



Università
Ca' Foscari
Venezia

Corso di Laurea in Informatica

Ordinamento ex D.M. 270/2004

Tesi di Laurea

Web APIs Security

Relatore

Dott. Stefano Calzavara

Correlatore

Dott. Marco Squarcina

Laureando

Daniele Rigon

Matricola 857319

Anno Accademico

2018/2019

Alla mia famiglia.

INDICE

1	Introduzione	1
2	Geolocation API	3
2.1	Overview	3
2.2	Metodi	3
2.3	Esempio di implementazione	4
2.4	Rischi e sicurezza	6
2.5	Conclusioni	7
3	Payment Request API	9
3.1	Overview	9
3.2	Metodi	9
3.3	Esempio di implementazione	10
3.4	Rischi e sicurezza	11
3.5	Conclusioni	13
4	Service Worker	15
4.1	Overview	15
4.2	Ciclo di vita di un Service Worker	16
4.3	Rischi e sicurezza	17
4.4	Conclusioni	18
5	Conclusioni	19

CAPITOLO 1

INTRODUZIONE

Nel corso degli ultimi decenni la tecnologia è diventata sempre più presente nella quotidianità di ogni individuo, che riguardi lavoro, svago o vita quotidiana, e in particolar modo Internet. Come ogni cosa presenta però aspetti positivi e negativi e, essendo il World Wide Web un mondo vastissimo, sono numerosi i rischi al suo interno.

In questa tesi ho preso in esame uno strumento molto utilizzato e che probabilmente alcuni di noi ne usufruiscono senza saperlo, le Application Programming Interface (API). Questo strumento, molto diffuso in ambiente informatico, semplifica molto la programmazione. Esso infatti permette di usufruire di informazioni messe a disposizione da terze parti evitando di scrivere da zero i codici, eliminando lavoro non a valore aggiunto oltre ad una correttezza della funzionalità utilizzate oltre a ciò questo insieme di protocolli e procedure rende la comunicazione tra piattaforme diverse molto più semplice. Ogni volta che ci si logga in un sito tramite Google o Facebook, o si utilizzano le mappe di Maps all'interno di pagine Web, o si accede ad un database durante la compilazione di un form online, stiamo utilizzando le API. Pensando più in grande le API si utilizzano anche nell'intelligenza artificiale, come servizi di chatbot ¹, nell'*Internet of Things (IoT)* ² e nel *Machine Learning* ³. Tutto ciò porta anche ad una esperienza utente più intuitiva e di facile apprendimento.

La tesi è articolata in tre capitoli. Il primo esplica la *Geolocation API* [1] utile a tracciare la posizione all'interno di un sito web o un'applicazione online. Nel secondo è spiegata *PaymentRequest API* [2] la quale, salvando le informazioni imputate, permette la riutilizzazione di essere senza il reinserimento manuale. Nel terzo ed ultimo capitolo l'attenzione verte sui *Service Worker* [3], usati principalmente nelle *Progressive Web App* ⁴ ai fini di miglioramento dell'esperienza utente.

Tutte le informazioni presenti nella tesi sono reperibili al seguente repository GitHub: <https://github.com/danielerigon4/Tesi-Web-API>.

¹Assistenti virtuali disponibili 24/7 capaci di dare risposte in tempo reale

²Internet applicato al mondo concreto (smarthouse, industria 4.0, ecc.)

³Ramo dell'intelligenza artificiale che studia e crea algoritmi che apprendano informazioni da un insieme di dati in modo da crearne delle previsioni

⁴Applicazioni presenti sulle pagine web che si comportano come applicazioni native

CAPITOLO 2

GEOLOCATION API

2.1 Overview

La Geolocation API viene utilizzata per ottenere la posizione geografica di un utente. Poiché questo può compromettere la privacy, la posizione non è disponibile a meno che l'utente non la approvi: in un dispositivo mobile si usano le coordinate provenienti dal sensore GPS, mentre in un computer l'indirizzo IP.

2.2 Metodi

Nella Geolocation API la posizione viene resa nota tramite la proprietà *navigator.geolocation* [4] dell'oggetto *Navigator*, la quale possiede esclusivamente tre metodi: *getCurrentPosition*, *watchPosition* e *clearWatch*. I primi due restituiscono la posizione corrente, mentre il terzo annulla la ricerca. La differenza tra i primi due è insita nella loro periodicità: se il primo metodo fornisce il dato una sola volta, il secondo lo comunica automaticamente a intervalli di tempo fissati o ogni qualvolta la posizione cambia. Va specificato che *getCurrentPosition* è pensata per restituire la posizione nel più breve tempo possibile, avendo quindi una bassa precisione. Da notare che *clearWatch* può essere utilizzato solamente se in precedenza uno dei due metodi è stato avviato, quindi se esiste una posizione.

La sintassi per invocare tali metodi è la seguente:

```
1 navigator.geolocation.getCurrentPosition(SuccessCallback ,  
    errorCallback , Opzioni);  
2 navigator.geolocation.watchPosition(SuccessCallback ,  
    errorCallback , Opzioni);  
3 navigator.geolocation.clearWatch(PosizioneEsistente);
```

E' possibile distinguere tre parametri della funzione per *getCurrentPosition* e *watchPosition*. La *SuccessCallback*, parametro obbligatorio, viene richiesta quando la chiamata ad una nuova posizione va a buon fine. *ErrorCallback* e *Opzioni*, invocate rispettivamente per gestire la presenza di uno o più errori e definire i parametri delle chiamate alle due callback, sono invece parametri opzionali.

2.3 Esempio di implementazione

Supponiamo di avere la seguente pagina di esempio [5] dove poter osservare il funzionamento dei tre metodi descritti precedentemente eseguendoli tramite i tre pulsanti presenti nella pagina.

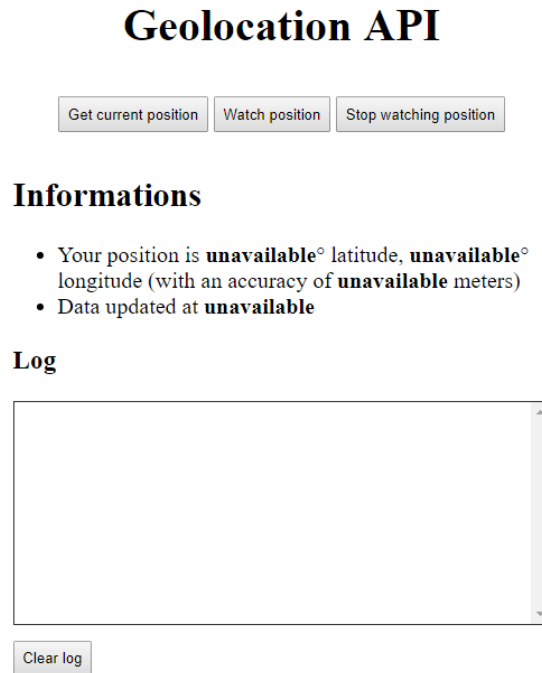


Figura 2.1: Geolocation API - Implementazione pagina d'esempio

L'invocazione avviene tramite un *addEventListener*

```
1 document.getElementById('button-get-position').  
  addEventListener('click', function() {  
2   navigator.geolocation.getCurrentPosition(success, error,  
    positionOptions);  
3 });  
4 document.getElementById('button-watch-position').  
  addEventListener('click', function() {  
5   watchId = navigator.geolocation.watchPosition(success,  
    error, positionOptions);  
6 });  
7 document.getElementById('button-stop-watching').  
  addEventListener('click', function() {  
8   if (watchId !== null) {  
9     navigator.geolocation.clearWatch(watchId);  
10  }  
11 });
```

che farà comparire una finestra di richiesta posizione.



Figura 2.2: Geolocation API - Richiesta permesso posizione

Se l'utente rifiuta la posizione non è calcolata, viene richiamata l'*ErrorCallback* visualizzando un messaggio di errore.

Log

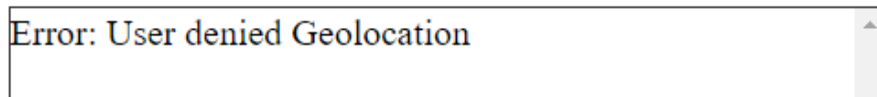


Figura 2.3: Geolocation API - Rifiuto autorizzazione

Viceversa, se l'utente acconsente, la chiamata alla funzione *SuccessCallback* andrà a buon fine

```

1 function success(position) {
2   document.getElementById('latitude').innerHTML = position.
     coords.latitude;
3   document.getElementById('longitude').innerHTML = position.
     coords.longitude;
4   document.getElementById('position-accuracy').innerHTML =
     position.coords.accuracy;
5   document.getElementById('altitude').innerHTML = position.
     coords.altitude ? position.coords.altitude: 'unavailable';
6   document.getElementById('altitude-accuracy').innerHTML =
     position.coords.altitudeAccuracy ? position.coords.
       altitudeAccuracy: 'unavailable';
7   document.getElementById('heading').innerHTML = position.
     coords.heading ? position.coords.heading: 'unavailable';
8   document.getElementById('speed').innerHTML = position.
     coords.speed ? position.coords.speed: 'unavailable';
9   document.getElementById('timestamp').innerHTML = (new Date(
     position.timestamp)).toString();
10 }

```

e compariranno tutte le informazioni disponibili, come raffigurato nella figura sottostante.

Geolocation API



Informations

- Your position is **45.5550451°** latitude, **12.0342556°** longitude (with an accuracy of **28** meters)

Figura 2.4: Geolocation API - Accettazione autorizzazione

2.4 Rischi e sicurezza

Ognuno di noi ogni giorno, consapevolmente o meno, condivide la propria posizione più e più volte. Ce ne accorgiamo quando utilizziamo applicazioni di fitness che tracciano i nostri percorsi o quando impostiamo il navigatore verso una destinazione. Succede però che ci compaiano delle notifiche nelle quali il cellulare ci chiede di impostare certe destinazioni frequenti come luoghi di interesse, o addirittura ci propone orari di partenza casa - lavoro per trovare meno traffico. Così come possiamo ricercare una foto tramite geo tag invece di scorrere album interi. Tutto ciò è reso possibile da questa API. Viene spontaneo chiederci dove vadano questi dati, se sono tenuti al sicuro e soprattutto se possono essere rubati. Le norme sulla privacy di Moves prevedono che i dati dell'app siano caricati dal tuo cellulare ai server, che contengono i dati personali, tramite una connessione sicura. I dati dei singoli utenti vengono localizzati assegnando un numero di ID al tuo telefono, o via email se decidi di registrarti in questo modo. Secondo le norme sulla privacy e sulla condivisione dei dati, tutti i canali di comunicazione che trasmettono dati personali devono essere crittografati, non saranno dunque ceduti a terze parti senza il consenso del diretto interessato. Tuttavia nulla è garantito totalmente.

Possono esserci infatti differenti modi per rubare informazioni sensibili, due dei quali vengono presi in analisi.

Una possibilità è data dal fatto che l'utente accetti di condividere la propria posizione all'interno di un sito malevolo, ignaro del fatto che tale indirizzo non sia sicuro. Un'altra modalità di attacco, più diffusa rispetto alla precedente, avviene attraverso un *Cross-Site Scripting* (XSS). Si tratta di una falla delle applicazioni web che provoca l'inserimento e l'esecuzione di script lato client permettendo la visualizzazione, raccolta e manipolazione di dati, oltre alla modifica delle pagine web. Tale tipo di vulnerabilità è di due tipi: persistente e non. Il Cross-Site Scripting non persistente è sfruttato prettamente nelle pagine web senza forme di controllo mentre l'attacco XSS persistente avviene quando i dati forniti vengono conservati in un server, o un database, restando visibili. Per capire meglio la differenza pensiamo a due esempi. Quando si compilano i campi di un form online e i dati vengono inviati si è nel primo caso. Si supponga invece di essere iscritti ad un sito; in questo caso informazioni come nome, cognome e mail sono salvati in un server, ma nascoste agli utenti. Un possibile attaccante può rubare tali informazioni inserendo uno script progettato in modo tale che, quando il proprio profilo viene visitato, o qualunque azione sia associata a tale script, il browser lo elabori in modo da rubare le informazioni sensibili degli altri utenti. Prendendo questo esempio, dato che latitudine e longitudine,

utili a tracciare le coordinate dell'utente, risiedono all'interno del *Document Object Model (DOM)*¹, ci troviamo nella situazione sopra descritta. All'apertura di una pagina all'interno di un sito web potrebbe comparire una finestra come quella in Figura 2.2. Questa potrebbe essere collegata ad uno script come questo [6]

```
1 function showPosition(position){
2     var latitude = position.coords.latitude;
3     var longitude = position.coords.longitude;
4     var pos= "Latitude: " + latitude + "<br>Longitude: " +
        longitude;
5     document.getElementById("mydiv").innerHTML = pos;
6 }
7 function getLocation(){
8     navigator.geolocation.getCurrentPosition(showPosition);
9 }
10 getLocation();
```

il quale salva *coords.latitude* e *coords.longitude* transitanti nel DOM memorizzandole in una variabile. Se l'utente non disabilita il tracciamento della posizione, che rimarrà esposta, si ha un attacco XSS persistente.

2.5 Conclusioni

In questo capitolo si sono analizzate le specifiche, i vantaggi e i problemi di questa API. Ci si rende conto come ognuno di noi ogni giorno, anche senza saperlo, utilizzi questo strumento, avendone molti vantaggi. Uno sportivo può tenere traccia dei percorsi di allenamento, un rappresentante degli itinerari lavorativi, un corriere delle consegne fatte e previste e chiunque può ritrovare il proprio smartphone smarrito. Oggigiorno, anche grazie alla connessione continua con questi strumenti, la necessità di fare uso di questi strumenti è quasi indispensabile, essendo comoda, veloce e utile a molte esigenze in un mondo ormai sempre più veloce. Nell'era dell'industria 4.0 ogni azienda adotta soluzioni tecnologiche per l'automatizzazione dei processi verso il futuro portando, a volte incosapevolmente per i problemi visti al paragrafo precedente, all'aumento del rischio di tutela della privacy personale anche dei dipendenti nel caso di sottrazione di dati. In ultima analisi ci possono essere anche dei problemi legati alla sicurezza personale. È stato detto all'inizio del paragrafo precedente che i moderni software sono in grado di identificare il luogo presente e futuro di un individuo grazie all'intersecazione di informazioni di geolocalizzazione con quelle personali. Questo può facilitare la possibilità di danni a cose, persone e/o proprietà (furto, stalking, rapimento, violenza domestica, furto d'identità, ecc) oltre alla sottrazione di informazioni personali (indirizzo di casa, lavoro, scuola, itinerari giornalieri, ecc) non solo individuali.

Ci si può difendere utilizzando dei comportamenti idonei a garantire la sicurezza delle nostre informazioni. Come fa già la maggioranza delle persone, installare un buon antivirus ed evitare siti e mail che non riteniamo sicure sono un buon punto

¹API standardizzata da W3C per accedere e modificare gli elementi di una pagina web

d'inizio. Connettersi ad una rete aperta è uno dei modi più semplici, per un hacker, di rubare dati. Chiunque, tramite la creazione di un hotspot Wi-fi, può rendere credibile il nome della propria rete monitorando ogni accesso. Un altro errore di uso comune è non cambiare le password periodicamente, o peggio usare la stessa password per la maggiorparte dei servizi che utilizziamo. Entrando più nel dettaglio si può salvaguardare ancora di più la propria sicurezza utilizzando browser alternativi ai più comuni, alcuni dei quali pensati apposta a questo scopo, come SRWIron o TorBrowser. Supponendo di non voler cambiare il proprio browser, per comodità o abitudine, si può fare uso delle estensioni (oltre a quelle offerte da antivirus, come Norton Security), come Disconnect, Ghostery, HTTPS Everywhere. Ricollegandosi a quest'ultima un altro comportamento importante per la salvaguardia dei dati è la navigazione in pagine crittografate, ovvero quello nella forma *https://* invece che *http://*. Oltre ai browser, anche la scelta del motore di ricerca è fondamentale. Infine, dato che un *Internet Service Provider (ISP)* ² può monitorare tutto ciò che un utente fa in rete, l'uso di una *Virtual Private Network (VPN)* ³.

²Organizzazione che offre, tramite la stipulazione di un contratto, servizi Internet ai propri utenti

³Rete Internet privata, nella quale servono delle credenziali per poter accedere, dove ogni trasferimento è criptato

CAPITOLO 3

PAYMENT REQUEST API

3.1 Overview

La *PaymentRequest API* [7] nasce con l'intento di semplificare i pagamenti online. Ogni sito web ha il proprio sistema di pagamento e molti richiedono la ridigitazione manuale più volte delle stesse informazioni che potrebbero invece essere memorizzate e riutilizzate dall'API stessa per completare più rapidamente le transazioni online. I vantaggi sono molteplici [8] e interessano aspetti differenti che vanno da una maggior rapidità nell'acquisto a una gestione delle informazioni più veloce e intuitiva. Chiunque utilizzi l'API può gestire i propri dati direttamente dal browser (carte di credito, indirizzi di spedizione, ecc.) avendo tutte queste informazioni sincronizzate così da potervi accedere da ogni dispositivo. Il browser, inoltre, è in grado di verificare la validità delle carte, di comunicare se ce ne sono di scadute (o in prossimità di scadenza) e di suggerire quale di queste utilizzare. In base alle esigenze è possibile anche impostare carte predefinite.

3.2 Metodi

Il costruttore *PaymentRequest* ha bisogno di diverse informazioni per poter costruire la richiesta di pagamento [2]. *PaymentMethodData* [9] rappresenta l'elenco dei metodi di pagamento accettati, *PaymentDetails* [10] rappresenta i dettagli della transazione (costi, servizi, opzioni di spedizione, ecc.) mentre *PaymentOptions* [11] fornisce le informazioni sulla consegna del prodotto (indirizzo fisico, e-mail, ecc.). Dopo aver raccolto tutti questi dati è possibile costruire l'oggetto *PaymentRequest*. Una cosa importante da specificare è che il metodo *show()* [12] garantisce la visualizzazione singola dell'interfaccia di pagamento, in modo da non portare l'utilizzatore a commettere errori. Provando ad aprire più di una finestra verrà visualizzato un messaggio di errore come mostrato nella seguente figura.

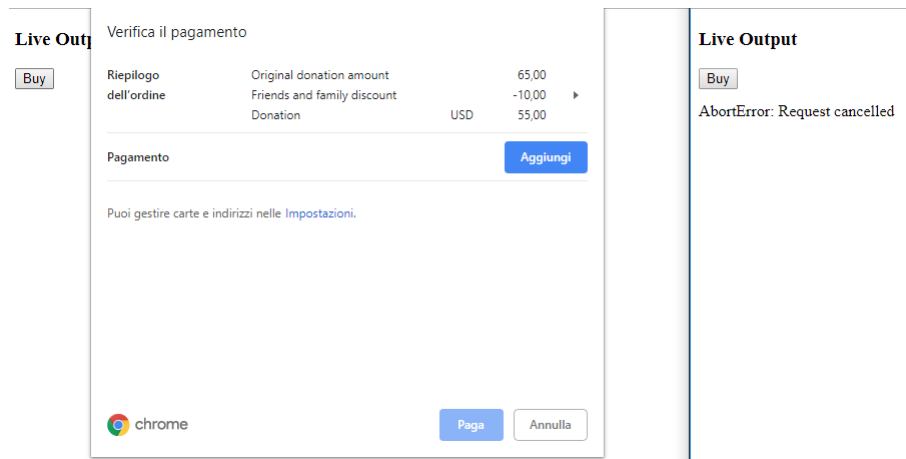


Figura 3.1: *PaymentRequest* API - Messaggio di errore generato dal metodo *show()*

3.3 Esempio di implementazione

Supponiamo di aver creato l'oggetto *PaymentRequest* contenente, come descritto nel paragrafo precedente, *MethodData*, *Details* e *Options*.

```

1 // PaymentMethodData
2 var methodData = [
3   {
4     supportedMethods: ['basic-card'],
5     data: {
6       supportedNetworks: ['visa', 'mastercard', 'amex'],
7       supportedTypes: ['credit']
8     }
9   }
10 ];
11 //PaymentDetails
12 var details = {
13   displayItems: [
14     {
15       label: "Sub-total",
16       amount: { currency: "USD", value : "100.00" }
17     },
18     {
19       label: "Sales Tax",
20       amount: { currency: "USD", value : "9.00" }
21     }
22   ],
23   total: {
24     label: "Total due",
25     amount: { currency: "USD", value : "109.00" }
26   }
27 };
28 //PaymentOptions
29 var options = {
30   requestShipping: true

```



```

31 };
32 //PaymentRequest
33 var payment = new PaymentRequest(methodData, details, options
    );

```

Una volta creato l'oggetto *PaymentRequest*, esso sarà visualizzabile tramite la chiamata al metodo *request.show()* dove possono essere inserite le informazioni di pagamento, l'indirizzo di spedizione e una serie di altri campi in base a come è stato costruito l'oggetto (Figura 3.1). Quando tutti i campi saranno compilati e si selezionerà il tasto "paga" sarà creata la *promise* che potrà avere esito positivo o negativo. In caso di esito positivo verrà soddisfatta la *promise* e saranno consegnate al commerciante tutte le informazioni inserite dall'acquirente cosicché possano essere elaborate sul back-end. Viceversa, in caso di esito negativo, l'operazione di pagamento non andrà a buon fine e comparirà un messaggio di errore che inviterà l'utente a modificare le credenziali reindirizzandolo alla pagina precedente.

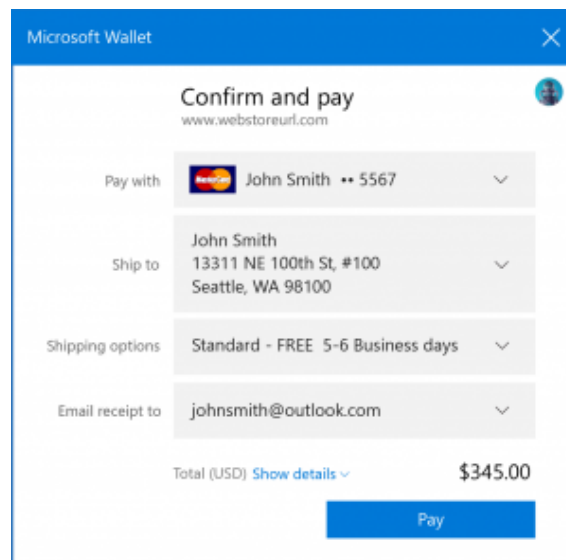


Figura 3.2: *PaymentRequest API* - Inserimento credenziali da parte dell'utente

3.4 Rischi e sicurezza

Più o meno ogni sito a cui accediamo al giorno d'oggi ha un'area riservata in cui creare il proprio profilo utente, costringendoci a ricordare le credenziali di ognuno di essi, che per comodità tendiamo a far memorizzare al browser. La *PaymentRequest API* cerca di garantire la sicurezza dei dati sensibili attraverso alcune scelte implementative [13]. Una di queste è riguardante il metodo *show()* descritta nei due paragrafi sopra. Oltre a quanto detto, l'API richiede inoltre che esso venga attivato solo tramite la volontà da parte dell'utente, ad esempio con un clic. Tale API, per garantirne la sicurezza, opera solamente in contesti sicuri HTTPS¹. Inoltre, i

¹HyperText Transfer Protocol over Secure Socket Layer, un protocollo che garantisce una comunicazione in rete sicura attraverso una rete di computer su Internet

commercianti possono delegare la memorizzazione delle informazioni a fornitori di servizi di pagamento o al browser, senza avere a carico informazioni sensibili dell'utente. Questo li solleva da responsabilità importanti in quanto l'API non supporta direttamente la crittografia dei campi dati.

Come per la precedente API, anche in questa vi sono differenti possibilità di attacco. Oltre all'attacco XSS, già descritto al capitolo precedente, del quale ne viene fornito un esempio successivamente, un'altro modo per rubare informazioni è il *phishing*. Esso è una truffa che vive all'interno di Internet attraverso la quale vengono inserite informazioni personali in pagine che, fingendosi affidabili e autorevoli, non lo sono. Nella maggiorparte dei casi inizia con banner pubblicitari, messaggi in social come Facebook e, la quasi totalità delle volte, con una mail. Il messaggio ricevuto si presenta come richiesta di apertura di un link da parte di enti, banche o conoscenti, che rimanda a pagine falsamente reali nelle quali inserire i propri dati. Esiste anche una modalità più aggressiva di phishing. Allo stesso modo di come già descritto viene mandata una mail, o un messaggio, apparentemente reale, contenente file .exe invece di indirizzi URL. Tali file sono virus che, una volta eseguiti, si diffondono all'interno del pc, carpando ogni informazioni in base a come sono costruiti. Tornando al primo tipo di attacco, simuliamo anche qui, come nella precedente API, l'inserimento di script dannosi all'interno di una pagina web. Supponiamo di navigare all'interno di un sito affidabile e presumiamo di essere in una pagina di pagamento, come il carrello di Amazon o un'inserzione su Facebook. Inserendo uno script in una di queste pagine, quando il consumatore cliccherà sulla voce di pagamento, sarà eseguito tale script invece di quello del sito. L'utente inserirà tutte le informazioni necessarie al pagamento convinto di essere sulla pagina sicura ma, al contrario, queste saranno mandate e/o visualizzate dall'attaccante.

← Modifica la carta

Carte di credito accettate

Numero carta* 4916 6293 0528 7782

Nome sulla carta di credito* aaa

Data di scadenza* 06 2021

Indirizzo di fatturazione* aaa 123, aaa Aggiungi

* Campo obbligatorio

Fine Annulla pagamento

← Inserisci il codice CVC della carta Visa **** 7782

Dopo essere stati confermati, i dati della carta saranno condivisi con questo sito.

111

Conferma Annulla pagamento

Figura 3.3: PaymentRequest API - Attacco XSS

Alla richiesta di avvio di un processo di pagamento, ovvero dopo la chiamata al metodo *show()*, è creato l'oggetto *PaymentRequest* nel quale sono inseriti tutti i dati dall'utente. Tale oggetto passa però attraverso il DOM e può quindi essere intercettato e visualizzato. Ad esempio nel campo *details.cardNumber* si può leggere il numero di carta di credito, mentre in *details.cardSecurityCode* si può leggere il codice CVV, come mostrato nella figura successiva.

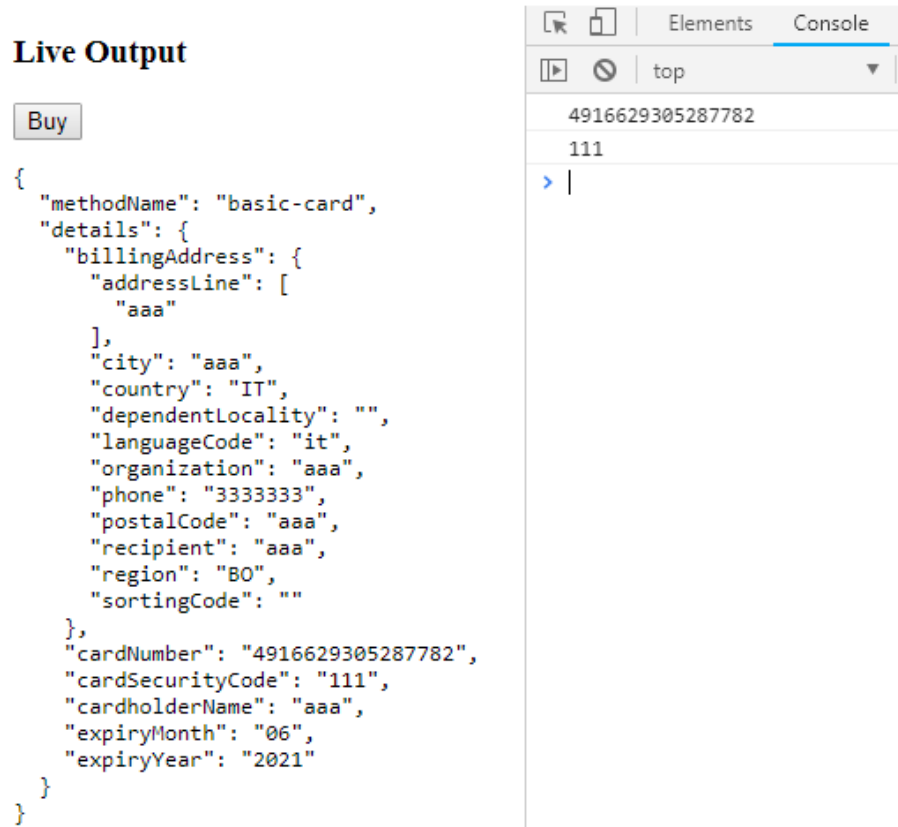


Figura 3.4: *PaymentRequest* API - Informazioni rubate

3.5 Conclusioni

È facile pensare come ognuno di noi utilizzi tutti i giorni gli strumenti descritti in questo capitolo, dagli acquisti su molti siti di e-commerce all'inserimento di credenziali in varie piattaforme. Lo si fa talmente tanto spesso che non si bada nemmeno più se il sito in cui inseriamo i nostri dati sia sicuro o meno, dimenticandoci che sito può essere costruito bene, ma non è detto che sia immune a vulnerabilità e problemi. I vantaggi descritti sono molteplici, quindi conviene pensare di prevenire possibili danni con comportamenti alla portata di tutti, per poter beneficiare di questo strumento. Per quanto riguarda la prevenzione agli attacchi XSS, questa informazione è stata descritta ampiamente nelle conclusioni del capitolo precedente. Vediamo ora le modalità di difesa contro il phishing. Vi sono alcune attività alla portata di tutti per aumentare la propria sicurezza, che possono sembrare banali ma è meglio ribadirle: non aprire mail o documenti da origini sconosciute, non rivelare le proprie

password, aggiornare sempre antivirus e browser applicando le patch di sicurezza ². È sempre buona norma, inoltre, verificare l'indirizzo URL della pagina, innanzitutto verificando sia HTTPS, oltre ad assicurarsi non sia una copia del sito originale (ad esempio se sappiamo che il dominio di un certo sito risulta essere .gov e notiamo avere .it allora qualcosa non va).

²Aggiornamenti contenenti correzioni di vulnerabilità riscontrate. La maggior parte ha cadenza mensile

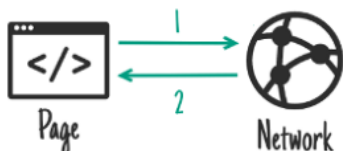
CAPITOLO 4

SERVICE WORKER

4.1 Overview

Un Service Worker [14] è uno script Javascript, avviato in background dal browser, utilizzato per aumentare le performance delle applicazioni web. Esso si trova tra l'applicazione Web e la rete e, similmente ad un *server proxy* ¹, può intercettare tutte le richieste passanti attraverso di lui agendo secondo regole decise in fase di creazione.[15]

Pagina web richiesta senza service worker



Pagina web richiesta con service worker

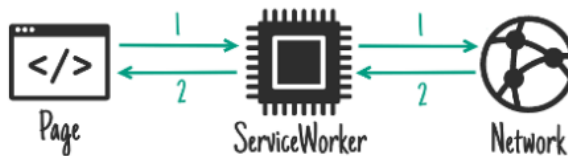


Figura 4.1: Service Worker API - Utilizzo Service Worker in una pagina web

Questo script, essendo separato dalla pagina, non può modificarne gli elementi, può comunicare con essi solamente attraverso le *Promises* ², pertanto ogni operazione è eseguita asincronicamente e gli elementi del DOM possono essere modificati solo tramite messaggi. Tale strumento migliora l'esperienza utente grazie a più funzionalità: l'aggiornamento delle risorse risiedenti nel server, la sincronizzazione dei dati in background, la prelettura delle risorse che serviranno all'utente nel prossimo futuro. In tutto ciò, è il browser a decidere quando il Service Worker sarà o meno

¹Intermediario tra web client e ricerca in altri server.

²In Javascript le funzioni, per aspettarsi l'un l'altra, fanno uso delle callback ma, se le funzioni sono molteplici, il codice può diventare lungo e di difficile comprensione. Utilizzando le promise le funzioni eseguiranno i pezzi di codice corrispondenti, in base al fatto che la promise sia stata mantenuta o meno.

in esecuzione. Da notare che, per evitare attacchi *man-in-the-middle* ³, i Service Worker operano esclusivamente in ambienti HTTPS.

4.2 Ciclo di vita di un Service Worker

Il ciclo di vita di un Service Worker [16, 17, 18] è composto da quattro fasi successive che si articolano a partire dalla Registrazione, passando per l'Installazione, l'Attivazione e concludendosi con il Fetch.

La fase di *Registrazione* avviene solamente dopo aver comunicato al browser l'esistenza di un Service Worker all'interno della pagina. La registrazione si realizza mediante il metodo `ServiceWorker.register()` inserendo in tutte le pagine coinvolte uno script simile a quello sotto riportato. Il codice inizia con il controllo, da parte del browser, della presenza del metodo `navigator.serviceWorker`: se supportato, il Service Worker viene registrato correttamente con `navigator.serviceWorker.register` restituendo una Promise. Dato che i Service Worker operano solo in ambienti sicuri, analogamente a quanto visto per la *Payment Request API*, se l'applicazione non è in HTTPS essi non vengono registrati.

```
1 if ('serviceWorker' in navigator) {  
2     navigator.serviceWorker.register('/service-worker.js').  
   then(function(registration) {  
3         // Il Service worker è installato correttamente  
4     }).catch(function(error) {  
5         // La registrazione del Service Worker non è andata a  
   buon fine  
6     });  
7 }
```

Nella fase di *Installazione* il Service Worker viene scaricato e installato al primo accesso ad una pagina. Sarà poi scaricato periodicamente ad intervalli di tempo fissati; se il nuovo file risulta differente da un Service Worker esistente la nuova versione sarà installata in background ma l'attivazione avverrà quando non ci saranno più pagine che stanno ancora utilizzando il vecchio servizio.

Una volta installato, il Service Worker passa alla fase di *Attivazione* nella quale può controllare gli eventi generati dal client. Possono esistere differenti Service Worker che saranno aggiornati e sostituiti ma solamente un Service Worker per volta potrà essere eseguito all'interno dello stesso contesto. È compito dell'evento *activate* svuotare la cache obsoleta delle versioni precedenti. Una volta che il Service Worker è installato e non sono presenti residui dei Service Worker precedenti, esso potrà effettuare il fetching di risorse restando in attesa di eventuali eventi.

L'ultima fase, relativa all'esperienza utente, è quella di *Fetch*, ovvero l'intercettazione da parte del Service Worker di ogni evento generato dal client, rispondendo alle esigenze secondo le differenti strategie di caching.

³Attacco informatico legato alla sicurezza web in cui viene alterata la comunicazione tra due parti interessate che credono di comunicare direttamente tra di loro.

4.3 Rischi e sicurezza

I Service Worker, pur operando solo in contesti protetti, non sono privi di limiti legati alla sicurezza [19, 20]. Un Service Worker infatti può importare script da qualsiasi altra origine tramite il metodo *importScripts*. Per mitigare rischi il browser richiede che l'URL di registrazione del Service Worker provenga dalla stessa origine dei propri chiamanti (*https://example.com* è diverso da *https://example.evil.com*). Per questo motivo, per registrare un Service Worker malevolo attraverso questo tipo di attacco, il malintenzionato ha bisogno di ospitare i propri script sul server. Considerando una situazione di attacco XSS, quindi con l'inserimento di script all'interno della pagina al fine di rubare dati sensibili, c'è la garanzia che dopo 24 ore la minaccia cesserà di esistere in quanto questo è il tempo limite di vita di un Service Worker (o meno in base alle regole della pagina). Tale rischio potrebbe quindi essere mitigato con una breve vita dei Service Worker, in modo ragionevole così da esserne sfruttate le potenzialità. Si nota che questo strumento, utilizzato per la velocizzazione e il miglioramento dei tempi di risposta delle applicazioni, potrebbe essere programmato per comportarsi similmente a *man-in-the-middle*, intercettando e modificando i messaggi che intercetta.

Si supponga di avere una pagina HTML che registra un Service Worker attraverso uno script (nell'esempio sottostante è *script.js* a riga 2) il quale intercetta ogni richiesta transitante nella pagina.

```
1 <script type="text/javascript">
2     navigator.serviceWorker.register('/script.js').then(
3         function(registration){
4             // Registrazione Service Worker
5         });
6 </script>
7 <script src="./install.js"></script>
```

Una volta completata la registrazione, il Service Worker sarà installato (in questo esempio tramite il file *install.js*).

```
1 if ('serviceWorker' in navigator) {
2     navigator.serviceWorker.register(
3         this.addEventListener('fetch', function(event) {
4             event.respondWith(new Response("Intercepted!"))
5         });
6     ).then(function(registration) {
7         // Service Worker installato correttamente
8     });
9 };
```

In questo esempio compare a video il messaggio di minaccia (Figura 4.2), ma in una situazione ordinaria una possibile vittima non saprà che le informazioni transittanti nella pagina gli stanno venendo sottratte. Finché il Service Worker non sarà disinstallato e non verrà cancellata la cache, ogni richiesta verrà intercettata.

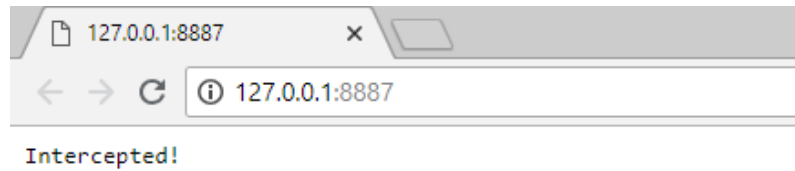


Figura 4.2: Service Worker - Intercettazione delle informazioni tramite SW

4.4 Conclusioni

CHE SCRIVO?

In conclusione, i Service Worker sono uno strumento potentissimo e, come ogni strumento e come visto nei due capitoli precedenti, possono essere usati in modo benevolo o malevolo. Se sfruttati per la caching rendono la navigazione all'interno delle applicazioni molto veloce; dal lato opposto sono uno strumento altrettanto potente da essere usato in malomodo.

CAPITOLO 5

CONCLUSIONI

SCHIFO...

Questa tesi ha cercato di rispondere alla domanda: *"Come faccio ad individuare i rischi di queste APIs e come posso affrontarli?"*. A tal fine è stato preso un campione di tre APIs e sono state condotte delle ricerche e degli studi per rivelare le vulnerabilità di ognuna di esse, trovandone diverse in ognuna di quelle prese in considerazione.

Come già detto durante tutta la tesi tali vulnerabilità possono essere colmate, o comunque evitate; chiaramente sono avvantaggiate a tale riguardo le persone sempre a contatto coi mezzi tecnologici e sicuramente anche le nuove generazioni le quali sono cresciute in mezzo alla tecnologia. Questo è un fattore da tenere in considerazione dato che a mio parere alcuni rischi, come immettere i propri dati o consentire di rilevare la propria posizione in un sito non sicuro, sono prevalenti in certe fasce d'età. Questa considerazione si basa sulla mia esperienza personale e, per questo motivo, non è una dichiarazione generale e oggettiva. Il dato oggettivo sono i problemi legati alle diverse APIs che ho espresso nel corso della mia tesi.

Una ricerca potrebbe essere quella di realizzare uno studio sui problemi della sicurezza in internet nelle diverse fasce d'età, concentrandosi invece che sulle vulnerabilità delle APIs su come cercare di educare più o meno tutti ai rischi del Web.

Ringraziamenti

A conclusione di questo lavoro di tesi mi sento in dovere di porre i miei più sentiti ringraziamenti alle persone che ho avuto modo di conoscere in questo importante periodo della mia vita e che mi hanno aiutato a crescere dal punto di vista professionale, ma soprattutto umano.

Un sincero ringraziamento al mio relatore Dott. Stefano Calzavara che, oltre alla sua vastissima e precisa conoscenza nel campo della sicurezza, mi ha saputo risolvere ogni dubbio, e soprattutto mi ha aiutato a non lasciare stare questo lavoro quando la rassegnazione stava prendendo il sopravvento.

Un grazie anche al mio correlatore Dott. Marco Squarcina che ha saputo darmi consigli fondamentali per riuscire ad andare avanti in alcuni punti della tesi che altrimenti non avrei saputo risolvere da solo.

Non possono mancare in questo elenco tutti i miei compagni di corso con i quali ho trascorso questi anni nelle aule dell'università tra risate, scene uniche e indimenticabili e terrore durante qualche esame, con i quali ho instaurato un rapporto di amicizia che spero duri nel tempo.

Una dose importante di ringraziamenti va a tutte le persone speciali che, dicendomi la giusta dose di parole nei momenti opportuni ma aiutandomi altrettante volte, e soprattutto spendendo gran parte del loro tempo per darmi una mano, sono riuscite a farmi compiere questo percorso. Grazie davvero.

Dovrei scrivere un'altra tesi se scrivessi ad uno a uno tutti quelli che, in diverse forme, mi hanno aiutato a superare i problemi e i brutti periodi di questi anni universitari. Sicuramente uno dei motivi per cui sono arrivato fino a qua è stato anche grazie al continuo "Ce la fai, non mollare!" in risposta ai miei "Lascio stare tutto, non sono capace".

Un grazie importante anche ai miei colleghi di lavoro per la pazienza e la disponibilità dimostratami durante le varie sessioni d'esame.

Per ultimi, ma non meno importanti, i miei genitori. Non so se troverò mai le parole giuste per ringraziarvi, però vorrei che questo traguardo ripagasse, per quanto possibile, tutti i sacrifici che hanno fatto per me, e che fosse un premio anche per loro. Un grande grazie a tutta la mia famiglia per esserci sempre, per sostenermi in ogni cosa io faccia, e sicuramente senza i vostri consigli e le vostre critiche non sarei la persona che sono oggi. Grazie per avermi fatto arrivare a questo punto di arrivo e contemporaneamente di partenza della mia vita.

ELENCO DELLE FIGURE

2.1	Geolocation API - Implementazione pagina d'esempio	4
2.2	Geolocation API - Richiesta permesso posizione	5
2.3	Geolocation API - Rifiuto autorizzazione	5
2.4	Geolocation API - Accettazione autorizzazione	6
3.1	PaymentRequest API - Messaggio di errore generato dal metodo show()	10
3.2	PaymentRequest API - Inserimento credenziali da parte dell'utente	11
3.3	PaymentRequest API - Attacco XSS	12
3.4	PaymentRequest API - Informazioni rubate	13
4.1	Service Worker API - Utilizzo Service Worker in una pagina web	15
4.2	Service Worker - Intercettazione delle informazioni tramite SW	18

BIBLIOGRAFIA

- [1] Mozilla Developer. Features restricted to secure contexts. <https://developer.mozilla.org/en-US/docs/Web/API/Geolocation>.
- [2] Mozilla Developer. Payment request api, mozilla developer. https://developer.mozilla.org/en-US/docs/Web/API/Payment_Request_API.
- [3] Mozilla Developer. Service worker api. https://developer.mozilla.org/en-US/docs/Web/API/Service_Worker_API.
- [4] Mozilla Developer. Geolocation api. https://developer.mozilla.org/en-US/docs/Web/API/Geolocation_API.
- [5] Aurelio De Rosa. An introduction to the geolocation api. 2014. <https://code.tutsplus.com/tutorials/an-introduction-to-the-geolocation-api--cms-20071>.
- [6] Rafay Baloch. Html5 modern day attack and defense vector. <http://www.xss-payloads.com/papers/HTML5AttackVectors.pdf>.
- [7] Mozilla Developer. Payment request api. 2018. <https://w3c.github.io/payment-request/>.
- [8] Google Developers. Introduction to the payment request api. <https://developers.google.com/web/ilt/pwa/introduction-to-the-payment-request-api>.
- [9] Matt Gaunt. Deep dive into payment request api. <https://developers.google.com/web/fundamentals/payments/merchant-guide/deep-dive-into-payment-request>.
- [10] Eiji Kitamura. Bringing easy and fast checkout with payment request api. <https://developers.google.com/web/updates/2016/07/payment-request>.
- [11] Microsoft. Payment request api samples. <https://developer.microsoft.com/en-us/microsoft-edge/testdrive/demos/paymentrequest/>.
- [12] What is payment request? <https://paymentrequest.show/>.

- [13] W3C. Paymentrequest privacy and security considerations. <https://w3c.github.io/payment-request/#privacy-and-security-considerations>.
- [14] Matt Gaunt. Service workers: an introduction. <https://developers.google.com/web/fundamentals/primers/service-workers/>.
- [15] Speedy Wordpress. Guida completa ai service worker javascript. <https://www.speedywordpress.it/guida-completa-ai-service-worker-javascript/>.
- [16] Mozilla Developer. Service worker concepts and usage. https://developer.mozilla.org/en-US/docs/Web/API/Service_Worker_API.
- [17] Angular University. Service workers - practical guided introduction. <https://blog.angular-university.io/service-workers/>.
- [18] jakearchibald Github. Service worker explained. 2017. <https://github.com/w3c/ServiceWorker/blob/master/explainer.md>.
- [19] W3C. Service workers nightly. 2018. <https://w3c.github.io/ServiceWorker/#security-considerations>.
- [20] Erling Ellingsen. Serviceworker is dangerous. <https://alf.nu/ServiceWorker>.