How to Secure the Electron Container for Capital Markets

FINOS Host
James McLeod, FINOS Director of Community

ChartIQ Presenters
Kris West, Director Solutions Engineering
Ian Mesner, Chief Architect

May 2020
The Electron Dichotomy
Do we need to secure Electron?
Business Applications on the Web

Web developers know what they're doing:
- SQL Injection
- Cross-site scripting
- etc.

Common vulnerabilities and best practices analyzed and published by organisations:
- OWASP (top 10)
- Carnegie Mellon Uni. SEI’s CERT program
- US Department of Homeland Security's Cyber & Infrastructure Security Agency
# Web Application Security

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>1</strong></td>
<td><strong>Injection</strong></td>
<td>Buffer overflow, SQL injection; parameters</td>
</tr>
<tr>
<td><strong>2</strong></td>
<td><strong>Broken Authentication</strong></td>
<td>Credential theft through snooping or brute force</td>
</tr>
<tr>
<td><strong>3</strong></td>
<td><strong>Sensitive Data Exposure</strong></td>
<td>Storing data without proper safeguards</td>
</tr>
<tr>
<td><strong>4</strong></td>
<td><strong>XML External Entities</strong></td>
<td>Remote code execution of remote xml resources</td>
</tr>
<tr>
<td><strong>5</strong></td>
<td><strong>Broken Access Control</strong></td>
<td>Gaining access to restricted systems.</td>
</tr>
<tr>
<td><strong>6</strong></td>
<td><strong>Security Misconfiguration</strong></td>
<td>Insufficient access control</td>
</tr>
<tr>
<td><strong>7</strong></td>
<td><strong>Cross Site Scripting</strong></td>
<td>Remote code execution due to code as data</td>
</tr>
<tr>
<td><strong>8</strong></td>
<td><strong>Insecure Deserialization</strong></td>
<td>Data retrieval as a point of attack or remote code execution, etc.</td>
</tr>
<tr>
<td><strong>9</strong></td>
<td><strong>Using components with known vulnerabilities</strong></td>
<td>Failure to audit dependencies</td>
</tr>
<tr>
<td><strong>10</strong></td>
<td><strong>Insufficient Logging and Monitoring</strong></td>
<td>Failing to audit access. Extraneous functionality exploit of logging sensitive info</td>
</tr>
</tbody>
</table>

[https://owasp.org/www-project-top-ten/](https://owasp.org/www-project-top-ten/)
Web Application Security: the Comfort of the Sandbox.

Browsers are designed to execute remote, untrusted code.

- Restricted operating system APIs
- Integrated sandbox
- Site isolation
- Web security policies
But isn't Electron based on a web browser?
This new class of software introduces new risks to manage.

Integrating applications from multiple sources
Bringing a variety of technologies from a variety of software providers onto your desktop can be risky if not well managed.

Web technology without the browser
The arrival of web technology on the desktop, outside of the browser, compounds the already complex challenge of desktop security.

Communications and Interop
The goal of the smart desktop is to promote communication and interoperability between applications and micro-frontends, but without compromising security.
Security for the Smart Desktop

SYSTEM SECURITY

Custom Desktop Services

Electron

Operating System

CONTENT SECURITY

Applications

App 1

App 2

App 3
# Security for the Smart Desktop

<table>
<thead>
<tr>
<th>System Security</th>
<th>Content Security</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Input Validation</strong></td>
<td><strong>Communication</strong></td>
</tr>
<tr>
<td>Buffer overflow, SQL injects, params, XSS, deserialization</td>
<td>Ability to listen to messages meant for others.</td>
</tr>
<tr>
<td><strong>Broken Authentication</strong></td>
<td><strong>Storage</strong></td>
</tr>
<tr>
<td>Credential theft through snooping or brute force</td>
<td>Unauthorized access to persisted data</td>
</tr>
<tr>
<td><strong>Broken Authorization</strong></td>
<td><strong>Runtime</strong></td>
</tr>
<tr>
<td>Gaining access to restricted systems or elevated rights</td>
<td>Information about what other applications are running and the current user or platform</td>
</tr>
<tr>
<td><strong>Dependency exploits</strong></td>
<td><strong>Configuration</strong></td>
</tr>
<tr>
<td>Failure to audit dependencies</td>
<td>Gain information about or modify access to runtime through config</td>
</tr>
</tbody>
</table>
Electron Security Checklist

1. Only load secure content (https, wss, sftp)
2. Disable the *Node.js integration* in all renderers
3. Enable *context isolation* in all renderers
4. Use `session.setPermissionRequestHandler()` to control what desktop API permissions remote content has access to
5. Do not disable *webSecurity*
6. Define a *Content-Security-Policy*
7. Do not set `allowRunningInsecureContent` to true
8. Do not enable *experimental features*
9. Do not use `enableBlinkFeatures`
10. `<webview>`: Do not use `allowpopups`
11. `<webview>`: Verify options and params
12. Disable or limit *navigation*
13. Disable or limit *creation of new windows*
14. Do not use `openExternal` with untrusted content
15. Disable the *remote module*
16. Filter the *remote module* (if you can't disable it)
17. Use a *current version* of Electron

[https://www.electronjs.org/docs/tutorial/security](https://www.electronjs.org/docs/tutorial/security)
Implement the Checklist

Content Security → Interoperability

System security → Desktop APIs
Back to the drawing board...

Handle secure, trusted code differently than content from untrusted sources

**Desktop Services**
- Build microservices for the desktop
- Implement interprocess comms

**Policy-based Security**
- Enable/Disable Electron APIs via config
- Principle of least privilege (POLP)
Announcing the Secure Electron Adapter

At ChartIQ, we believe in both:

- Open Source software
- Collaboration

Secure Electron Adaptor (SEA)

- Adheres to Electron’s own security recommendations by design.
- Provides support for policy-based security, making it much easier to work with
- Implements inter-process communication, filtered by that policy-based security
Secure Electron Adapter Next Steps

Where can I get it?

github.com/finos/secure-electron-adapter

Quick-start project

github.com/finos/sea-quick-start
- Minimal Electron app using SEA
- Based on the Electron quick start guide
  https://www.electronjs.org/docs/tutorial/quick-start
Secure Electron Adapter

SEA is config-driven

- `/public/manifest-local.json`
  Used to configure:
  - Main process
    - Loaded from a remote location
    - Can be a visible window or service
    - Can have content preloaded into it
  - Other 'components'
    - Also loaded from a remote location
    - Can have permissions specified
  - Electron adapter settings
    - Such as 'trusted' preloads
      (ideal for creating clients for your own desktop services)
What does SEA not do?

SEA doesn't include:

- Detailed desktop services
- Ready-made UI components
- Full solutions to the 'Big 8' (Rather its focused on secure foundation on which to build these)
- Support services

Need more?

FINSEMBLE

A fully featured Smart Desktop for finance

www.finsemble.com
Enterprise-secured container

- Leveraged use of Chromium, Electron & SEA
- Layers of protection, least-privilege by default
- Only vendor to provide 100% full source to clients
- Third-party security assessment by Bishop Fox

COMPREHENSIVE REPORT
CHARTIQ, INC.
FINSEMBLE HYBRID APPLICATION ASSESSMENT 2019
03.18.2019

BISHOPFOX
Takeaways

- Electron isn't designed to be secure out-of-the-box
- Building a 'Smart Desktop' leveraging web deployment create new risks to manage
- Electron project cares deeply about the security of your applications
  - 17-point security checklist for securing untrusted content
  - Implementing the checklist eliminates many of the benefits of using the container
- Policy-based security and Desktop Services provide the answer to practical development
- Secure Electron Adapter provides an ideal foundation to build on
Thank you for attending.

Contact us at info@chartiq.com