



SE4030 – Secure Software Development

Assignment 01

Name	Student ID
Jayasinghe J.A.M.P.	IT21268144
Wickramasinghe W.A.R.M.	IT21233494
Bandara S.Y.T.D.	IT21185052
Shabry S.M.	IT21296796

Table of Contents

About.....	3
Testing Tools.....	3
Vulnerabilities - Manual testing.....	4
NoSQL Injection Prevention	4
Denial of Service (DoS) via Excessive Request Flooding	5
Insecure Direct Object References (IDOR)	7
Regular Expression Denial Of Service (REDOS Attack).....	8
Vulnerabilities – Detected by Tools.....	10
Sensitive Data Exposure	10
Denial of Service (DoS) via Uncontrolled Resource Consumption	11
Using Weak Pseudorandom Number Generators (PRNGs).....	12
Permissive Cross-Origin Resource Sharing (CORS) Policy.....	13
Unverified Remote Resource Inclusion	14
Version Disclosure	15
Cross-site Scripting (XSS)	16
Add SSL Certificate.....	18
OAuth connect-based grant.....	19
Other vulnerabilities	20
Best Practices.....	22
Security-first Design & Development (Security by Design)	22
Use of Environment Variables for Sensitive Information.....	22
Code Reviews and Pair Programming	23
DevSecOps Approach.....	24
CI/CD with Security Gates	24

About

The OneTel Mobile Accessories Shop replaces its manual system with automation to benefit the business by improving customer experiences and problem-solving. The developed automation allows the following features:

- **QR System for Item Specification:** It would be easy for a customer to reach the specifications of any product used, and they will not need to spend much time surfing the website or looking somewhere else through advertising. That will save time and resources both for the customer and the seller.
- **Live Chat for After-Sales Support:** In case of problems with their products, the customer can easily go into live chat with representatives of the store, and problems may get sorted without them actually having to visit the store. It saves time and increases convenience.
- **Order Tracking:** There will be a tracking system put in place that will keep customers updated about the status of their orders in real time. It provides full transparency with less confusion once the purchase has been made.
- **Rental Management:** The system shall require the customer to upload his or her ID and personal information to reduce losses of rental items at least the store shall track them. In cases of loss, the accountability shall fall on the customer with ease.

Generally, this automated system tends to bring OneTel's operations up to date by adding efficiency, customer satisfaction, and security.

Testing Tools

- Sonar Qube
- OWASP Zap
- Snyk

Vulnerabilities - Manual testing

NoSQL Injection Prevention

The application was vulnerable to NoSQL injection attacks. This issue arose because unvalidated user inputs were directly passed into the database query, allowing attackers to inject malicious commands.

```
58 // Login route
59 router.post('/login', (req, res) => {
60
61     User.findOne({ email: req.body.email })
62     .then(user => {
63         if (user) {
64             if (bcrypt.compareSync(req.body.password, user.password)) {
65                 const payload = {
66                     _id: user._id,
67                     first_name: user.first_name,
68                     last_name: user.last_name,
69                     email: user.email
70                 };
71                 let token = jwt.sign(payload, process.env.SECRET_KEY, { expiresIn: 1440 });
72                 res.send(token);
73             } else {
74                 res.status(401).json({ error: 'Invalid password' });
75             }
76         } else {
77             res.status(404).json({ error: 'User not found' });
78         }
79     })
80     .catch(err => {
81         res.status(500).json({ error: 'Server error' });
82     });
83 });
```

Fix Implemented:

- Input validation has been added to prevent NoSQL injection. Specifically, the login route now checks if the email input is of the correct type (string) before querying the database. This ensures that only valid strings are used in the query, blocking any NoSQL injection attempts.

```

58 // Login route
59 router.post('/login', (req, res) => {
60
61     // Prevent NoSQL injection by validating input type
62     if (typeof req.body.email !== 'string'){
63         return res.status(400).json({ error: 'Invalid input type' });
64     }
65
66     User.findOne({ email: req.body.email })
67     .then(user => {
68         if (user) {
69             if (bcrypt.compareSync(req.body.password, user.password)) {
70                 const payload = {
71                     _id: user._id,
72                     first_name: user.first_name,
73                     last_name: user.last_name,
74                     email: user.email
75                 };
76                 let token = jwt.sign(payload, process.env.SECRET_KEY, { expiresIn: 1440 });
77                 res.send(token);
78             } else {
79                 res.status(401).json({ error: 'Invalid password' });
80             }
81         } else {
82             res.status(404).json({ error: 'User not found' });
83         }
84     })
85     .catch(err => {
86         res.status(500).json({ error: 'Server error' });
87     });
88 });

```

Denial of Service (DoS) via Excessive Request Flooding

A Denial of Service (DoS) attack can occur when an attacker sends an overwhelming number of requests to the server in a short period of time. This type of attack, known as Request Flooding, can degrade the server's performance or cause it to become unresponsive by exhausting resources like CPU, memory, or bandwidth.

To prevent such attacks, a rate-limiting mechanism can be applied, restricting the number of requests a client can send within a specific timeframe. If the limit is exceeded, subsequent requests from the same client are blocked for a set duration.

Fix Implemented:

- The implemented solution involves using the rate-limiting middleware from Express to protect the application from excessive request flooding. By setting a limit of 100 requests per IP address within a 15-minute window, the server can prevent a potential attacker from overwhelming the application with rapid, repeated requests. If the threshold is reached, the client will be blocked temporarily.

```
1  const bodyParser = require('body-parser');
2  const express = require('express');
3  const mongoose = require('mongoose');
4  const cors = require('cors');
5  const app = express();
6  app.set('trust proxy', 1);
7  const rateLimit = require('express-rate-limit');
8
9  //prevent DoS attack
10 const limiter = rateLimit({
11   windowMs: 15 * 60 * 1000, // 15 minutes
12   max: 100, // Limit each IP to 100 requests per windowMs
13   message: "Too many requests, please try again later."
14 });
15
16 app.use(limiter);
17
18 const PORT = process.env.PORT || 8070;
```

Insecure Direct Object References (IDOR)

IDOR: An application discloses an internal implementation object reference-for example, a file or record in the database-without proper authorization checks on it. This can be tampered by the attacker to provide unauthorized data access.

Fix Implemented:

- Implemented in React a ProtectedRoute component, using JWT and the role in the payload of the token to check if there was admin privilege, hence give or not access to certain routes.
- Added extra protection to the back-end routes by verification of a token and checking user's role prior to serving protected resources.

```
onatel > client > src > JS ProtectedRoute.js > ProtectedRoute
1  import React from 'react';
2  import { Navigate } from 'react-router-dom';
3  import jwt_decode from 'jwt-decode';
4
5  const ProtectedRoute = ({ children, roleRequired }) => {
6
7      const token = localStorage.getItem('token');
8
9      // If there is no token, redirect to login
10     if (!token) {
11         return <Navigate to="/login" />;
12     }
13
14     try {
15         // Decode the token to check user role or expiration
16         const decoded = jwt_decode(token);
17
18         // Check if the token has expired
19         const currentTime = Date.now() / 1000; // in seconds
20         if (decoded.exp < currentTime) {
21             // If expired, clear token and redirect to login
22             localStorage.removeItem('token');
23             return <Navigate to="/login" />;
24         }
25
26         // If the roleRequired is "admin", check if the user has the admin role
27         if (roleRequired === 'admin' && decoded.role !== 'admin') {
28             return <Navigate to="/" />;
29         }
30
31         // If everything checks out, allow access to the component
32         return children;
33     } catch (err) {
34         console.error('Error decoding token:', err);
35         return <Navigate to="/login" />;
36     }
37 }
38
39 export default ProtectedRoute;
```

Route>Admin.js

```
router.post('/admin/login', verifyToken, (req, res) => {
  Admin.findOne({ email: req.body.email })
    .then(admin => {
      if (admin) {
        console.log('Stored Hashed Password:', admin.password);
        console.log('Plain Password:', req.body.password);

        // Compare the hashed password with the plain password
        if (bcrypt.compareSync(req.body.password, admin.password)) {
          const payload = {
            _id: admin._id,
            email: admin.email,
            role: "admin"
          };

          // Generate JWT token
          let token = jwt.sign(payload, process.env.SECRET_KEY, { expiresIn: 1440 });
          res.json({ token: token }); // Send token to client
        } else {
          res.status(401).json({ error: 'Invalid password' });
        }
      } else {
        res.status(404).json({ error: 'Admin not found' });
      }
    })
    .catch(err => {
      res.status(500).json({ error: 'Server error' });
    });
});
```

Regular Expression Denial Of Service (REDOS Attack)

Regular Expression Denial of Service (REDOS) attacks exploit the inefficiencies in regular expressions (regex). When poorly designed regular expressions are used in input validation, especially with nested quantifiers, malicious actors can craft inputs that take an exponentially long time to process. An attacker can use this vulnerability to cause a program using a regular expression to hang for a very long time. This causes the system to use excessive CPU resources, effectively creating a Denial of Service (DoS) scenario.

Code Vulnerability- Email Validation Using Regex (REDOS Risk):

In this snippet, an email is validated using a regular expression. While this pattern seems simple, if crafted with certain input, it can lead to REDOS. For example, certain combinations of characters can cause the regex engine to take much longer to evaluate the input, resulting in performance degradation or system unavailability.


```

38
39 // Validate email format without validator(ReDos Attack)
40
41 const emailRegex = /^[^\s@]+@[^\s@]+\.[^\s@]+$/;
42 if (!emailRegex.test(email)) {
43   toast.error('Invalid email address');
44   return;
45 }
46

```

Why This Code Is Vulnerable:

1. **Complexity of Regex:** The regex includes multiple quantifiers (+), which can cause catastrophic backtracking if the input is sufficiently complex.
2. **Lack of Input Length Limitation:** Without proper constraints on the input size, the risk increases if the attacker inputs an abnormally large or specifically designed string.

Preventing REDOS: Using Validator Library:

By switching to a library like validator, the risk of REDOS is minimized because this library's functions are optimized for safety and performance. The built-in isEmail() function does not rely on complex regex patterns that are vulnerable to REDOS attacks.

```

30 // Validate email format
31
32 // Validate email using validator
33
34 if (!validator.isEmail(this.state.email)) {
35   toast.error('Invalid email format');
36   return;
37 }
38

```

How This Prevents REDOS:

- **Pre-optimized Functions:** The validator library is built to avoid inefficiencies and vulnerabilities like REDOS by internally handling edge cases.
- **No Regex Complexity:** Instead of manually handling regex, the library uses secure methods that are designed to handle all email formats efficiently without the risk of backtracking or excessive computation time.

By migrating from manual regex to a well-tested library like validator, your email validation process becomes not only more reliable but also significantly more secure against REDOS attacks.

Vulnerabilities – Detected by Tools

Sensitive Data Exposure

sensitive information, such as authentication credentials (like MongoDB passwords), is hardcoded in the source code. This practice poses a significant security risk, especially when the code is shared or deployed across multiple environments. If an attacker gains access to this information, they can potentially exploit it to compromise the database, access sensitive data, and undermine the security of the entire application.

Make sure this MongoDB database password gets changed and removed from the code. [🔗](#)

MongoDB database passwords should not be disclosed [secrets:S6694](#)

Line affected: L56 • Effort: 30min • Introduced: 24 hours ago • [🔒 Vulnerability](#) • [🚫 Blocker](#)

○ Open ▾

Not assigned ▾

cwe [+](#)

Where is the issue?

Why is this an issue?

How can I fix it?

Activity

More info

54

55

56

57

58

59

60

61

62

63

64

65

66

67

68

69

70

71

72

73

74

75

76

77

78

79

80

81

82

83

84

85

86

87

88

89

90

91

92

93

94

95

96

97

98

99

100

101

102

103

104

105

106

107

108

109

110

111

112

113

114

115

116

117

118

119

120

121

122

123

124

125

126

127

128

129

130

131

132

133

134

135

136

137

138

139

140

141

142

143

144

145

146

147

148

149

150

151

152

153

154

155

156

157

158

159

160

161

162

163

164

165

166

167

168

169

170

171

172

173

174

175

176

177

178

179

180

181

182

183

184

185

186

187

188

189

190

191

192

193

194

195

196

197

198

199

200

201

202

203

204

205

206

207

208

209

210

211

212

213

214

215

216

217

218

219

220

221

222

223

224

225

226

227

228

229

230

231

232

233

234

235

236

237

238

239

240

241

242

243

244

245

246

247

248

249

250

251

252

253

254

255

256

257

258

259

260

261

262

263

264

265

266

267

268

269

270

271

272

273

274

275

276

277

278

279

280

281

282

283

284

285

286

287

288

289

290

291

292

293

294

295

296

297

298

299

300

301

302

303

304

305

306

307

308

309

310

311

312

313

314

315

316

317

318

319

320

321

322

323

324

325

326

327

328

329

330

331

332

333

334

335

336

337

338

339

340

341

342

343

344

345

346

347

348

349

350

351

352

353

354

355

356

357

358

359

360

361

362

363

364

365

366

367

368

369

370

371

372

373

374

375

376

377

378

379

380

381

382

383

384

385

386

387

388

389

390

391

392

393

394

395

396

397

398

399

400

401

402

403

404

405

406

407

408

409

410

411

412

413

414

415

416

417

418

419

420

421

422

423

424

425

426

427

428

429

430

431

432

433

434

435

436

437

438

439

440

441

442

443

444

445

446

447

448

449

450

451

452

453

454

455

456

457

458

459

460

461

462

463

464

465

466

467

468

469

470

471

472

473

474

475

476

477

478

479

480

481

482

483

484

485

486

487

488

489

490

491

492

493

494

495

496

497

498

499

500

501

502

503

504

505

506

507

508

509

510

511

512

513

514

515

516

517

518

519

520

521

522

523

524

525

526

527

528

529

530

531

532

533

534

535

536

537

538

539

540

541

542

543

544

545

546

547

548

549

550

551

552

553

554

555

556

557

558

559

560

561

562

563

564

565

566

567

568

569

570

571

572

573

574

575

576

577

578

579

580

581

582

583

584

585

586

587

588

589

590

591

592

593

594

595

596

597

598

599

600

601

602

603

604

605

606

607

608

609

610

611

612

613

614

615

616

617

618

619

620

621

622

623

624

625

626

627

628

629

630

631

632

633

634

635

636

637

638

639

640

641

642

643

644

645

646

647

648

649

650

651

652

653

654

655

656

657

658

659

660

661

662

663

664

665

666

667

668

669

670

671

672

673

674

675

676

677

678

679

680

681

682

683

684

685

686

687

688

689

690

691

692

693

694

695

696

697

698

699

700

701

702

703

704

705

706

707

708

709

710

711

712

713

714

715

716

717

718

719

720

721

722

723

724

725

726

727

728

729

730

731

732

733

734

735

736

737

738

739

740

741

742

743

744

745

746

747

748

749

750

751

752

753

754

755

756

757

758

759

760

761

762

763

764

765

766

767

768

769

770

771

772

773

774

775

776

777

778

779

780

781

782

783

784

785

786

787

788

789

790

791

792

793

794

795

796

797

798

799

800

801

802

803

804

805

806

807

808

809

810

811

812

813

814

815

816

817

818

819

820

821

822

823

824

825

826

827

828

829

830

831

832

833

834

835

836

837

838

839

840

841

842

843

844

845

846

847

848

849

850

851

852

853

854

855

856

857

858

859

860

861

862

863

864

865

866

867

868

869

870

871

872

873

874

875

876

877

878

879

880

881

882

883

884

885

886

887

888

889

890

891

892

893

894

895

896

897

898

899

900

901

902

903

904

905

906

907

908

909

910

911

912

913

914

915

916

917

918

919

920

921

922

923

924

925

926

927

928

929

930

931

932

933

934

935

936

937

938

939

940

941

942

943

944

945

946

947

948

949

950

951

952

953

954

955

956

957

958

959

960

961

962

963

964

965

966

967

968

969

970

971

972

973

974

975

976

977

978

979

980

981

982

983

984

985

986

987

988

989

990

991

992

993

994

995

996

997

998

999

1000

1001

1002

1003

1004

1005

1006

1007

1008

1009

1010

1011

1012

1013

1014

1015

1016

1017

1018

1019

1020

1021

1022

1023

1024

1025

1026

1027

1028

1029

1030

1031

1032

1033

1034

1035

1036

1037

1038

1039

1040

1041

1042

1043

1044

1045

1046

1047

1048

1049

1050

1051

1052

1053

1054

1055

1056

1057

1058

1059

1060

1061

1062

1063

1064

1065

1066

1067

1068

1069

1070

1071

1072

1073

1074

1075

1076

1077

1078

1079

1080

1081

1082

1083

1084

1085

1086

1087

1088

1089

1090

1091

1092

1093

1094

1095

1096

1097

1098

1099

1100

1101

1102

1103

1104

1105

1106

1107

1108

1109

1110

1111

1112

1113

1114

1115

1116

1117

1118

1119

1120

1121

1122

1123

1124

1125

1126

1127

1128

1129

1130

1131

1132

1133

1134

1135

1136

1137

1138

1139

1140

1141

1142

1143

1144

1145

1146

1147

1148

1149

1150

1151

1152

1153

1154

1155

1156

1157

1158

1159

1160

1161

1162

1163

1164

1165

1166

1167

1168

1169

1170

1171

1172

1173

1174

1175

1176

1177

1178

1179

1180

1181

1182

1183

1184

1185

1186

1187

1188

1189

1190

1191

1192

1193

1194

1195

1196

1197

1198

1199

1200

1201

1202

1203

1204

1205

1206

1207

1208

1209

1210

1211

1212

1213

1214

1215

1216

1217

1218

1219

1220

1221

1222

1223

1224

1225

1226

1227

1228

1229

1230

1231

1232

1233

1234

1235

1236

1237

1238

1239

1240

1241

1242

1243

1244

1245

1246

1247

1248

1249

1250

1251

1252

1253

1254

1255

1256

1257

1258

1259

1260

1261

1262

1263

1264

1265

1266

1267

1268

1269

1270

1271

1272

1273

1274

1275

1276

1277

1278

1279

1280

1281

1282

1283

1284

1285

1286

1287

1288

1289

1290

1291

1292

1293

1294

1295

1296

1297

1298

1299

1300

1301

1302

1303

1304

1305

1306

1307

1308

1309

1310

1311

1312

1313

1314

1315

1316

1317

1318

1319

1320

1321

1322

1323

1324

1325

1326

1327

1328

1329

1330

1331

1332

1333

1334

1335

1336

1337

1338

1339

1340

1341

1342

1343

1344

1345

1346

1347

1348

1349

1350

1351

1352

1353

1354

1355

1356

1357

1358

1359

1360

1361

1362

1363

1364

1365

1366

1367

1368

1369

1370

1371

1372

1373

1374

1375

1376

1377

1378

1379

1380

1381

1382

1383

1384

1385

1386

1387

1388

1389

1390

1391

1392

1393

1394

1395

1396

1397

1398

1399

1400

1401

1402

1403

1404

1405

1406

1407

1408

1409

1410

1411

1412

1413

1414

1415

1416

1417

1418

1419

1420

1421

1422

1423

1424

1425

1426

1427

1428

1429

1430

1431

1432

1433

1434

1435

1436

1437

1438

1439

1440

1441

1442

1443

1444

1445

1446

1447

1448

1449

1450

1451

1452

1453

1454

1455

1456

1457

1458

1459

1460

1461

1462

1463

1464

1465

1466

1467

1468

1469

1470

1471

1472

1473

1474

1475

1476

1477

1478

1479

1480

1481

1482

1483

1484

1485

1486

1487

1488

1489

1490

1491

1492

1493

149

0.0% Security Hotspots Reviewed ?

To review

Acknowledged

Fixed

Safe

23 Security Hotspots

Review priority: Medium

Denial of Service (DoS) 16

Make sure the regex used here, which is vulnerable to super-linear runtime due to backtracking, cannot lead to denial of service.

Make sure the regex used here, which is vulnerable to super-linear runtime due to backtracking, cannot lead to denial of service.

Make sure the regex used here, which is vulnerable to super-linear runtime due to backtracking, cannot lead to denial of service.

WhereWhatAssessHowActivity

onetel/client/src/components/Account/Addemp.js

Open in IDE

```

21  const changeOnClick = (e) => {
22    e.preventDefault();
23
24    // Validate fields
25    if (!first_name || !last_name || !email || !Address || !NIC || !Phone || !CusIng) {
26      toast.error('Please fill in all fields');
27      return;
28    }
29    // Validate email format
30    const emailRegex = /^[^@%0-9]{0,50}@[^\s@]{3,100}\.[^\s@]{2,10}$/;
31
32    if (!emailRegex.test(email)) {
33      toast.error('Invalid email address');
34      return;
35    }
36
37    const formData = new FormData();
38    formData.append('first_name', first_name);
39    formData.append('last_name', last_name);

```

Make sure the regex used here, which is vulnerable to super-linear runtime due to backtracking, cannot lead to denial of service.

Denial of Service (DoS) via Uncontrolled Resource Consumption

When an application doesn't limit the size of the requests (e.g., file uploads, JSON bodies), attackers can send excessively large payloads, causing the system to consume all available memory, CPU, or bandwidth.

0.0% Security Hotspots Reviewed ?

To review

Acknowledged

Fixed

Safe

23 Security Hotspots

Review priority: Medium

Denial of Service (DoS) 16

Make sure the regex used here, which is vulnerable to super-linear runtime due to backtracking, cannot lead to denial of service.

Make sure the content length limit is safe here.

Make sure the content length limit is safe here.

Make sure the content length limit is safe here.

Make sure the content length limit is safe here.

Make sure the content length limit is safe here.

Make sure the content length limit is safe here.

WhereWhatAssessHowActivity

onetel/routes/emp.js

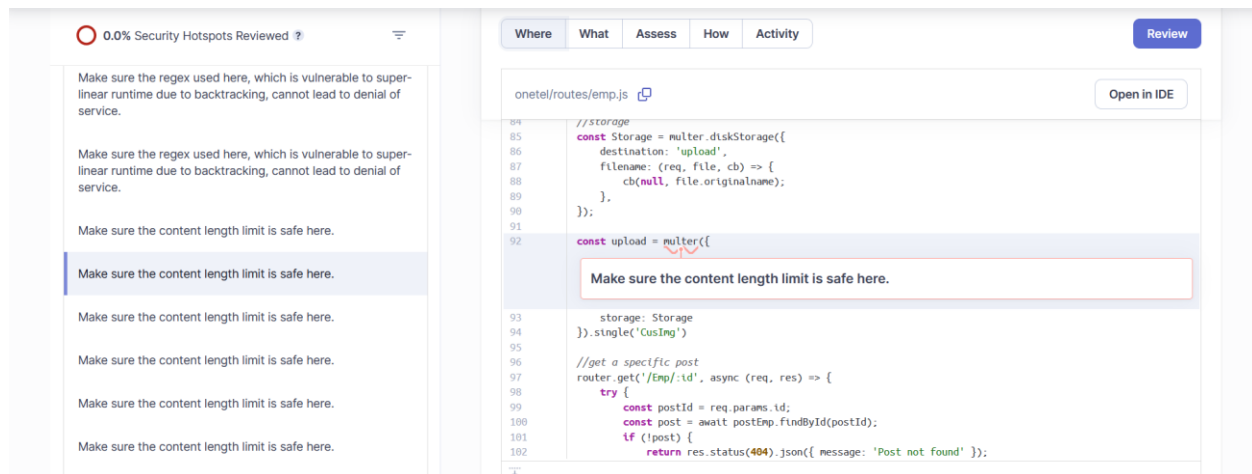
Open in IDE

```

80  });
81  });
82  });
83
84  //storage
85  const Storage = multer.diskStorage({
86    destination: 'upload',
87    filename: (req, file, cb) => {
88      cb(null, file.originalname);
89    },
90  });
91
92  const upload = multer({
93    storage: Storage
94  }).single('CusIng')

```

Make sure the content length limit is safe here.



```

92 //this fixed accordingly sonar qube
93 // const upload = multer({
94 //   storage: Storage
95 // }).single('CusImg')
96
97 //fixed
98 const upload = multer({
99   storage: Storage,
100   limits: {
101     fileSize: 8000000 // 8MB limit
102   }
103 }).single('file');
104

```

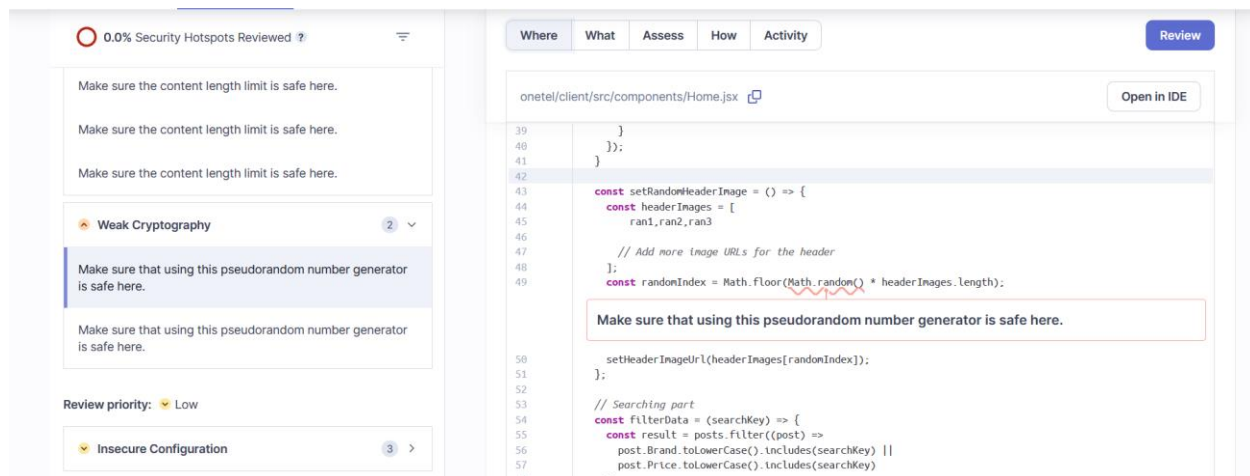
Using Weak Pseudorandom Number Generators (PRNGs)

Using pseudorandom number generators (PRNGs) is security-sensitive. For example, it has led in the past to the following vulnerabilities:

- [CVE-2013-6386](#)
- [CVE-2006-3419](#)
- [CVE-2008-4102](#)

When software generates predictable values in a context requiring unpredictability, it may be possible for an attacker to guess the next value that will be generated, and use this guess to impersonate another user or access sensitive information.

As the `Math.random()` function relies on a weak pseudorandom number generator, this function should not be used for security-critical applications or for protecting sensitive data. In such context, a cryptographically strong pseudorandom number generator (CSPRNG) should be used instead.



Preventing Using Weak Pseudorandom Number Generators.

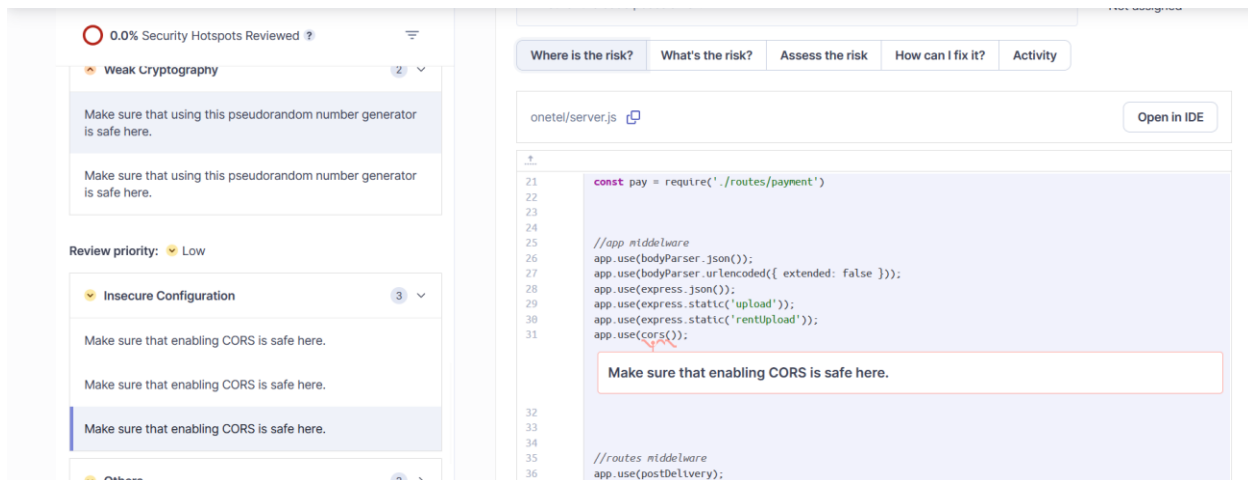
```

42 //issue accordingly sonar qube
43 // const setRandomHeaderImage = () => {
44 //   const headerImages = [
45 //     ran1,ran2,ran3
46 //   ]
47 //   // Add more image URLs for the header
48 //   };
49 // const randomIndex = Math.floor(Math.random() * headerImages.length);
50 // setHeaderImageUrl(headerImages[randomIndex]);
51 // };
52
53 //this is fixed one
54 const setRandomHeaderImage = () => {
55   const headerImages = [ran1, ran2, ran3]; // Your image URLs
56
57   const crypto = window.crypto || window.msCrypto; // Use browser CSPRNG
58   const array = new Uint32Array(1); // Generate a secure random number
59   crypto.getRandomValues(array); // Get a cryptographically secure random number
60
61   const randomIndex = array[0] % headerImages.length; // Ensure index is within array bounds
62   setHeaderImageUrl(headerImages[randomIndex]); // Set random image
63 };
64

```

Permissive Cross-Origin Resource Sharing (CORS) Policy.

CORS is a browser security feature that restricts web applications from making requests to a different domain than the one that served the web page. By default, the **Same-Origin Policy** blocks cross-origin requests to protect against attacks such as **Cross-Site Request Forgery (CSRF)** or **Cross-Site Scripting (XSS)**. However, having a permissive CORS policy (e.g., using a wildcard * in Access-Control-Allow-Origin) can expose your web application to security risks. Attackers could exploit this vulnerability to make unauthorized cross-origin requests, potentially compromising user data or application functionality.



Preventing Permissive Cross-Origin Resource Sharing (CORS) Policy vulnerabilities.

```

10
11 // fix the issue
12 // Define trusted origins for CORS policy
13 const corsOptions = {
14   origin: ['http://localhost:3000'], // Replace this with the trusted domains
15   optionsSuccessStatus: 200 // For legacy browser support
16 };
17
18 // Apply CORS middleware with restricted origins
19 router.use(cors(corsOptions)); // UPDATED: Restrict CORS to trusted origins only
20
21

```

Unverified Remote Resource Inclusion

Using remote artifacts without integrity checks can lead to the unexpected execution of malicious code in the application.

On the client side, where front-end code is executed, malicious code could:

- impersonate users' identities and take advantage of their privileges on the application.
- add quiet malware that monitors users' session and capture sensitive secrets.
- gain access to sensitive clients' personal data.
- deface, or otherwise affect the general availability of the application.
- mine cryptocurrencies in the background.

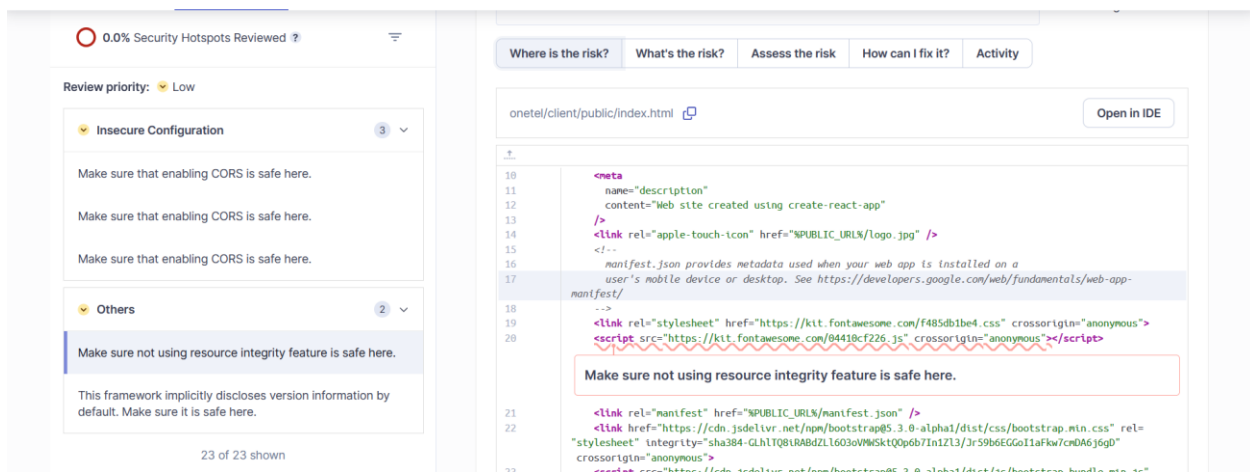
Likewise, a compromised software piece that would be deployed on a server-side application could badly affect the application's security. For example, server-side malware could:

- access and modify sensitive technical and business data.
- elevate its privileges on the underlying operating system.
- Use the compromised application as a pivot to attack the local network.

By ensuring that a remote artifact is exactly what it is supposed to be before using it, the application is protected from unexpected changes applied to it before it is downloaded.

Especially, integrity checks will allow for identifying an artifact replaced by malware on the publication website or that was legitimately changed by its author, in a more benign scenario.

Important note: downloading an artifact over HTTPS only protects it while in transit from one host to another. It provides authenticity and integrity checks **for the network stream** only. It does not ensure the authenticity or security of the artifact itself.



Preventing Unverified Remote Resource Inclusion



Version Disclosure

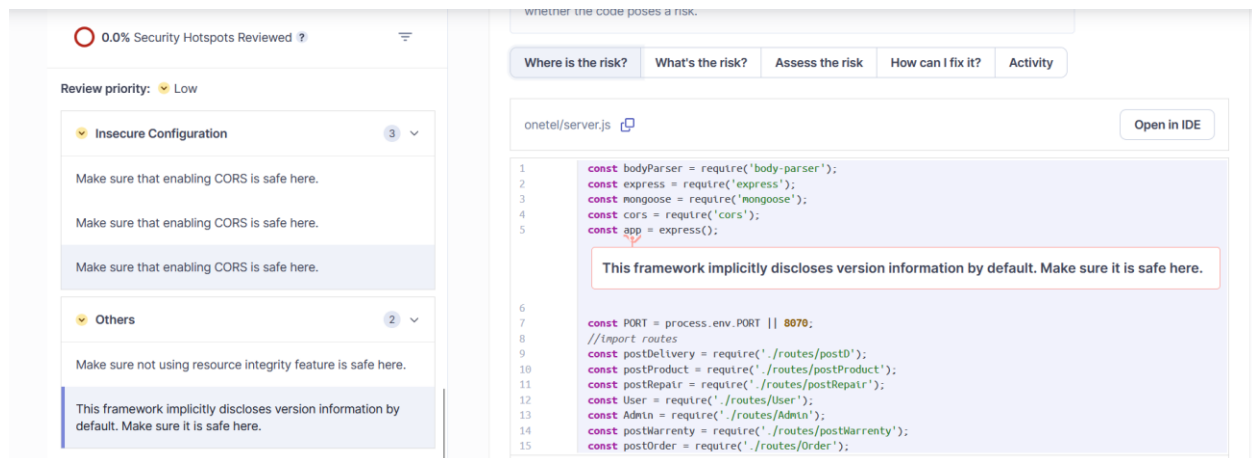
In general, it is recommended to keep internal technical information within internal systems to control what attackers know about the underlying architectures. This is known as the "need to know" principle.

The most effective solution is to remove version information disclosure from what end users can see, such as the "x-powered-by" header.

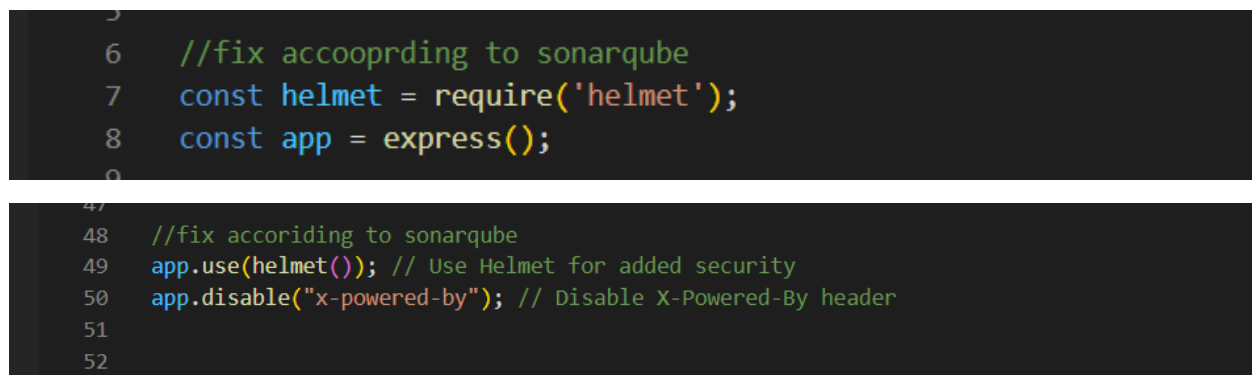
This can be achieved directly through the web application code, server (nginx, apache) or firewalls.

Disabling the server signature provides additional protection by reducing the amount of information available to attackers. Note, however, that this does not provide as much protection as regular updates and patches.

Security by obscurity is the least foolproof solution of all. It should never be the only defense mechanism and should always be combined with other security measures.



Preventing Version Disclosure :



Cross-site Scripting (XSS)

The application is vulnerable to Cross-Site Scripting (XSS) attacks due to the unsanitized input being received from the HTTP request body. Specifically, user inputs, such as email addresses in the login functionality, are processed without adequate validation or sanitization before being reflected back in the application's responses. This oversight can allow an attacker to inject malicious scripts that are executed in the context of the user's browser. If exploited, such an attack could lead to unauthorized actions on behalf of the user, session hijacking, and exposure of sensitive data.

Cross-site Scripting (XSS)

Data flow
Fix analysis
X

SNYK CODE

CWE-79

Unsanitized input from *the HTTP request body flows* into *send*, where it is used to render an HTML page returned to the user. This may result in a Cross-Site Scripting attack (XSS).

Data flow - 16 steps in 1 file

onetel/routes/User.js

16 steps

65:31

body, body, req.bo...

SOURCE

1-5

66:13

user

6

67:13

user

7

Find out how to remediate this issue through our [Fix analysis](#)

onetel/routes/User.js

```

58 // Login route
59 router.post('/login', (req, res) => {
60   // Input validation
61   if (typeof req.body.email !== 'string'){
62     return res.status(400).json({ error: 'Invalid input type'
63   }
64
65   User.findOne({ email: req.body.email })
66   .then(user => {
67     if (user) {
68       if (bcrypt.compareSync(req.body.password, user.password)
69       const payload = {
70         _id: user._id,
71         first_name: user.first_name,
72         last_name: user.last_name,
73         email: user.email
74       };
75       let token = jwt.sign(payload, process.env.SECRET_KEY);

```

Ignore

Preventing Cross-site Scripting (XSS)

```

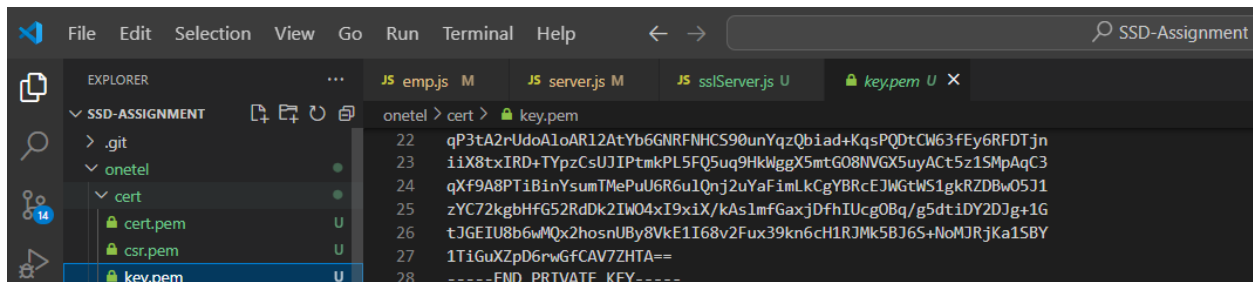
91 //Fix XSS
92 // Login route
93 router.post('/login', (req, res) => {
94   // Input validation
95   if (typeof req.body.email !== 'string') {
96     return res.status(400).json({ error: 'Invalid input type' });
97   }
98
99   // Sanitize email input to prevent XSS
100   const sanitizedEmail = sanitizeHtml(req.body.email);
101
102   User.findOne({ email: sanitizedEmail })
103   .then(user => {
104     if (user) {
105       if (bcrypt.compareSync(req.body.password, user.password)) {
106         const payload = {
107           _id: user._id,
108           first_name: user.first_name,
109           last_name: user.last_name,
110           email: user.email
111         };
112         let token = jwt.sign(payload, process.env.SECRET_KEY, { expiresIn: 1440 });
113         res.send(token);
114       } else {
115         res.status(401).json({ error: 'Invalid password' });
116       }
117     } else {
118       res.status(404).json({ error: 'User not found' });
119     }
120   })
121   .catch(err => {
122     res.status(500).json({ error: 'Server error' });
123   });
124 });
125

```

17 | Page

Add SSL Certificate

```
JS emp.js M JS server.js M JS sslServer.js U X
oneteel > client > JS sslServer.js > app.get("/") callback
1  const https = require('https');
2  const fs = require('fs');
3  const path = require('path');
4  const express = require('express');
5
6  const app = express();
7  const PORT = 3000;
8
9  // Serve static files (for React or any frontend framework)
10 app.use(express.static(path.join(__dirname, 'build')));
11
12 // Handle all routes (for React Router or any SPA routing)
13 app.get('*', (req, res) => {
14   res.sendFile(path.join(__dirname, 'build', 'index.html'));
15 });
16
17 // Set up SSL
18 const sslServer = https.createServer({
19   key: fs.readFileSync(path.join(__dirname, 'cert', 'key.pem')),
20   cert: fs.readFileSync(path.join(__dirname, 'cert', 'cert.pem')),
21 }, app);
22
23 sslServer.listen(PORT, () => {
24   console.log(`Frontend running securely on https://localhost:${PORT}`);
25 });
26
```

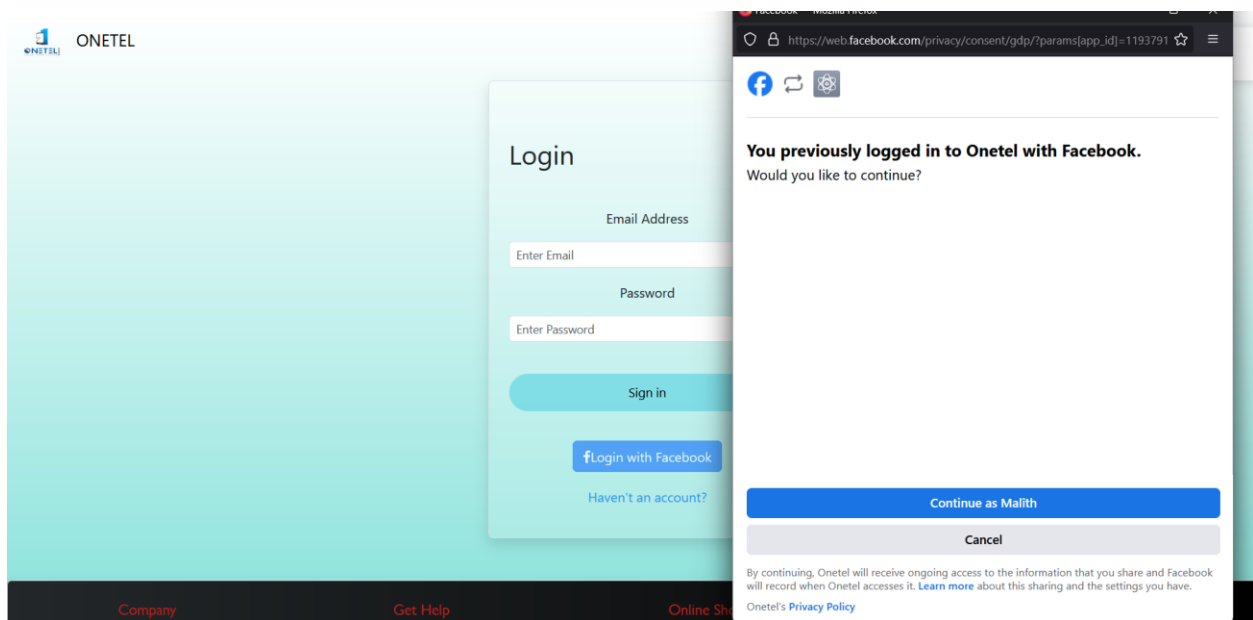


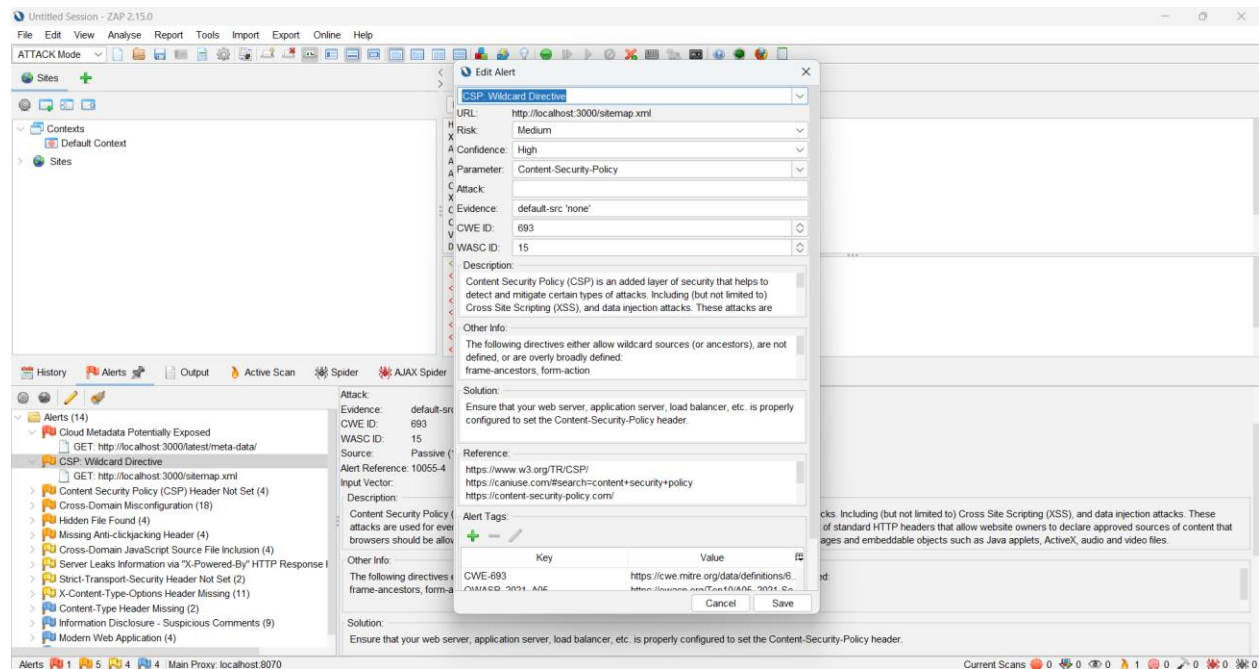
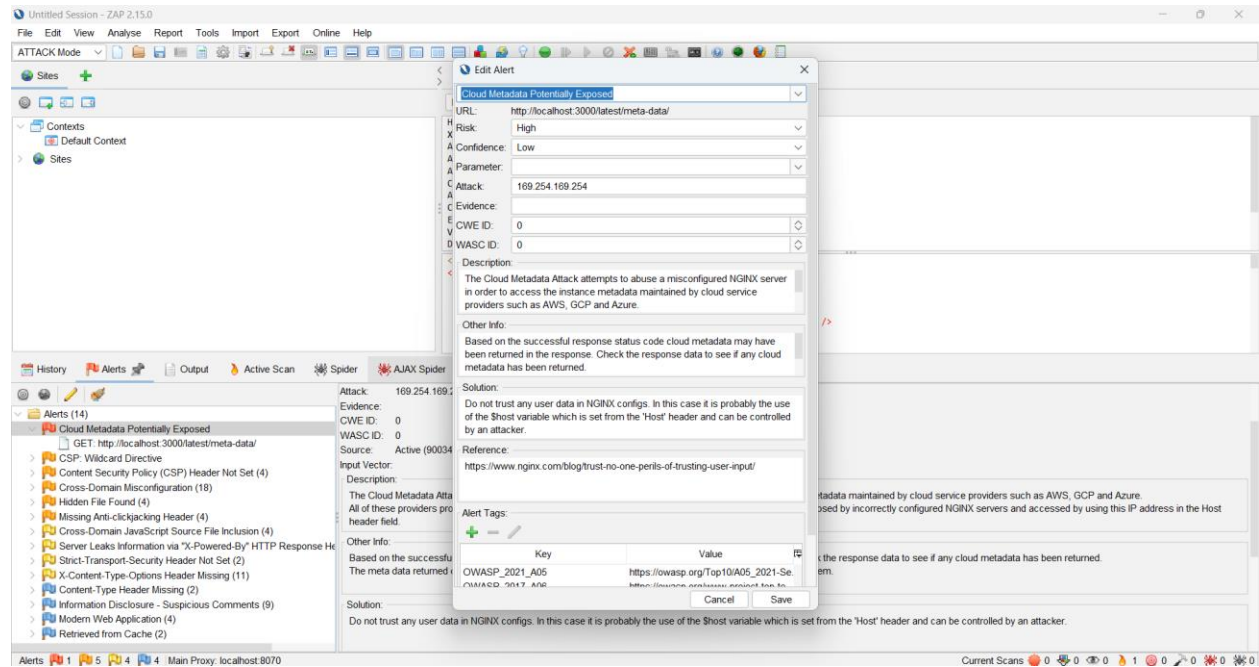
OAuth connect-based grant

```
<div className='form-group'>
  <label htmlFor='password'>Password</label><br /><br />
  <input
    type='password'
    className='form-control form-control-sm'
    name='password'
    placeholder='Enter Password'
    value={this.state.password}
    onChange={this.onChange}
  />
</div><br /><br />
<button className='btn logbtn'>Sign in</button>
<br /><br />

{/* Facebook Login Button */}
<FacebookLogin
  appId="1193791145246680"
  autoLoad={false}
  fields="name,email,picture"
  callback={this.responseFacebook}
  icon="fa-facebook"
  textButton="Login with Facebook"
  cssClass="btn btn-primary btn-block mt-3"
/>

<br /><br />
<Link className="link" to="/register">Haven't an account?</Link>
</h6>
</form>
</div>
```





Untitled Session - ZAP 2.15.0

File Edit View Analyse Report Tools Import Export Online Help

ATTACK Mode

Sites

Contexts

Default Context

Sites

History

Alerts

Output

Active Scan

Spider

AJAX Spider

Alerts (14)

- Cloud Metadata Potentially Exposed
 - GET: http://localhost:3000/latest/meta-data/
- CSP: Wildcard Directive
 - GET: http://localhost:3000/robots.txt
 - GET: http://localhost:3000/register
- Content Security Policy (CSP) Header Not Set (4)
 - GET: http://localhost:3000/
 - GET: http://localhost:3000/chat
 - GET: http://localhost:3000/login
 - GET: http://localhost:3000/register
- Cross-Domain Misconfiguration (18)
 - GET: http://localhost:3000/
 - GET: http://localhost:3000/favicon.ico
 - GET: http://localhost:3000/logo.jpg
 - GET: http://localhost:3000/manifest.json
 - GET: http://localhost:3000/robots.txt
 - GET: http://localhost:3000/sitemap.xml
 - GET: http://localhost:3000/static/js/bundle.js
 - GET: http://localhost:3000/static/media/0.8fe90accf5103cd
 - GET: http://localhost:3000/static/media/0.37b9a5e23fa911
 - GET: http://localhost:3000/static/media/0.4793e85ac1346
 - GET: https://cdn.jsdelivr.net/npm/bootstrap@5.3.0-alpha.1/dist
 - GET: https://cdn.jsdelivr.net/npm/bootstrap@5.3.0-alpha.1/dist
 - GET: https://kit.fontawesome.com/04410c226.js
 - GET: https://kit.fontawesome.com/1485db1be4.css
- Hidden File Found (4)
 - GET: http://localhost:3000/
 - GET: http://localhost:3000/
 - GET: http://localhost:3000/
 - GET: http://localhost:3000/
- Missing Anti-clickjacking Header (4)
 - GET: http://localhost:3000/
 - GET: http://localhost:3000/
 - GET: http://localhost:3000/
 - GET: http://localhost:3000/
- Cross-Domain JavaScript Source File Inclusion (4)
 - GET: http://localhost:3000/
 - GET: http://localhost:3000/
 - GET: http://localhost:3000/
 - GET: http://localhost:3000/
- Server Leaks Information via "X-Powered-By" HTTP Response (1)
 - GET: http://localhost:3000/
- Strict-Transport-Security Header Not Set (2)
 - GET: http://localhost:3000/
 - GET: http://localhost:3000/

Alerts

1 5 4 4 Main Proxy: localhost:8070

Current Scans 0 0 0 0 1 0 0 0 0 0

Content Security Policy (CSP) Header Not Set

URL: http://localhost:3000/

Risk: Medium

Confidence: High

Parameter:

Attack:

Evidence:

CWE ID: 693

WASC ID: 15

Description:

Content Security Policy (CSP) is an added layer of security that helps to detect and mitigate certain types of attacks, including Cross Site Scripting (XSS) and data injection attacks. These attacks are used for everything from

Other Info:

Solution:

Ensure that your web server, application server, load balancer, etc. is configured to set the Content-Security-Policy header.

Reference:

https://developer.mozilla.org/en-US/docs/Web/Security/CSP/Introducing_Content_Security_Policy

https://cheatsheetseries.owasp.org/cheatsheets/Content_Security_Policy_C

Alert Tags:

Key	Value
CWE-693	https://cwe.mitre.org/data/definitions/693
OWASP_2017_A05	https://owasp.org/www-project-top-ten/2017/A5_2017-Broken_Access_Control.html

Cancel Save

Untitled Session - ZAP 2.15.0

File Edit View Analyse Report Tools Import Export Online Help

ATTACK Mode

Sites

Contexts

Default Context

Sites

History

Alerts

Output

Active Scan

Spider

AJAX Spider

Alerts (14)

- Cloud Metadata Potentially Exposed
 - GET: http://localhost:3000/latest/meta-data/
- CSP: Wildcard Directive
 - GET: http://localhost:3000/robots.txt
 - GET: http://localhost:3000/register
- Content Security Policy (CSP) Header Not Set (4)
 - GET: http://localhost:3000/
 - GET: http://localhost:3000/chat
 - GET: http://localhost:3000/login
 - GET: http://localhost:3000/register
- Cross-Domain Misconfiguration (18)
 - GET: http://localhost:3000/
 - GET: http://localhost:3000/favicon.ico
 - GET: http://localhost:3000/logo.jpg
 - GET: http://localhost:3000/manifest.json
 - GET: http://localhost:3000/robots.txt
 - GET: http://localhost:3000/sitemap.xml
 - GET: http://localhost:3000/static/js/bundle.js
 - GET: http://localhost:3000/static/media/0.8fe90accf5103cd
 - GET: http://localhost:3000/static/media/0.37b9a5e23fa911
 - GET: http://localhost:3000/static/media/0.4793e85ac1346
 - GET: https://cdn.jsdelivr.net/npm/bootstrap@5.3.0-alpha.1/dist
 - GET: https://cdn.jsdelivr.net/npm/bootstrap@5.3.0-alpha.1/dist
 - GET: https://kit.fontawesome.com/04410c226.js
 - GET: https://kit.fontawesome.com/1485db1be4.css
- Hidden File Found (4)
 - GET: http://localhost:3000/
 - GET: http://localhost:3000/
 - GET: http://localhost:3000/
 - GET: http://localhost:3000/
- Missing Anti-clickjacking Header (4)
 - GET: http://localhost:3000/
 - GET: http://localhost:3000/
 - GET: http://localhost:3000/
 - GET: http://localhost:3000/
- Cross-Domain JavaScript Source File Inclusion (4)
 - GET: http://localhost:3000/
 - GET: http://localhost:3000/
 - GET: http://localhost:3000/
 - GET: http://localhost:3000/
- Server Leaks Information via "X-Powered-By" HTTP Response (1)
 - GET: http://localhost:3000/
- Strict-Transport-Security Header Not Set (2)
 - GET: http://localhost:3000/
 - GET: http://localhost:3000/

Alerts

1 5 4 4 Main Proxy: localhost:8070

Current Scans 0 0 0 0 1 0 0 0 0 0

Cross-Domain Misconfiguration

URL: http://localhost:3000/logo.jpg

Risk: Medium

Confidence: Medium

Parameter:

Attack:

Evidence:

CWE ID: 264

WASC ID: 14

Description:

Web browser data loading may be possible, due to a Cross Origin Resource Sharing (CORS) misconfiguration on the web server.

Other Info:

The CORS misconfiguration on the web server permits cross-domain read requests from arbitrary third party domains, using unauthenticated APIs on this domain. Web browser implementations do not permit arbitrary third

Solution:

Ensure that sensitive data is not available in an unauthenticated manner (using IP address white-listing, for instance).

Configure the "Access-Control-Allow-Origin" HTTP header to a more restrictive manner.

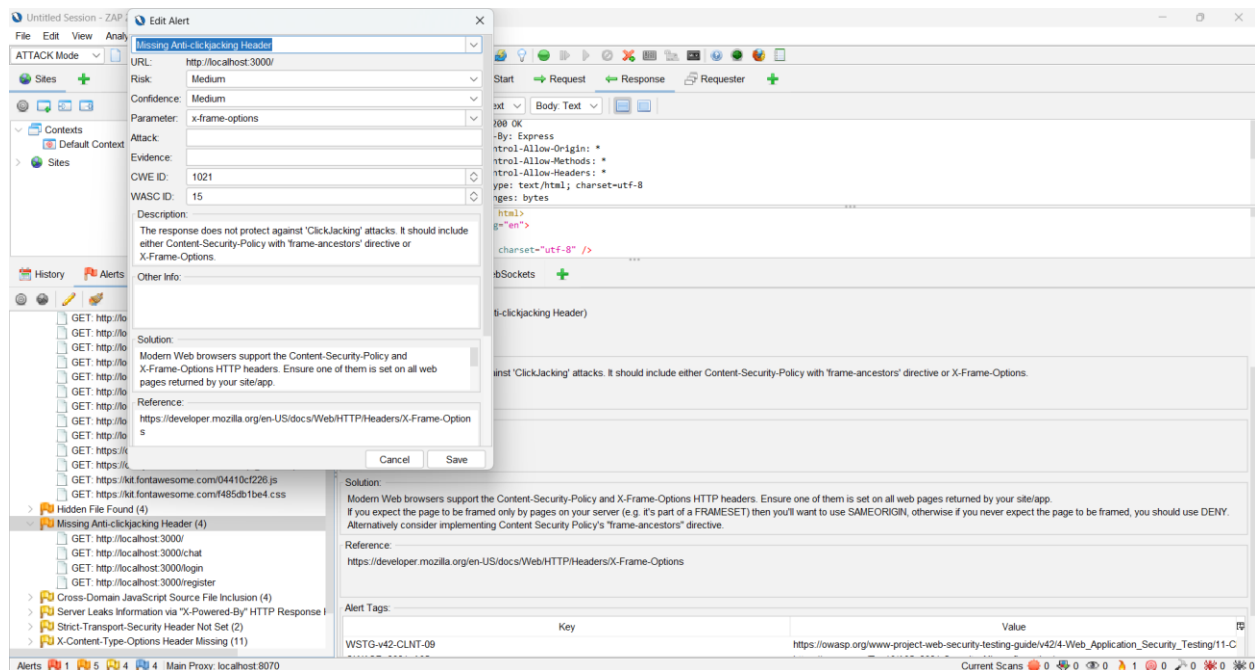
Reference:

https://vuln.catfortify.com/en/detail?id=desc.config.dotnet.html5_overly_permiss

Alert Tags:

Key	Value
OWASP_2017_A05	https://owasp.org/www-project-top-ten/2017/A5_2017-Broken_Access_Control.html
CWE-264	https://cwe.mitre.org/data/definitions/264.html

Cancel Save



Best Practices

Security-first Design & Development (Security by Design)

Description: Security considerations should be implemented right from the beginning of the SDLC. A security-first mindset supports the goal of making an application resilient against most common vulnerabilities and attacks.

Prevention

- Clearly define security requirements within the design phase.
- Plan on possible attack vectors, such as injection attacks or data breaches, while designing a feature.
- Follow secure coding guidelines and practices to avoid introducing vulnerabilities.

Use of Environment Variables for Sensitive Information

Description: Never hard-code sensitive credentials like API keys, database passwords, and secret keys directly into the source code. They always must be kept in environment variables.

Prevention:

- Use .env files during development and use environment variables during production.
- Ensure that the .env file was added to the .gitignore to avoid committing into version control.
- AWS Secrets Manager, HashiCorp Vault - Secret Management Services: Implement secret management services that securely store, manage, and rotate sensitive information in production.

Code Reviews and Pair Programming

Description: This is reviewing each other's code before merging to the main branch-code reviews. It consists of having two developers collaborating on one task-pair programming. Such activities would help identify practices such as hardcoded secrets, bad error handling, or other security flaws.

How to Avoid It:

Regular code reviews will find hardcoded secrets, poor validation, and other common security mistakes. Pair programming introduces immediate feedback that reduces the likelihood of security issues being missed during the coding phase.

DevSecOps Approach

Description: DevSecOps integrates security practices into DevOps. In this respect, it ensures that security considerations are taken along the whole development and deployment cycle.

How to Avoid:

- Implement code security checks like hardcoded credential scanning in your CI/CD pipeline.
- Automate security testing and vulnerability scanning throughout development, from code commit to deployment.

CI/CD with Security Gates

Description: CI/CD pipeline automates the testing of code and its deployment. Moreover, embedding security checks within the pipeline ensures that most vulnerabilities are detected before the code reaches production.

Prevention:

- Use the security gates part of the CI/CD process that will scan for vulnerabilities like Hard Coded Credentials and Insecure Dependencies.
- Tools like Jenkins, GitLab CI, and CircleCI can integrate with security scanners and SAST tools to catch vulnerabilities well in advance.