Same Origin Method Execution, Advanced Aspects

@BenHayak - HiTB 2017
How prevalent is SOME?

Is it real?

- Google Plus - Document
- Yammer.com (Microsoft) - Flash
- WordPress <= 4.5.1 - Flash
- Paypal - Flash
- Salesforce - Documents
- Private bug bounties – Both
What is Same Origin Method Execution?

The ability of (same origin) arbitrary method execution in a web application context.

- Abusing plugin systems such as Adobe Flash.
- Abusing dedicated (usually popup) endpoints.
What is Same Origin Method Execution?

Arbitrary Method Execution Demo!

Execute
What is Same Origin Method Execution?

Callbacks are limited to \[A-Za-z0-9_.] only, otherwise it’d be an XSS.

\(/^[A-Za-z0-9_.]+$/\)

Execute
The SOME Attack

SOME bypasses three obstacles

- **Useless Execution Context** - The callback is not executing where we want (usually in an empty document)
- **Wrong Timing** - The execution triggers too "fast" and only once.
- **Limited Charset** – Only alphanumerics and a dot.
What are the SOME steps?

1. Target
2. Plan
3. Execute
The SOME Attack - Steps

Same Origin Method Execution - Target

Start by choosing a target document/page.

Target
The SOME Attack - Steps

Same Origin Method Execution - Target
The SOME Attack - Steps

Same Origin Method Execution - Plan

Setup the cards right.

– Build a setup of windows before the execution takes place.
The SOME Attack - Steps

Same Origin Method Execution - Plan

Win1 = window.open("/step1.html", "_blank")
Same Origin Method Execution - Execute

Abuse the arbitrary execution at the right place at the right time!

- **Redirect** the documents to designate the execution context.
The SOME Attack - Steps

Same Origin Method Execution - Execute

Same Origin Policy

WIN1

opener === MAIN (still apply)
The SOME Attack - Steps

Same Origin Method Execution - Execute

Same Origin Policy

WIN1

opener === MAIN (still apply)
Same Origin Method ("alert") Execution:
<script>opener.alert()</script>
The ability of (same origin) arbitrary method execution in a web application context.

- Abusing plugin systems such as Adobe Flash.
- Abusing dedicated (usually popup) endpoints.
What are the vulnerable endpoints?

• Active context content types ONLY
  – text/html, text/xml
  – Plugins – such as Adobe Flash Player

• JSONP is **NOT** vulnerable without a chain
  – Must be chained with another bug/injection to become active.
Vulnerable Endpoints

Active context:
URL?callback=Vulnerable

Passive context:
URL?callback=NotVulnerable

Empty Page with callback execution only

```
TypeError: Cannot read property 'Vulnerable' of null
ReferenceError: Vulnerable is not defined
```
Vulnerable Endpoints

Potentially Active Execution Environments:

http://www.example.com/service

Hey, how are you?

Everyone in one place


<table>
<thead>
<tr>
<th>Email</th>
<th><a href="mailto:user@third-party.com">user@third-party.com</a></th>
</tr>
</thead>
<tbody>
<tr>
<td>Password</td>
<td>**********</td>
</tr>
</tbody>
</table>

Accounts

Service Name

Services
Potentially Active Execution Environments:

http://www.example.com/service


Redirect to www.example.com to try to notify a complete event
Hey,
how are you?

http://www.example.com/service

Success!

http://www.example.com/callback_endpoint?callback=succes

Execute callback on the “Same Origin”
To notify a ”success” event
Vulnerable Endpoints

Passive context—direct access tojsonp endpoints.

```
NotVulnerable({"ip": "6..."});
```

Response Headers:

- Access-Control-Allow-Origin: *
- Cache-Control: private
- Content-Encoding: gzip
- Content-Length: 60
- Content-Type: application/javascript; charset=ISO-8859-1

Date: Sun, 05 Mar 2017 14:16:27 GMT
Server: Google Frontend
Vulnerable Endpoints

Not SOME even if you DM me ;)

NotVulnerable({"ip": "6..."});
Vulnerable Adobe Flash Code

Action script code:

```javascript
public function cleanEIString(arg1: String): String {
    return arg1.replace(new RegExp("[^A-Za-z0-9_.]", "gi"), "");
}
```

```javascript
if (loaderInfo.parameters.readyFunction != undefined) {
    ExternalInterface.call(_app.model.cleanEIString,
    ExternalInterface.objectID,
    loaderInfo.parameters.readyFunction);
}
```

yammer.com - Same Origin Method Execution found by Jakub Zoczek
Vulnerable Endpoints - Reasons

Why is this happening?

<title>code ninja</title>
Vulnerable Endpoints - Reasons

Endpoints may be:

• Plugin Systems
• Documents (Pages)
Why developers make vulnerable callback documents?

- Interaction – perform action across context
- The service is not willing to lose open content (e.g. chat), hence uses a Pop-up
- Prompts - “secure delegated access” to third party (OAuth e.g. “Login With”)
- Simple – require less effort
Vulnerable Endpoints - Reasons

Why developers make vulnerable plugins?

- Conventional Cross Context Interaction
  - Respond to events from/to the DOM.
- Large scale of users – arbitrary cb names
  - Example: Adobe Flash plugin calls a "ready" event using a callback - embedding document can start interacting with it.
And... Above all Others

Our Callback is limited to only [A-Za-z0-9._],

What can possibly go wrong?
Breaking it down

- Assembling a SOME Attack
- Arbitrary Callback Execution
Arbitrary Method Execution


Would be exploited as:
Arbitrary Method Execution

How to

This is simple HTML5 colorpicker. Please click at Preview element to see color picker

Uncaught TypeError: Cannot read property 'executeMe' of null
at ColorPicker.php:6
Arbitrary Method Execution

Damn.. Nothing we can target in the callback endpoint

Useless DOM
Targeting

• Choosing your target URL
• Pinpointing a Target Reference
• Targeting Demo
• Targeting tool
Choosing your target
Bounty hunters – alert / opener.alert

Bug hunters –

DOM Navigation:

<table>
<thead>
<tr>
<th>firstElementChild</th>
<th>lastElementChild</th>
</tr>
</thead>
<tbody>
<tr>
<td>nextElementSibling</td>
<td>previousElementSibling</td>
</tr>
<tr>
<td>parentElement</td>
<td></td>
</tr>
</tbody>
</table>

Javascript functions:

– e.g. obj.sales.delOrders
Pinpointing a Target Reference

A reference may look like this:

```javascript
opener.document.body.lastElementChild.previousElementSibling.previousElementSibling.previousElementSibling.previousElementSibling.previousElementSibling.firstElementChild.nextElementSibling.submit
```

Or:

```javascript
opener.javascriptObj.function1
```
Traveling the DOM tree using only allowed characters:

- Start from the body or a valid element id.
  - `document.body` - to reference: `<body>`
  - `divBox1` - to reference: `<div id="divBox1">...
- Use **navigation properties** to reach the target element.
- Select a **method** (click, submit, stepUp, stepDown, select,...).
Trigger a SOME alert using the callback endpoint

Don't Click the Red Button
Trigger a SOME alert using the callback endpoint

Don't Click the Red Button
Trigger a SOME alert using the callback endpoint

Don't Click the Red Button

document.body.firstElementChild

main.firstElementChild.nextElementSibling

main.firstElementChild
Pinpointing a Target Reference

Targeting Demo!
Pinpointing a Target Reference

SOME – Targeting Tool (Chrome Extension)

Reference Generator
Exploit Generator
About

Choose your background color

127.0.0.1 says:
[ ] Same Origin Method Execution Target Reference
[ ] Copy to clipboard: Ctrl+C, Enter

Don't Click the Red Button

[!] Same Origin Method Execution Target Reference
You're welcome to use SOME PoC Generator at: http://www.SOMEAttack.com/SOMEPlayground/SOMEGenerator.html
[+] Target URL:
http://127.0.0.1/SOMEPlayground/index.html
[+] Target Reference:
box.nextElementSibling.nextElementSibling.nextElementSibling.firstElementChild

Range color, then Find SOME

CALLBACK

CALLBACK
Planning

• Designating an Execution Context
Designating an Execution Context

• What we have so far is:
  – **Useless** method execution (nullified DOM)
  – **Target URL** on same origin: 
    http://www.someattack.com/Playground/
  – **Target reference** in that URL: a “Red Button”

  • box.nextElementSibling.nextElementSibling.nextElementSibling.firstElementChild.click
Designating an Execution Context

Craft a page that opens a new window

Win1 = window.open("/step1.html","_blank")

opener === MAIN
Designating an Execution Context

• What happened now?
  – There’s more than one execution context
    • Win1 (window object)
    • Main (window object)
Designating an Execution Context

• Windows can communicate using javascript opener and/or variable references.
  – Unbounded the execution from one useless DOM context (i.e. the callback endpoint)
  – We can use “opener” as a context
    • URL?callback=opener.targetReference.targetMethod
  – Execution Context - DEMO
Execution

- Putting it all together
- SOME – Demo
- SOME POC Generator
Putting it all together - SOME

• We now have:
  – **Useful** method execution

Since we can now **choose the context**!

  – **Target URL** on same origin:
    ```
    http://www.someattack.com/Playground/
    ```

  – **Target reference** in that URL: a “Red Button”
    ```
    box.nextElementSibling.nextElementSibling.nextElementSibling.firstElementChild.click
    ```
Putting it all together - SOME

• Same Origin Policy Problem: ❌
  – The “Target URL” must be rendered in the opener context document **BEFORE** Execution!

• Solution: ✔️
  – Redirect MAIN to Target Page.
  – Redirect Popup to Callback Endpoint AFTER timeout.
The Execution success is based on:

Redirecting the document location of any of the execution contexts won’t break the memory reference of the popup (opener reference)
• Putting it all together - DEMO
• Why work hard?
Same Origin Method Execution - Basic Exploit Generator

Generate a PoC using this tool

SOME PoC Exploit Generator

Vulnerable instance: http://vulnerable.com/callback_endpoint?otherparams=values
Vulnerable parameter: callback
Target endpoint: http://vulnerable.com/target_endpoint
Target method reference:

document.body.firstElementChild.etc.etc.submit
Note: Add opener/parent reference (depending on the vulnerable endpoint)
Delay in milliseconds: 3000

Generate Exploit

Download

Currently the SOME exploit generator only supports callbacks executed by a GET Response. If you tackle the need to add support for a POST endpoint feel free to contact me in twitter @BenHayak
SOME Extras

- Setting data using callbacks
- Takeover Wordpress using SOME
- XSS using the SOME technique.

- Time ticking - Thank you!
Setting data using callbacks

Possible SOME Methods vary based on Elements

- **click** (click buttons/anchors)
- **submit** (submit forms)
- **stepUp/stepDown** (inc/dec count fields)
- **setRangeText** (set input elements values)
- **reset** (clear forms)
- **etc**
Setting data using callbacks

Setting a value in input boxes:

- **Callback methods**
  - `setRangeText` (set input elements values)
  - `reset` (clear forms)
- A ”Failure” of CSRF may set input values in forms (useless? Or is it?)
Takeover Wordpress using SOME

Adding Wordpress Admins

Mathias Karlsson’s PoC (@avlidienbrunn)

- CSRF failure – fills in values in Wordpress admin panel
  - https://domain/wordpress/wp-admin/user-new.php

- SOME – hijack four method executions
  - reset, setRangeText x2, click
Takeover Wordpress using SOME
Its all about the technique (XSS)

Even XSS could use SOME

JavaScript Alert
XSS

OK
Sanitized injections are not safe!

- **Callback injection:**
  - `Injection.Injection('static-value')`
    - Leads to *Same Origin Method Execution*

- **Argument Injection:**
  - `opener.static.Function('Injection')`
    - May lead to *less-trivial XSS bugs.*
When endpoint’s opener is null, execution fails.

- opener.navigateToURL('Injection')
Fixed opener vulnerable functions calls:

- opener.navigateToURL('Injection')
- opener.document.getElementById('id').innerHTML = 'Injection'
- opener.document.write('Injection')
- opener.document.writeln('Injection')
- opener.location.href = 'Injection'
- Etc...

Fixed top/parent references can be xssed using iframes yet only when the CB endpoint is missing Fame-Busting protection.
• Setup the opener reference to a controlled context.
• Redirect the context to target page.
• Redirect the popup to the vulnerable endpoint that will trigger the XSS.
• Static Callbacks
  – no external control
• White-listing
  – Accept only authorized callbacks
• Cross-Domain Messaging
  – Use postMessage for interaction (secure your channels!)
Thank you!

beer.opener.please.TY.Folks()