balancing
- CDN, accertamento user-agent autentico
- HTTPS-Only Standard (HSTS)
- Token anti-CSRF
- Autenticazione, usage-quota
- Autenticazione, Cross-Origin Resource Sharing (CORS)
- Autenticazione, Cross-Origin Resource Sharing (CORS)
- Fare attenzione, separare dati dal codice
- Fare attenzione, separare dati dal codice
- Accessi controllati, crittografia hardware, disattivazione porte USB
- Autenticazione multi-fattore (codice OTP)
- Altro

Denial of Service (DOS)

- Un hacker chiama ripetutamente le API, migliaia di richieste al secondo
 - Il server non è in grado di gestire il carico di lavoro
 - Gli utenti onesti non riescono più a comunicare con il server
-
- Load-balancing su più server
 - L'hacker riesce a bloccare uno solo dei server
 - La maggior parte degli utenti onesti continuano ad usufruire del servizio

Distributed Denial of Service (DDOS)

- Un hacker fa un DOS con più client, milioni di richieste al secondo
 - I server in load-balancing non riescono a gestire il carico di lavoro
 - Gli utenti onesti non riescono più a comunicare con i server
-
- CDN distribuita su più regioni
 - L'hacker è in grado di bloccare il servizio solo sulla regione nella quale si trovano i client usati per fare l'attacco DDOS
 - Gli altri utenti continuano ad usufruire del servizio
-
- Il server si accerta che lo user-agent sia autentico, captcha
 - L'hacker non è più in grado di fare DDOS tramite bot

Man in the Middle (MITM)

- Utente visita <http://www.banca.it> da WiFi pubblica in un bar
- Proprietario del bar non sa nulla di sicurezza informatica
- Hacker collegato alla stessa WiFi, intercetta la richiesta ad <http://www.banca.it> e la reindirizza al suo <http://localhost> con una copia identica della home-page del sito originale
- Utente fa login con le proprie credenziali
- L'hacker entra in possesso delle credenziali dell'utente
- Il sito è configurato con HTTPS-Only Standard
- Se l'utente ha in passato visitato almeno una volta <https://www.banca.it>, il browser si rifiuta di aprire la versione non autentica del sito, in quanto non HTTPS, altrimenti...
- L'hacker tenta di reindirizzare l'utente sul suo <https://localhost>
- Il browser rileva che il certificato non è valido ed avvisa l'utente del tentativo di hacking

Cross-site Request Forgery (CSRF)

- Utente visita il sito www.banca.it e fa login
 - API www.banca.it/trasferisci?iban=CONTO&qta=EURO
 - Utente chiude il sito dimenticandosi di fare logout
 - Utente visita un sito www.streaming.it di pirateria digitale
 - Di nascosto il sito fa una richiesta all'API per trasferire 10000 euro
 - L'utente non si accorge di nulla, e sono stati prelevati 10000 euro dal suo conto
-
- API www.banca.it/trasferisci?iban=CONTO&qta=EURO&csrf=SECRET
 - Il server genera un token anti-CSRF nella pagina su www.banca.it per fare trasferimenti
 - Quando il server riceve una richiesta all'API, verifica che il token coincida con quello generato
 - Se il token non coincide, la richiesta viene ignorata
-
- In ASP.NET Core questo token viene aggiunto ad ogni form e validato automaticamente

Abuso di API pubbliche

- DOS/DDOS, l'hacker fa migliaia di richieste al secondo
- Server sovraccarico
- Autenticazione, per fare una richiesta alle API pubbliche è necessario autenticarsi con un token
- Usage-quota, max 10 richieste al secondo per ciascun token
- Quando il server rileva un eccesso di richieste dallo stesso utente comincia ad ignorarle

Abuso di API private

- Hacker chiama API private per uso interno del servizio stesso tramite uno script
 - L'hacker è in grado di automatizzare operazioni che normalmente prevedono la presenza di un umano
 - È in grado anche di utilizzare le API private del servizio dal proprio sito pubblico, facendo chiamate REST esterne
-
- Il server viene configurato per mandare direttive Cross-Origin Resource Sharing (CORS) al browser
 - Il browser si rifiuta di chiamare le API a meno che la chiamata non venga fatta dal sito stesso che espone le API private
-
- L'hacker usa/scrive un client che ignora le direttive CORS
 - L'hacker non è più in grado di fare chiamate dal proprio sito pubblico, ma può comunque farle da un client non ufficiale
-
- Autenticazione, per fare richieste il server richiede un token, il token identifica l'utente che fa richieste
 - L'hacker non riesce più a fare richieste senza prima ottenere un token, ma può comunque farle usare un client non ufficiale
-
- Anche se l'utente è un hacker, per fare le richieste dovrà usare un token personale che lo identifica
 - Dai log risulterà un utilizzo anomalo associato all'hacker, i log dovranno essere analizzati per identificare le anomalie
 - L'hacker può essere bannato

Abuso di API interne

- Le API interne si hanno gli stessi problemi delle API private
- Il server può essere configurato con direttive CORS più avanzate
- Il browser permette di fare richieste solo se il dominio dal quale parte la richiesta è tra quelli autorizzati dal server

SQL-injection

- L'hacker identifica la pagina www.banca.it/login?usr=USR&pwd=PWD
- Quando un utente fa login, il server fa una richiesta al database per validare le credenziali
`SELECT u.Role FROM Users u WHERE u.Username = 'USR' AND u.Password = 'PWD';`
- L'hacker fa una richiesta a www.banca.it/login?usr=admin';--&pwd=x
- Il server fa una richiesta al database per validare le credenziali
`SELECT u.Role FROM Users u WHERE u.Username = 'admin';--' AND u.Password = 'x';`
- L'hacker si autentica con l'utenza di admin
- Il programmatore separa codice e dati, utilizzando variabili per passare i dati al database
`SELECT u.Role FROM Users u WHERE u.Username = @usr AND u.Password = @pwd;`
- SQL-injection evitabile usando le variabili, il database non le converte mai in codice
- Entity Framework separa automaticamente codice e dati, ma è importante ricordarsi di usare le variabili ogni volta che si fa una query senza passare da un ORM come Entity Framework

Cross-site Scripting (XSS)

- L'hacker crea un profilo sul sito www.gattini.it/user/h4ck3r, un sito di NFT di gattini
- Il sito www.gattini.it permette di caricare una biografia Bio... visibile in una sezione `<p>Bio...</p>` sulla pagina di profilo
- L'hacker carica una biografia contenente codice HTML
`Ciao</p><script>fetch("/api/invia_gattino?id=1234&user=h4ck3r")</script><p>`
- Tutti gli utenti che visitano la pagina di profilo dell'hacker eseguono il codice iniettato
`<p>Ciao</p><script>fetch("/api/invia_gattino?id=1234&user=h4ck3r")</script><p></p>`
- Se uno di questi utenti è il proprietario del gattino 1234, il gattino viene trasferito sull'account dell'hacker
- Il programmatore fa attenzione a separare i dati «contaminati dagli utenti» dal resto del codice del sito
`<p>@biografia</p>`
- ASP.NET Core fa automaticamente l'escape di tutte le variabili nei template cshtml (Razor), limitando il rischio di attacchi XSS
`<p></p><script>fetch("/api/invia_gattino?id=1234&user=h4ck3r")</script><p></p>`
- È previsto anche un meccanismo a carico del browser, chiamato CSP (Content Security Policy), che previene l'esecuzione di script non riconosciuti bloccando potenziali attacchi XSS anche se il programmatore si è dimenticato di fare un escape di valori «contaminati»

<https://learn.microsoft.com/aspnet/core/security/cross-site-scripting>

<https://developer.mozilla.org/docs/Web/HTTP/CSP>

Password

Come salvarle in modo sicuro su un database

Sicurezza

- Il database potrebbe essere rubato...
 - ...furto di credenziali di uno degli amministratori del database
 - ...tramite SQL-injection
 - ...sfruttando uno zero-day sul server
 - ...errore di configurazione
 - ...accesso fisico al server del database

Password su database

- Clear-text
 - Se il database viene rubato, tutte le password sono leggibili
- Hash della password
 - Se il database viene rubato, le password non sono leggibili
 - Le password non sicure sono recuperabili con un dictionary-attack
 - Password recuperabili tramite rainbow-tables
- Hash con salt
 - Se il database viene rubato, le password non sono leggibili
 - Le password non sicure sono recuperabili con un dictionary-attack
 - Password non recuperabili tramite rainbow-tables
- Hash con salt e key-derivation
 - Se il database viene rubato, le password non sono leggibili
 - Non è possibile fare un dictionary-attack (almeno non in un tempo ragionevole)
 - Password non recuperabili tramite rainbow-tables

Autenticazione

Cookie, bearer token, JWT, e protocollo OpenID Connect

Cookie

- È possibile memorizzare l'identità di un utente usando un cookie
 - L'utente si autentica inserendo le proprie credenziali
 - Viene generato un cookie firmato contenente la sua identità
 - Ad ogni richiesta viene allegato il cookie che identifica l'utente
-
- Questo meccanismo è valido se si lavora su un singolo dominio
 - Non permette di identificare l'utente tra domini diversi
 - Non permette di dimostrare l'identità dell'utente a servizi esterni

Bearer token

- È un token semplice di autenticazione
- Un utente ottiene un “bearer token” (token al portatore) al login
- L'utente si identifica allegando il token alle richieste successive
`Authorization = Bearer SOME_TOKEN_HERE`
- È parte dello standard web OAuth 2.0

JSON Web Token (JWT)

- È un token strutturato, può essere usato come Bearer token
- Composto da tre sezioni codificate in base64
 - Intestazione: definisce il formato delle altre sezioni
 - Payload: contiene dati arbitrari in JSON
 - Firma: firma di intestazione e payload, usando un secret noto al server

token	intestazione	payload	firma
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6IkpvaG4gRG9lIiwiaWF0IjoxNTE2MzkwMjQsImF0IjoiIn0.XbPfbiHMI6arZ3Y922BhjWgQzWXcXNrZ0ogtVhfEd2o	{ "alg": "HS256", "typ": "JWT" }	{ "sub": "1234567890", "name": "John Doe", "iat": 1516239022 }	HMACSHA256(base64UrlEncode(intestazione) + "." + base64UrlEncode(payload), "secret")

OAuth 2.0

- È lo standard di autenticazione inter-servizio più utilizzato
- Prevede che autenticazione e autorizzazione siano distinte
 - Autenticazione consiste nell'identificare l'utente
 - Autorizzazione definisce per quali risorse è consentito l'accesso
- È ricco di funzionalità, versatile
- Abbastanza complesso

OpenID Connect (OIDC)

- Si basa su OAuth 2.0
- Prevede (almeno) tre servizi distinti:
 - Client: è colui che desidera accedere ad una **risorsa** a nome di un **utente**
 - **Authorization service**: permette di ottenere un **token di autorizzazione** che identifica l'**utente** e garantisce accesso ad **un set di risorse (scope)**
 - Token service: permette di scambiare un **token di autorizzazione** con un token
 - **Resource server**: permette di accedere ad una **risorsa**, tramite un token che garantisce accesso a quella **risorsa**
- Definisce un meccanismo per fare in modo che sia lo user-agent (il browser) dell'utente a gestire tutti i token, i vari servizi/server non comunicano mai direttamente tra di loro, se non per effettuare la validazione del token ricevuto con la richiesta

Flusso OIDC

- Un client desidera accedere ad una o più risorse a nome di un utente
- Il client contatta il server OIDC per leggerne la configurazione
<https://example.com/.well-known/openid-configuration>
- Nella configurazione è indicato un indirizzo di autorizzazione
<https://example.com/authorize>
- Il client apre la pagina del server di autorizzazione su un browser web (se il client è una pagina web, fa un redirect), aggiungendo un parametro con l'indirizzo di ritorno e le risorse (scope) alle quali desidera accedere
- A questo punto il controllo passa al server di autorizzazione
- Se l'utente non è autenticato, il server chiede all'utente di autenticarsi
- Eventualmente viene chiesto all'utente se è sicuro di voler autorizzare il client ad accedere alle risorse richieste a proprio nome
- Una volta che l'utente è autenticato, l'utente viene reindirizzato all'indirizzo di ritorno, aggiungendo un parametro con il token di autorizzazione
- Il client scambia il token di autorizzazione con un token, inviandolo all'indirizzo indicato nella configurazione
<https://example.com/token>
- Il servizio genera un token che restituisce al client, ed invalida il token di autorizzazione utilizzato, per evitare che possa essere riciclato
- Il token può essere utilizzato per accedere alle risorse, allegandolo ad una richiesta indirizzata al server delle risorse
- Il server delle risorse verifica che il token sia autorizzato ad accedere alla risorsa richiesta
- Il token viene validato tramite una richiesta al servizio relativo
- Se il token risulta valido, viene restituita al client la risorsa richiesta

Flusso OIDC

- Isola i vari attori coinvolti nel processo, evitando la necessità di un'interconnessione
- Lo user-agent (browser) ha la responsabilità di gestire i token
- È totalmente sicuro e mantiene la sicurezza tra i vari attori
- È relativamente complesso
- Sono previsti altri meccanismi non descritti in questo corso
- OpenIdDict, implementazione open-source in .NET

Autenticazione

Come implementarla in ASP.NET Core

<https://learn.microsoft.com/aspnet/core/security/authentication/>

Risorse esterne

OpenIddict <https://github.com/openiddict/openiddict-core>

The background is a solid blue color with a repeating pattern of small, white, stylized icons. These icons represent various technology and business concepts, including a smartphone, a robot head, a document with a checklist, a gear, a network of three nodes, a line graph, a lightning bolt, a database cylinder, a laptop, a cloud, a flask, a document with a code symbol, a headset, and a target symbol.

Grazie!