Building Least Privileged Web Applications with Node.js

Deian Stefan

Joint work with Devon Rifkin, Annie Liu, Christian Garcia Almenar
Oh ye web apps…

U.S. Postal Service data breach may compromise staff, customer details

United website breach let fliers see each others’ private data

Dental reports data breach

Snapchat security breach affects 4.6 million users. Did Snapchat drag its feet on a fix?

WellPoint email glitch puts colonoscopy test in the subject line

Yahoo Password Breach Puts SQL Injection in the Crosshairs

Adobe customer data breached - login and credit card data probably stolen, all passwords reset

Credit agency mistakenly sends 300 confidential reports to Maine woman

Man Finds Easy Hack to Delete Any Facebook Photo Album

Massive Data Breach

University of Chicago data breach employee and student data


Target Confirms Unauthorized Access To Payment Card Data in U.S. Stores

Over 99 percent of About.com lists exposed in a data breach

50,000 Uber driver names, licenses and Social Security numbers put online votes at risk

Credit Card Breach Los Angeles International Police Investigating At LAX Tom Bre
Recipe for disaster

1. Apps handle sensitive user data

2. Programming models follow the principle of most privilege ▶️ ad-hoc security mechanisms

3. Developers write buggy code
Example: ghost.org

• Production blog app’s data model:

<table>
<thead>
<tr>
<th>Blog posts:</th>
<th>Users:</th>
</tr>
</thead>
<tbody>
<tr>
<td>id</td>
<td>authors</td>
</tr>
<tr>
<td>0</td>
<td>alice</td>
</tr>
<tr>
<td>1</td>
<td>bob, claire</td>
</tr>
<tr>
<td></td>
<td>claire</td>
</tr>
</tbody>
</table>

➤ Sensitive data: unpublished posts, passwords, emails

• App functionality:

➤ List all posts, show post, show user profile, ...
Example: ghost.org

• Production blog app’s data model:

<table>
<thead>
<tr>
<th>id</th>
<th>authors</th>
<th>pub?</th>
<th>title</th>
<th>body</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>alice</td>
<td>FALSE</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>1</td>
<td>bob, claire</td>
<td>TRUE</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>user</th>
<th>password</th>
<th>email</th>
<th>name</th>
</tr>
</thead>
<tbody>
<tr>
<td>alice</td>
<td>0dea48ff</td>
<td>...</td>
<td>A. Alyokhina</td>
</tr>
<tr>
<td>bob</td>
<td>15a8ccd8f</td>
<td>...</td>
<td>B. Digital</td>
</tr>
<tr>
<td>claire</td>
<td>v3991e5</td>
<td>...</td>
<td>C. Hopper</td>
</tr>
</tbody>
</table>

➤ Sensitive data: unpublished posts, passwords, emails

• App functionality:

➤ List all posts, show post, show user profile, ...
### Example: ghost.org

**Production blog app’s data model:**

<table>
<thead>
<tr>
<th>id</th>
<th>authors</th>
<th>pub?</th>
<th>title</th>
<th>body</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>alice</td>
<td>FALSE</td>
<td>…</td>
<td>…</td>
</tr>
<tr>
<td>1</td>
<td>bob, claire</td>
<td>TRUE</td>
<td>…</td>
<td>…</td>
</tr>
</tbody>
</table>

#### Blog posts:

- **Sensitive data:** unpublished posts, passwords, emails

#### Users:

<table>
<thead>
<tr>
<th>user</th>
<th>password</th>
<th>email</th>
<th>name</th>
</tr>
</thead>
<tbody>
<tr>
<td>alice</td>
<td>0dea48ff</td>
<td>…</td>
<td>A. Alyokhina</td>
</tr>
<tr>
<td>bob</td>
<td>15a8ccd8f</td>
<td>…</td>
<td>B. Digital</td>
</tr>
<tr>
<td>claire</td>
<td>v3991e5</td>
<td>…</td>
<td>C. Hopper</td>
</tr>
</tbody>
</table>

#### App functionality:

- List all posts, show post, show user profile, …
• By default, handlers are most privileged
  ➤ Unrestricted access to storage, fs, net, child process, ...

• Developers retrofit security on top
App architecture

- By default, handlers are most privileged
  - Unrestricted access to storage, fs, net, child process, ...
- Developers retrofit security on top
• By default, handlers are most privileged
  ➤ Unrestricted access to storage, fs, net, child process, ...

• Developers retrofit security on top
App architecture

- By default, handlers are most privileged
- Unrestricted access to storage, fs, net, child process, ...
- Developers retrofit security on top

App services

Storage

- • By default, handlers are most privileged
- • Unrestricted access to storage, fs, net, child process, ...
- • Developers retrofit security on top

Dispatcher

Show post

Update profile

read: function read(options) {
  var attrs = ['id', 'slug', 'status'],
  data = _.pick(options, attrs);
  options = _.omit(options, attrs);
  // only published posts if no user is present
  if (!options.context || !options.context.user) {
    data.status = 'published';
  }

  if (options.include) {
    options.include = prepareInclude(options.include);
  }

  return dataProvider.Post.findOne(data, options).then(function (result) {
    return {posts: [result.toJSON()]};
  });
}

return Promise.reject(new errors.NotFoundError('Post not found.'));

edit: function edit(obj, options) {
  var editOperation = function () {
    return dataProvider.User.edit(obj, options);
  }

  if (options.include) {
    options.include = prepareInclude(options.include);
  }

  return util.raises.checkObject(obj, 'title').then(function (data) {
    // Edit operation
    editOperation = function () {
      return dataProvider.User.edit(data, options);
    }

    if (result) {
      return {users: [result.toJSON()]};
    }

    return Promise.reject(new errors.NotFoundError('User not found.'));
  });
}

// Check permissions
return canThis(options, edituser).then(function () {
  // if roles aren't in the payload, proceed with the edit
  if (!(!data.users[0].roles & data.users[0].roles[0])) {
    return editOperation();
  }

  var role = data.users[0].roles[0],
    roleId = parseInt(role.id || role.id),
    edituserid = parseInt(options.id || id);

  return dataProvider.User.findOne
    (id, options.context.user, status: 'all'),
     (include: ['roles'])
    .then(function (contextuser) {
      var contextroleid = contextuser.related('roles').toJSON()[0].id;

      if (role.id !== contextroleid && edituserid !== contextuserid.id) {
        return Promise.reject(new errors.NoPermissionError('You cannot change your own role.'));
      }

      return dataProvider.User.findOne
        (role: 'Owner'),
        function (owner) {
          if (contextuser.id !== owner.id) {
            if (edituserid === owner.id) {
              if (owner.related('roles').set(0).id !== role.id) {
                return Promise.reject(new errors.NoPermissionError('Cannot change Owner\'s role.'));
              }
            }
            else if (roleid === contextroleid) {
              return canThis(options.context).assign.role(role),
              function () {
                return editOperation();
              }
            }
          }
        }

      return editOperation();
    });

  return Promise.reject(new errors.NoPermissionError('You do not have permission to edit this user');
});
Problem with existing approach

• Missing single security check ➔ vulnerability
  ➤ E.g., ghost.org exposed passwords and drafts

• Checks don’t always extend to third-party libs
  ➤ Libraries may expose vulnerabilities

• Damage due to vulnerabilities can be grave
  ➤ All code runs with same privilege
  ➤ E.g., st library didn’t handle “.” correctly ➔ leaked files
Problem with existing approach

• Missing single security check ➔ vulnerability
  ➤ E.g., ghost.org exposed passwords and drafts

• Checks don’t always extend to third-party libs
  ➤ Libraries may expose vulnerabilities

• Damage due to vulnerabilities can be grave
  ➤ All code runs with same privilege
  ➤ E.g., st library didn’t handle “.” correctly ➔ leaked files
Problem with existing approach

• Missing single security check ➫ vulnerability
  ➤ E.g., ghost.org exposed passwords and drafts

• Checks don’t always extend to third-party libs
  ➤ Libraries may expose vulnerabilities

• Damage due to vulnerabilities can be grave
  ➤ All code runs with same privilege
  ➤ E.g., st library didn’t handle “..” correctly ➫ leaked files
Change how developers build apps

- Minimize trusted computing base (TCB)

  - Make security robust to bugs in most code

- **Challenge**: perennial goal in computer security

  - Can we actually do this?
Can we do this for Node.js?

```bash
 thinly:~ d$ node
 > Array(16)
 [ , , , , , , , , , , , , , , , ]
 > Array(16).join("wat")
 'watwatwatwatwatwatwatwatwatwatwatwatwatwatwatwat'
 > Array(16).join("wat" + 1)
 'watlwatlwatlwatlwatlwatlwatlwatlwatlwatlwatlwatlwatlwatlwatlwatlwatlwatlwatl'
 > Array(16).join("wat" - 1) + " Batman!"
 'NaNaNNaNNaNNaNNaNNaNNaNNaNNaNNaNNaNNaNNaNNaNNaNNaNNaNNaNNaNNaNNaNNaNNaNNaNNaNNaNNaNNaNNaNNaNNaNNaNNaNNaNNaNNaNNaNNaNNaNNaNNaNNaNNaNNaNNaNNaNNaNNa
 Batman!'
 > []
```
Turns out...

JavaScript is well-suited for executing untrusted code

...if you just look at it just right
Node.js apps at a high level

- Code runs in (V8) contexts
  - Global object + execution stack
- Language (EcmaScript) doesn’t have built-in IO
- Embedder (Node.js) attaches props to global object to provide IO
  - E.g., fs, http, net, process, etc.
Looking at it just right

- Expose V8 contexts as isolation primitives
  - New context has separate heap: no access to fs, etc.
- Execute untrusted code in new contexts
  - E.g., run different request handlers in isolation
Providing useful APIs to ctxs

• **By default, code has minimal privileges**
  ➤ Can’t do anything except execute “pure” JavaScript

• **Problem: real code needs to perform IO**
  ➤ Fresh contexts do not have access to Node.js APIs

• **Solution: expose message passing primitives**
  ➤ Untrusted context can send and receive messages to and from main/parent context
  ➤ To perform IO: ask parent context to do it
Providing useful APIs to ctxs

- By default, code has minimal privileges
  - Can’t do anything except execute “pure” JavaScript

- Problem: real code needs to perform IO
  - Fresh contexts do not have access to Node.js APIs

- Solution: expose message passing primitives
  - Untrusted context can send and receive messages to and from main/parent context
  - To perform IO: ask parent context to do it
Virtualization w/ message passing

- Function calls ➔ messages to parent context
  - Parent can perform checks before (and after) calling actual function
  - E.g., implementing synchronous file read:

```javascript
ctx1.js

fs.readFileSync = function (fname, opts) {
  return _espectro.RPC('fs:readFileSync')(fname, opts);
}

// ...

main.js

var ctx1 = new Ctx('ctx1.js');
ctx1.onrpc('fs:readFileSync', function (fname, opts) {
  if (!(fname in _allowed)) throw 'denied!';
  var res = fs.readFileSync(fname, opts);
  return res;
});
```
Virtualized Node.js libraries

- In untrusted contexts: core libraries using message passing
- In main context: hooks library used to register pre/post hooks for each function call
  - High-level policies implemented atop hooks
Different architectures

- **App / context**

- **Controller / context**

- **Request / context**
Different architectures

• App / context

• Controller / context

• Request / context
Different architectures

- **App / context**

- **Controller / context**

- **Request / context**
By default, handlers are least privileged

- Restricted access to storage, fs, net, etc. according to current user and app functionality
• By default, handlers are least privileged
  ➢ Restricted access to storage, fs, net, etc. according to current user and app functionality
Least privileged ghost.org

- By default, handlers are least privileged
  - Restricted access to storage, fs, net, etc. according to current user and app functionality
Consequence of design

- Policy can be declarative specified in main context
- Policy extends to third-party libraries
  - Policy applies to request handler and any library it uses
- DAC policies can limit damage due to bugs
  - Fine-grained/user request limits attack surface
- MAC policies can prevent damage due to bugs
  - MAC enforces policy even once code has access to data
Beyond access control

- Virtualization layer can be used for:
  - Transparently encrypting/decrypting files
  - Caching files, DB queries, responses, etc.
  - Rewriting HTML to add CSRF tokens
  - MACing cookies
  - Setting custom headers (e.g., CSP, SRI, etc.)
  - ...

Conclusions

• Today: writing insecure code is the default
  ➢ Building least-privileged apps is notoriously difficult

• App-level virtualization can be used to protect app from itself and third-party code
  ➢ Policy must allow functionality for it to be available
  ➢ Can build least privileged apps more easily
Thanks!

Availability: this summer from gitstar.com

Follow up: @deiandelmars