Raport consolidat eveniment cibernetic

Atacuri Drive-by JavaScript

Recent, cercetătorii în securitate au descoperit o metodă inovatoare prin care API-ul de Acces la Sistemul de Fișiere din Google Chrome poate fi exploatat pentru a accesa și manipula fișiere pe sistemele utilizatorilor. Această tehnică de atac nu necesită exploatarea unei vulnerabilități zero-day (0-day), ci se bazează pe folosirea legăturilor simbolice (symlinks) pentru a eluda măsurile de securitate și a obține acces la fișiere sensibile, generând execuție neautorizată de cod.

Informații suplimentare

Vector de atac	❖ **API-ul de Acces la Sistemul de Fișiere:** Construit pentru a permite aplicațiilor web să interacționeze cu sistemul de fișiere local, cu permisiunea utilizatorului.
Scenariul de atac	❖ **Pregătirea Site-ului Rău Intenționat** • Un atacator creează un website cu o funcționalitate aparent legitimă, cum ar fi un editor de cod online sau un "asistent AI". • Site-ul solicită acces la fișiere pentru a oferi funcționalități atractive, cum ar fi salvarea proiectelor pe care utilizatorul le creează. ❖ **Interacțiunea cu Utilizatorul** • Utilizatorul este convins să accepte permisiunile de acces la fișiere. Acest lucru poate fi realizat prin inginerie socială sau prin funcționalități atractive ale aplicației (de exemplu, promisiunea de salvare a fișierelor pe disc). • Când utilizatorul acordă permisiunile cerute, site-ul primește acces la un director sau fișier specific de pe sistemul utilizatorului. ❖ **Manipularea și Exploatarea prin Symlinks** • Atacatorul folosește symlinks pentru a redirecționa accesul de la fișierele inițiale către fișiere sensibile de pe sistemul utilizatorului (ex. fișiere de configurare sau fișiere executabile). • De exemplu, în loc să acceseze un fișier simplu de proiect, aplicația web redirecționează accesul către un fișier sensibil printr-o legătură simbolică.

	• Acest lucru ocolește lista neagră a Chrome, deoarece protecția existentă nu gestionează adecvat legăturile simbolice. ❖ **Execuția de Comenzi Abuzive** • Cu permisiunea de a accesa și modifica fișiere, atacatorul poate insera cod malițios sau poate modifica fișiere esențiale, provocând execuția de comenzi arbitrare în mediul sistemului de operare. • Pe macOS, atacul poate ocoli chiar și mecanismele de protecție, precum Gatekeeper, permițând atacatorilor să ruleze cod neautorizat.
Proof of Concept (PoC)	**Aplicația de Ajutor AI** și **Editorul de Cod "Malefic"** Ambele PoC-uri simulează aplicații legitime, solicitând permisiuni de acces la fișiere. Dacă utilizatorul le acordă, acestea pot executa comenzi arbitrare.
Implicații de securitate	❖ **Ocolirea Gatekeeper pe macOS** Prin exploatarea acestui API, atacatorii pot ocoli protecția Gatekeeper din macOS, obținând posibilitatea de a executa cod neautorizat. ❖ **Fără Dependență de Vulnerabilități de Tip 0-Day** Metoda se bazează pe funcționalitățile existente în Chrome, fiind eficientă fără a necesita vulnerabilități nepatch-uite.
Exemple Practice	❖ **Website rău intenționat** *"<!DOCTYPE html> <html lang="en"> <head> <meta charset="UTF-8"> <title>Evil Code Editor</title> </head> <body> Bine ați venit în Editorul de Cod Online Vă rugăm să acordați permisiuni de acces la fișiere pentru a salva proiectul local. <button onclick="requestFileAccess()">Salvează Proiectul</button> <script> async function requestFileAccess() { try {*

```
      const handle = await window.showDirectoryPicker();
      // Se utilizează manipularea pentru a accesa fișierele sensibile
      await readSensitiveFiles(handle);
    } catch (err) {
      console.error('Accesul la fișiere a fost refuzat', err);
    }
  }

  async function readSensitiveFiles(handle) {
    for await (const entry of handle.values()) {
      if (entry.kind === 'file') {
        const file = await entry.getFile();
        const contents = await file.text();
        console.log('Fișier citit:', contents); // Date sensibile furate
      }
    }
  }
 </script>
</body>
</html>"
```

❖ **Exploatarea prin Symlink**

```
"// Node.js code to create a symbolic link to a sensitive file
const fs = require('fs');
const path = require('path');

// Fișierul pe care atacatorul dorește să-l acceseze (ex. fișier de configurare de sistem)
const sensitiveFile = '/etc/passwd';

// Directorul în care utilizatorul oferă acces
const userDirectory = '/path/to/user/directory';
```

	`` // Crearea unui symlink care va redirecționa utilizatorul către fișierul sensibil const symlinkPath = path.join(userDirectory, 'project_symlink'); fs.symlink(sensitiveFile, symlinkPath, 'file', (err) => { if (err) { console.error('Eroare la crearea symlink-ului:', err); } else { console.log('Symlink creat cu succes!'); } });" ```
Măsuri de protecție	1. **Restricționarea Permisiunilor de Execuție**: Google este sfătuit să limiteze permisiunile de execuție pe fișierele accesate prin API-ul de Acces la Sistemul de Fișiere. 2. **Îmbunătățirea Mecanismelor de Listă Neagră**: Să dezvolte o listă neagră mai robustă în Chrome pentru a detecta și gestiona încercările de evitare prin legături simbolice. 3. **Creșterea Conștientizării Utilizatorilor**: Educați utilizatorii asupra riscurilor asociate cu acordarea permisiunilor de acces la fișiere aplicațiilor web.

Resurse externe

- https://securityonline.info/javascript-drive-by-attacks-new-exploits-without-0-day-in-google-chrome/
- https://developer.mozilla.org/en-US/docs/Web/API/File_System_API
- https://developer.chrome.com/docs/capabilities/web-apis/file-system-access