Tesi - Contro API

Daniele Rigon - 857319

2 agosto 2018

Indice

SEZIONE FATTA

1 Geolocation

- viene chiesto all'utente il permesso per usare l'api (unica forma di sicurezza)
- i metodi getCurrentPosition e WatchPosition danno la posizione dell'utente. Possono essere intercettati (?) -> esempio di come prendo la posizione dell'utente (?)

SEZIONE FATTA

2 PaymentRequest

- Usando questa API essa chiama la gestione delle credenziali del browser, caricando una finestra, e quindi non inserendo le credenziali nel DOM
  Su Chrome i dati inseriti sono gestiti dalla compilazione automatica di google `chrome://settings/autofill` -> sicuro
  Su Edge viene usato Microsoft Pay (prima collegandosi all'account Microsoft) -> sicuro
- Non ci sono metodi che prendono la password o le credenziali, vengono gestiti altrove, non si possono intercettare -> sicuro
- può essere aperta una sola pagina con l'API per volta, altrimenti la creazione dell'oggetto PaymentRequest non andrà a buon fine

3 ServiceWorker

- lavora solo su https o localhost, no http o file://
  Usando i ServiceWorker si possono dirottare le connessioni, rispondere in modo diverso a certe richieste e filtrare le risposte. Mentre si usano questi mezzi in modo benevolo, un man-in-the-middle potrebbe usarli per altri scopi. Per evitare ciò si possono registrare i ServiceWorker solo nelle pagine HTTPS, cosi sappiamo che il ServiceWorker non è stato manomesso durante il suo viaggio attraverso la rete. Se non in localhost o https il ServiceWorker non viene registrato-installato-eseguito.
- intercetta le richieste http (puo essere usato male?), puo fare caching ma anche altro. Come viene usato in modo malevolo intercettando dati sensibili?
- sincronizzazione background (contro?)

4 Scenari

- FATTO Geolocation: al "si" dell'utente viene rubata la posizione tramite Geolocation API;
- FATTO PaymentRequest: vedere se ci sono metodi con i quali si può rubare il numero di carta o altre info con PaymentRequest (poco probabile perchè viene usata la gestione delle credenziali Microsoft e Google) -> capire qual è la pagina di "pop-up"(che non è un pop-up) che viene aperta; FATTO vedere se si puo caricare una pagina finta sopra per far si che siamo convinti di comprare una cosa invece compriamo altro (phishing)
- ServiceWorker: /*ServiceWorker usato non per caching ma per intercettare un messaggio, cambiandolo in negativo (simile a man-in-the-middle).*/
  - Attaccante carica su una pagina fb (o un altro sito) uno script js: vedere se può prendere le password e portarle fuori.
  - Pro: vedere se i SW possono migliorare la sicurezza di una pagina, senza essere usati solo per fare

caching.
- Spiegare bene il discorso degli scope. Vedere se si possono caricare SW solo dalla stessa origine, in quel caso non possiamo caricare SW su fb da un'altra pagina, ma possiamo caricarli solo da fb.
I Service Worker obbediscono alla politica della stessa origine?
La registrazione dei Service Worker specifica che i Service Worker devono essere eseguiti nella stessa origine dei loro chiamanti.

Il confronto dell'origine per la ricerca di una registrazione di Service Worker per una richiesta viene specificato per essere una corrispondenza con prefisso più lungo degli URL serializzati, compreso il percorso. (Es. https://example.com/! = https://example.com.evil.com/.) Questo gap di specifiche ci sembra fragile e dovrebbe essere fissato per essere specificato e implementato come effettiva parità di origine , ma al momento non sembra sfruttabile.

Solo i contesti protetti possono registrarsi o utilizzare i Service Worker.

Poiché i SW possono chiamare importScripts per importare script (da qualsiasi altra origine), è una buona idea per gli operatori del sito impostare un'intestazione di risposta Content-Security-Policy sulla risposta JavaScript di ServiceWorker, istruendo il browser su quali fonti di script l'origine considera attendibili. Ciò ridurrebbe la capacità di un attaccante XSS di inserire il proprio codice.
- Se un sito ha una vulnerabilità XSS, l'autore dell'attacco può compromettere definitivamente tale origine per me?
Un attaccante XSS può effettivamente registrare un SW malvagio. Come prima dei SW, XSS è una modalità di attacco molto potente su un'origine web. Per mitigare il rischio che un attacco XSS registri un SW malevolo, il browser richiede che l'URL di registrazione SW provenga dall'origine stessa. Pertanto, per utilizzare un attacco XSS per registrare un SW malevolo, l'utente malintenzionato ha bisogno della capacità aggiuntiva di ospitare i propri script sul server.

Ecco un altro scenario di exploit: se la pagina con una vulnerabilità XSS ha anche un endpoint JSONP, l'utente malintenzionato potrebbe utilizzarlo per (1) bypassare CSP; (Cryptographic Service Provider – libreria software sviluppata da Microsoft) (2) registrare un SW; e (3) chiamata importScripts per importare uno script di terze parti da conservare fino a che:

- gli operatori del sito rilevano e risolvono il problema;
- gli utenti navigano di nuovo sul sito mentre sono online.

In una situazione XSS, il limite della direttiva cache di 24 ore garantisce che un SW malevolo o compromesso sopravviverà a una correzione della vulnerabilità XSS di un massimo di 24 ore (presupponendo che il client sia online). Gli operatori del sito possono ridurre la finestra di vulnerabilità impostando TTL più bassi sugli script SW.
- Vedere se ci sono metodi che limitano alcune funzionalita per mitigare questi problemi.

5 Crawler

Chi usa, e come:

- Geolocation
- PaymentRequest
- ServiceWorker -> principalmente questo e come lo usano