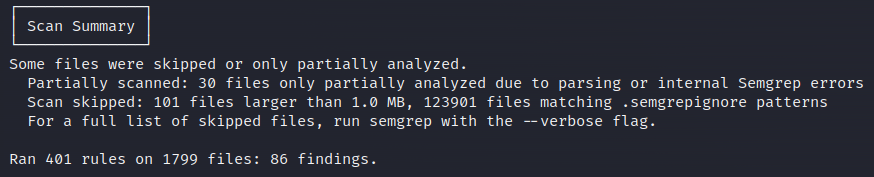
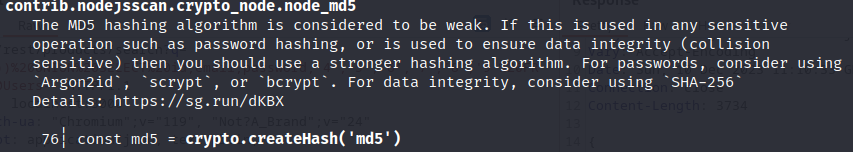
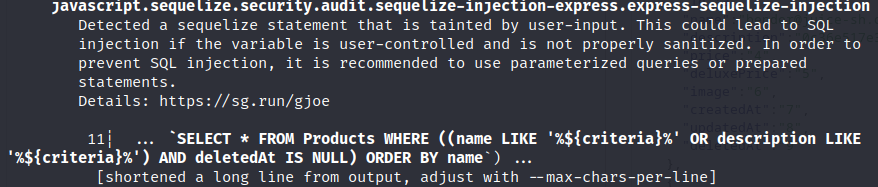
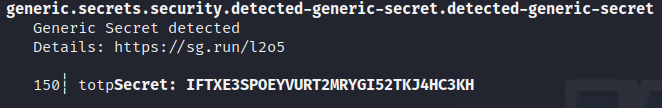
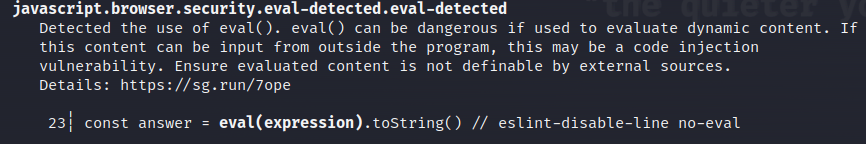
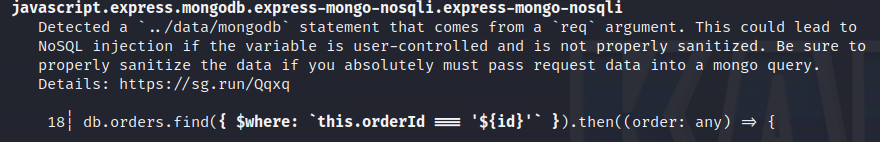
**Практическое задание. Анализ защищённости веб-приложений**

1. **Введение**  
   Juice Shop – это проект, созданный некоммерческой организацией Open Web Application Security Project). Вся деятельность организации посвящена вопросам обеспечения безопасности веб-приложений и данный проект призван продемонстрировать самые часто встречаемые уязвимости в приложениях и худшие практики веб-разработки.   
   Мы будем использовать Juice Shop в качестве полигона для практики при поиске уязвимостей.
2. **Результаты статического анализа**Статический анализ выявил 86 уязвимостей.   
   

Полный текст вывода при приведен в приложении 1.

1. **Уязвимости из OWASP Top-10, обнаруженные в результате статического анализа (минимум 5шт).**В 2023 году рейтинг OWASP Top-10 выглядит так:
   1. Broken Object Level Authorization
   2. Broken Authentication
   3. Broken Object Property Level Authorization
   4. Unrestricted Resource Consumption
   5. Broken Function Level Authorization
   6. Unrestricted Access to Sensitive Business Flows
   7. Server-Side Request Forgery
   8. Security Misconfiguration
   9. Improper Inventory Management
   10. Unsafe Consumption of APIs

В результате статического анализа были выявлены:

1. Использования алгоритма MD-5 для хэширования паролей (Security Misconfiguration).  
   
2. Отсутствие контроля над SQL-запросами, риск SQL-инъекций (Broken Object Level Authorization).   
   
3. Хранение ключа в открытом виде (Security Misconfiguration).   
   
4. Уязвимый алгоритм проверки капчи (Security Misconfiguration).   
   
5. Уязвимый алгоритм отслеживания заказа (Unrestricted Access to Sensitive Business Flows).   
   
6. **Демонстрация эксплуатации трех уязвимостей из OWASP-10.**Для демонстрации работы уязвимостей буду использовать результаты предыдущей работы, точнее текстовые описания действий (к сожалению, лицензия на BurpSuite истекла, а новую не присылают какую бы почту я не указывал).

**Уязвимость капчи.**   
Включаем перехват.

Оставляем отзыв, проходим капчу. Несколько раз.

Через Proxy->HTTP history изучаем структуру получения и отправки запросов.

Отправляем Request с успешными captchaID и captcha в Repeater.

Через Repeater отправляем нужное количество раз.

Также подобный сформированный запрос можно использовать в любых программных средствах, скриптах и т.д.

**Риск SQL-инъекций**Включаем перехват.

Через строку поиска делаем легитимный запрос.

Через историю находим запрос поиска, отправляем в Repeater.

Пробуем менять параметры поиска напрямую в запросе:   
/rest/products/search?q=

Добавляем к запросу одинарную кавычку.  
Пример: ‘apple или apple’

Получаем ошибку и узнаем какая база данных используется.  
message": "SQLITE\_ERROR: near \"apple\": syntax error"

В процессе экспериментов с запросом сначала получаем ошибку незавершённого запроса и добавляем две закрывающие скобки.   
Зная продукт (БД) и синтаксис пробуем к запросу поиска добавить свой:  
UNION%20SELECT%201,2,3,4,5,6,7,8,9%20FROM%20sqlite\_master—

Успех. Развиваем:  
UNION%20SELECT%20sql,2,3,4,5,6,7,8,9%20FROM%20sqlite\_master

Успех. В ответе видим всю схему.

Далее получаем данные из таблицы Users:  
RAZ'))%20UNION%20SELECT%20id,email,password,'4','5','6','7','8','9'%20FROM%20Users--

**Использования алгоритма MD-5 для хэширования паролей**

Из таблицы Users мы получаем информацию: в поле Description написаны символы похожие на хэш.

Используем сервис crackstation.net, получаем пароль (для шифрования использовался алгоритм MD5).

1. **Рекомендации по устранению трех выявленных уязвимостей.**

**1. Уязвимость капчи**Провести рефакторинг кода и, по возможности, избавиться от использования фунции eval().   
<https://cwe.mitre.org/data/definitions/95.html>

**2. Риск SQL-инъекций**Внедрение различных способ для митигации данного риска. Варианты различные, от изменений архитектурного подхода, создания учетных записейс максимально ограниченными правами до параметризации запросов и использования профильных библиотек и фреймворков.   
<https://cwe.mitre.org/data/definitions/89.html>

**3. Использование алгоритма MD-5**Использование «слабого» алгоритма шифрования решается заменой на более «стойкий» к дешифровке алгоритм.   
<https://cwe.mitre.org/data/definitions/328.html>

Приложение 1.

┌──(slavautkin㉿SF-Kali-01)-[~/Downloads/juice-shop]

└─$ semgrep scan --config auto --no-git-ignore

┌──── ○○○ ────┐

│ Semgrep CLI │

└─────────────┘

Scanning 125828 files with:

✔ Semgrep OSS

✔ Basic security coverage for first-party code vulnerabilities.

✔ Semgrep Code (SAST)

✔ Find and fix vulnerabilities in the code you write with advanced scanning and expert security rules.

✘ Semgrep Supply Chain (SCA)

✘ Find and fix the reachable vulnerabilities in your OSS dependencies.

━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━╸ 100% 0:00:44

Warning: 1 timeout error(s) in frontend/src/assets/private/three.js when running the following rules:

[typescript.react.security.audit.react-unsanitized-method.react-unsanitized-method]

┌──────────────────┐

│ 86 Code Findings │

└──────────────────┘

Gruntfile.js

contrib.nodejsscan.crypto\_node.node\_md5

The MD5 hashing algorithm is considered to be weak. If this is used in any sensitive

operation such as password hashing, or is used to ensure data integrity (collision

sensitive) then you should use a stronger hashing algorithm. For passwords, consider using

`Argon2id`, `scrypt`, or `bcrypt`. For data integrity, consider using `SHA-256`

Details: https://sg.run/dKBX

76┆ const md5 = crypto.createHash('md5')

data/datacreator.ts

javascript.lang.security.audit.path-traversal.path-join-resolve-traversal.path-join-resolve-

traversal

Detected possible user input going into a `path.join` or `path.resolve` function. This could

possibly lead to a path traversal vulnerability, where the attacker can access arbitrary

files stored in the file system. Instead, be sure to sanitize or validate user input first.

Details: https://sg.run/OPqk

41┆ const filePath = path.resolve('./data/static/' + file + '.yml')

data/static/codefixes/dbSchemaChallenge\_1.ts

javascript.sequelize.security.audit.sequelize-injection-express.express-sequelize-injection

Detected a sequelize statement that is tainted by user-input. This could lead to SQL

injection if the variable is user-controlled and is not properly sanitized. In order to

prevent SQL injection, it is recommended to use parameterized queries or prepared

statements.

Details: https://sg.run/gjoe

5┆ ... "SELECT \* FROM Products WHERE ((name LIKE '%"+criteria+"%' OR description LIKE

'%"+criteria+"%') AND deletedAt IS NULL) ORDER BY name") ...

[shortened a long line from output, adjust with --max-chars-per-line]

data/static/codefixes/dbSchemaChallenge\_3.ts

javascript.sequelize.security.audit.sequelize-injection-express.express-sequelize-injection

Detected a sequelize statement that is tainted by user-input. This could lead to SQL

injection if the variable is user-controlled and is not properly sanitized. In order to

prevent SQL injection, it is recommended to use parameterized queries or prepared

statements.

Details: https://sg.run/gjoe

11┆ ... `SELECT \* FROM Products WHERE ((name LIKE '%${criteria}%' OR description LIKE

'%${criteria}%') AND deletedAt IS NULL) ORDER BY name`) ...

[shortened a long line from output, adjust with --max-chars-per-line]

data/static/codefixes/restfulXssChallenge\_2.ts

javascript.audit.detect-replaceall-sanitization.detect-replaceall-sanitization

Detected a call to `replaceAll()` in an attempt to HTML escape the string

`tableData[i].description`. Manually sanitizing input through a manually built list can be

circumvented in many situations, and it's better to use a well known sanitization library

such as `sanitize-html` or `DOMPurify`.

Details: https://sg.run/AzoB

59┆ tableData[i].description = tableData[i].description.replaceAll('<', '&lt;').replaceAll('>',

'&gt;')

⋮┆----------------------------------------

javascript.audit.detect-replaceall-sanitization.detect-replaceall-sanitization

Detected a call to `replaceAll()` in an attempt to HTML escape the string

`tableData[i].description.replaceAll('<', '&lt;')`. Manually sanitizing input through a

manually built list can be circumvented in many situations, and it's better to use a well

known sanitization library such as `sanitize-html` or `DOMPurify`.

Details: https://sg.run/AzoB

59┆ tableData[i].description = tableData[i].description.replaceAll('<', '&lt;').replaceAll('>',

'&gt;')

data/static/codefixes/unionSqlInjectionChallenge\_1.ts

javascript.sequelize.security.audit.sequelize-injection-express.express-sequelize-injection

Detected a sequelize statement that is tainted by user-input. This could lead to SQL

injection if the variable is user-controlled and is not properly sanitized. In order to

prevent SQL injection, it is recommended to use parameterized queries or prepared

statements.

Details: https://sg.run/gjoe

6┆ ... `SELECT \* FROM Products WHERE ((name LIKE '%${criteria}%' OR description LIKE

'%${criteria}%') AND deletedAt IS NULL) ORDER BY name`) ...

[shortened a long line from output, adjust with --max-chars-per-line]

data/static/codefixes/unionSqlInjectionChallenge\_3.ts

javascript.sequelize.security.audit.sequelize-injection-express.express-sequelize-injection

Detected a sequelize statement that is tainted by user-input. This could lead to SQL

injection if the variable is user-controlled and is not properly sanitized. In order to

prevent SQL injection, it is recommended to use parameterized queries or prepared

statements.

Details: https://sg.run/gjoe

10┆ ... `SELECT \* FROM Products WHERE ((name LIKE '%${criteria}%' OR description LIKE

'%${criteria}%') AND deletedAt IS NULL) ORDER BY name`) ...

[shortened a long line from output, adjust with --max-chars-per-line]

data/static/users.yml

generic.secrets.security.detected-generic-secret.detected-generic-secret

Generic Secret detected

Details: https://sg.run/l2o5

150┆ totpSecret: IFTXE3SPOEYVURT2MRYGI52TKJ4HC3KH

docker-compose.test.yml

yaml.docker-compose.security.no-new-privileges.no-new-privileges

Service 'app' allows for privilege escalation via setuid or setgid binaries. Add 'no-new-

privileges:true' in 'security\_opt' to prevent this.

Details: https://sg.run/0n8q

7┆ app:

⋮┆----------------------------------------

yaml.docker-compose.security.writable-filesystem-service.writable-filesystem-service

Service 'app' is running with a writable root filesystem. This may allow malicious

applications to download and run additional payloads, or modify container files. If an

application inside a container has to save something temporarily consider using a tmpfs. Add

'read\_only: true' to this service to prevent this.

Details: https://sg.run/e4JE

7┆ app:

frontend/.angular/cache/15.2.10/babel-webpack/11d729f670bcbcec51618ac7dedf81b969cb121c00db83a6c70ca7d67a840be3.json

generic.secrets.security.detected-facebook-oauth.detected-facebook-oauth

Facebook OAuth detected

Details: https://sg.run/Klq6

1┆ ... n if (rf & 1) {\n const \_r12 = i0.ɵɵgetCurrentView();\n i0.ɵɵelementStart(0,

\"mat-expansion-panel\", 20)(1, \"mat-expansion-panel-header\", 21) ...

[shortened a long line from output, adjust with --max-chars-per-line]

frontend/.angular/cache/15.2.10/babel-webpack/87f8b5c1c586c8a428a52a8e7eadca5cb2cc46b37ec2d5161c1fe708551084e7.json

generic.secrets.security.detected-generic-api-key.detected-generic-api-key

Generic API Key detected

Details: https://sg.run/qxj8

1┆ ... ApiKey = '84842078b09946638c03157f83405213';\nconst alchemyRpcUrls = {\n mainnet:

'https://eth-mainnet.alchemyapi.io/v2',\n ropsten: 'https://eth ...

[shortened a long line from output, adjust with --max-chars-per-line]

frontend/src/app/app.guard.spec.ts

generic.secrets.security.detected-jwt-token.detected-jwt-token

JWT token detected

Details: https://sg.run/05N5

40┆ ...

eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6IkpvaG4gRG9lIiwiaWF0IjoxNTE2MjM5MDIyf

Q.SflKxwRJSMeKKF2QT4fwpMeJf36POk6yJV\_ ...

[shortened a long line from output, adjust with --max-chars-per-line]

frontend/src/app/last-login-ip/last-login-ip.component.spec.ts

generic.secrets.security.detected-jwt-token.detected-jwt-token

JWT token detected

Details: https://sg.run/05N5

50┆ ...

eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJkYXRhIjp7Imxhc3RMb2dpbklwIjoiMS4yLjMuNCJ9fQ.RAkmdqwNypuOxv3SDjPO4x

MKvd1CddKvDFYDBfUt3bg') ...

[shortened a long line from output, adjust with --max-chars-per-line]

⋮┆----------------------------------------

56┆ localStorage.setItem('token',

'eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJkYXRhIjp7fX0.bVBhvll6IaeR3aUdoOeyR8YZe2S2DfhGAxTGfd9enLw')

frontend/src/app/search-result/search-result.component.ts

typescript.angular.angular-route-bypass-security-trust.angular-route-bypass-security-trust

Untrusted input could be used to tamper with a web page rendering, which can lead to a

Cross-site scripting (XSS) vulnerability. XSS vulnerabilities occur when untrusted input

executes malicious JavaScript code, leading to issues such as account compromise and

sensitive information leakage. Validate the user input, perform contextual output encoding,

or sanitize the input. A popular library used to prevent XSS is DOMPurify. You can also use

libraries and frameworks such as Angular, Vue, and React, which offer secure defaults when

rendering input.

Details: https://sg.run/JpBW

151┆ this.searchValue = this.sanitizer.bypassSecurityTrustHtml(queryParam) // vuln-code-snippet

vuln-line localXssChallenge xssBonusChallenge

frontend/src/hacking-instructor/helpers/helpers.ts

javascript.lang.security.audit.prototype-pollution.prototype-pollution-loop.prototype-pollution-

loop

Possibility of prototype polluting function detected. By adding or modifying attributes of

an object prototype, it is possible to create attributes that exist on every object, or

replace critical attributes with malicious ones. This can be problematic if the software

depends on existence or non-existence of certain attributes, or uses pre-defined attributes

of object prototype (such as hasOwnProperty, toString or valueOf). Possible mitigations

might be: freezing the object prototype, using an object without prototypes (via

Object.create(null) ), blocking modifications of attributes that resolve to object

prototype, using Map instead of object.

Details: https://sg.run/w1DB

36┆ replacementValue = replacementValue[property]

frontend/src/hacking-instructor/index.ts

javascript.browser.security.insecure-document-method.insecure-document-method

User controlled data in methods like `innerHTML`, `outerHTML` or `document.write` is an

anti-pattern that can lead to XSS vulnerabilities

Details: https://sg.run/LwA9

107┆ textBox.innerHTML = snarkdown(hint.text)

frontend/src/index.html

html.security.audit.missing-integrity.missing-integrity

This tag is missing an 'integrity' subresource integrity attribute. The 'integrity'

attribute allows for the browser to verify that externally hosted files (for example from a

CDN) are delivered without unexpected manipulation. Without this attribute, if an attacker

can modify the externally hosted resource, this could lead to XSS and other types of

attacks. To prevent this, include the base64-encoded cryptographic hash of the resource

(file) you’re telling the browser to fetch in the 'integrity' attribute for all externally

hosted files.

Details: https://sg.run/krXA

14┆ <link rel="stylesheet" type="text/css"

href="//cdnjs.cloudflare.com/ajax/libs/cookieconsent2/3.1.0/cookieconsent.min.css" />

⋮┆----------------------------------------

15┆ <script

src="//cdnjs.cloudflare.com/ajax/libs/cookieconsent2/3.1.0/cookieconsent.min.js"></script>

⋮┆----------------------------------------

16┆ <script src="//cdnjs.cloudflare.com/ajax/libs/jquery/2.2.4/jquery.min.js"></script>

lib/codingChallenges.ts

javascript.lang.security.audit.path-traversal.path-join-resolve-traversal.path-join-resolve-

traversal

Detected possible user input going into a `path.join` or `path.resolve` function. This could

possibly lead to a path traversal vulnerability, where the attacker can access arbitrary

files stored in the file system. Instead, be sure to sanitize or validate user input first.

Details: https://sg.run/OPqk

24┆ files.map(file => path.resolve(currPath, file))

⋮┆----------------------------------------

24┆ files.map(file => path.resolve(currPath, file))

⋮┆----------------------------------------

javascript.lang.security.audit.detect-non-literal-regexp.detect-non-literal-regexp

RegExp() called with a `challengeKey` function argument, this might allow an attacker to

cause a Regular Expression Denial-of-Service (ReDoS) within your application as RegExP

blocks the main thread. For this reason, it is recommended to use hardcoded regexes instead.

If your regex is run on user-controlled input, consider performing input validation or use a

regex checking/sanitization library such as https://www.npmjs.com/package/recheck to verify

that the regex does not appear vulnerable to ReDoS.

Details: https://sg.run/gr65

76┆ if (new RegExp(`vuln-code-snippet vuln-line.\*${challengeKey}`).exec(lines[i]) != null) {

⋮┆----------------------------------------

78┆ } else if (new RegExp(`vuln-code-snippet neutral-line.\*${challengeKey}`).exec(lines[i]) !=

null) {

lib/insecurity.ts

javascript.lang.hardcoded.strings.detected-private-key.detected-private-key

A secret is hard-coded in the application. Secrets stored in source code, such as

credentials, identifiers, and other types of sensitive data, can be leaked and used by

internal or external malicious actors. Use environment variables to securely provide

credentials and other secrets or retrieve them from a secure vault or Hardware Security

Module (HSM).

Details: https://sg.run/2dRY

23┆ ... -----BEGIN RSA PRIVATE

KEY-----\r\nMIICXAIBAAKBgQDNwqLEe9wgTXCbC7+RPdDbBbeqjdbs4kOPOIGzqLpXvJXlxxW8iMz0EaM4BKUqYsIa+ndv3NAn2RxCd5ubVdJJcX43zO

6Ko0TFEZx/ ...

[shortened a long line from output, adjust with --max-chars-per-line]

⋮┆----------------------------------------

contrib.nodejsscan.crypto\_node.node\_md5

The MD5 hashing algorithm is considered to be weak. If this is used in any sensitive

operation such as password hashing, or is used to ensure data integrity (collision

sensitive) then you should use a stronger hashing algorithm. For passwords, consider using

`Argon2id`, `scrypt`, or `bcrypt`. For data integrity, consider using `SHA-256`

Details: https://sg.run/dKBX

43┆ export const hash = (data: string) => crypto.createHash('md5').update(data).digest('hex')

⋮┆----------------------------------------

javascript.lang.security.audit.hardcoded-hmac-key.hardcoded-hmac-key

Detected a hardcoded hmac key. Avoid hardcoding secrets and consider using an alternate

option such as reading the secret from a config file or using an environment variable.

Details: https://sg.run/K9bn

44┆ export const hmac = (data: string) => crypto.createHmac('sha256',

'pa4qacea4VK9t9nGv7yZtwmj').update(data).digest('hex')

⋮┆----------------------------------------

javascript.jsonwebtoken.security.jwt-hardcode.hardcoded-jwt-secret

A hard-coded credential was detected. It is not recommended to store credentials in source-

code, as this risks secrets being leaked and used by either an internal or external

malicious adversary. It is recommended to use environment variables to securely provide

credentials or retrieve credentials from a secure vault or HSM (Hardware Security Module).

Details: https://sg.run/4xN9

56┆ export const authorize = (user = {}) => jwt.sign(user, privateKey, { expiresIn: '6h',

algorithm: 'RS256' })

⋮┆----------------------------------------

javascript.lang.security.audit.hardcoded-hmac-key.hardcoded-hmac-key

Detected a hardcoded hmac key. Avoid hardcoding secrets and consider using an alternate

option such as reading the secret from a config file or using an environment variable.

Details: https://sg.run/K9bn

158┆ const hmac = crypto.createHmac('sha256', privateKey)

⋮┆----------------------------------------

javascript.express.session-fixation.session-fixation

Detected `req` argument which enters `res.cookie`, this can lead to session fixation

vulnerabilities if an attacker can control the cookie value. This vulnerability can lead to

unauthorized access to accounts, and in some esoteric cases, Cross-Site-Scripting (XSS).

Users should not be able to influence cookies directly, for session cookies, they should be

generated securely using an approved session management library. If the cookie does need to

be set by a user, consider using an allow-list based approach to restrict the cookies which

can be set.

Details: https://sg.run/0qDv

201┆ res.cookie('token', token)

lib/startup/restoreOverwrittenFilesWithOriginals.ts

javascript.lang.security.audit.path-traversal.path-join-resolve-traversal.path-join-resolve-

traversal

Detected possible user input going into a `path.join` or `path.resolve` function. This could

possibly lead to a path traversal vulnerability, where the attacker can access arbitrary

files stored in the file system. Instead, be sure to sanitize or validate user input first.

Details: https://sg.run/OPqk

30┆ files.map((filename: string) => copyFile(filename, path.resolve('i18n/',

filename.substring(filename.lastIndexOf('/') + 1))))

lib/startup/validatePreconditions.ts

javascript.lang.security.audit.path-traversal.path-join-resolve-traversal.path-join-resolve-

traversal

Detected possible user input going into a `path.join` or `path.resolve` function. This could

possibly lead to a path traversal vulnerability, where the attacker can access arbitrary

files stored in the file system. Instead, be sure to sanitize or validate user input first.

Details: https://sg.run/OPqk

116┆ return access(path.resolve(pathRelativeToProjectRoot)).then(() => {

lib/utils.ts

javascript.jssha.jssha-sha1.jssha-sha1

The SHA1 hashing algorithm is considered to be weak. If this is used in any sensitive

operation such as password hashing, or is used to ensure data integrity (collision

sensitive) then you should use a stronger hashing algorithm. For passwords, consider using

`Argon2id`, `scrypt`, or `bcrypt`. For data integrity, consider using `SHA-256`.

Details: https://sg.run/ERyN

97┆ const shaObj = new jsSHA('SHA-1', 'TEXT') // eslint-disable-line new-cap

models/index.ts

javascript.sequelize.node-sequelize-hardcoded-secret-argument.node-sequelize-hardcoded-secret-

argument

A secret is hard-coded in the application. Secrets stored in source code, such as

credentials, identifiers, and other types of sensitive data, can be leaked and used by

internal or external malicious actors. Use environment variables to securely provide

credentials and other secrets or retrieve them from a secure vault or Hardware Security

Module (HSM).

Details: https://sg.run/E7ZB

31┆ const sequelize = new Sequelize('database', 'username', 'password', {

routes/b2bOrder.ts

javascript.express.security.audit.express-detect-notevil-usage.express-detect-notevil-usage

Detected usage of the `notevil` package, which is unmaintained and has vulnerabilities.

Using any sort of `eval()` functionality can be very dangerous, but if you must, the `eval`

package is an up to date alternative. Be sure that only trusted input reaches an `eval()`

function.

Details: https://sg.run/W70E

22┆ vm.runInContext('safeEval(orderLinesData)', sandbox, { timeout: 2000 })

⋮┆----------------------------------------

javascript.lang.security.audit.vm-injection.vm-runincontext-context-injection

Make sure that unverified user data can not reach vm.runInContext.

Details: https://sg.run/9oey

22┆ vm.runInContext('safeEval(orderLinesData)', sandbox, { timeout: 2000 })

routes/captcha.ts

javascript.browser.security.eval-detected.eval-detected

Detected the use of eval(). eval() can be dangerous if used to evaluate dynamic content. If

this content can be input from outside the program, this may be a code injection

vulnerability. Ensure evaluated content is not definable by external sources.

Details: https://sg.run/7ope

23┆ const answer = eval(expression).toString() // eslint-disable-line no-eval

routes/dataErasure.ts

javascript.express.security.audit.express-path-join-resolve-traversal.express-path-join-resolve-

traversal

Possible writing outside of the destination, make sure that the target path is nested in the

intended destination

Details: https://sg.run/weRn

69┆ const filePath: string = path.resolve(req.body.layout).toLowerCase()

⋮┆----------------------------------------

javascript.lang.security.audit.path-traversal.path-join-resolve-traversal.path-join-resolve-

traversal

Detected possible user input going into a `path.join` or `path.resolve` function. This could

possibly lead to a path traversal vulnerability, where the attacker can access arbitrary

files stored in the file system. Instead, be sure to sanitize or validate user input first.

Details: https://sg.run/OPqk

69┆ const filePath: string = path.resolve(req.body.layout).toLowerCase()

routes/dataExport.ts

javascript.express.mongodb.express-mongo-nosqli.express-mongo-nosqli

Detected a `../data/mongodb` statement that comes from a `req` argument. This could lead to

NoSQL injection if the variable is user-controlled and is not properly sanitized. Be sure to

properly sanitize the data if you absolutely must pass request data into a mongo query.

Details: https://sg.run/Qqxq

61┆ db.orders.find({ email: updatedEmail }).then((orders: Array<{

routes/fileServer.ts

javascript.express.security.audit.express-res-sendfile.express-res-sendfile

The application processes user-input, this is passed to res.sendFile which can allow an

attacker to arbitrarily read files on the system through path traversal. It is recommended

to perform input validation in addition to canonicalizing the path. This allows you to

validate the path against the intended directory it should be accessing.

Details: https://sg.run/7DJk

33┆ res.sendFile(path.resolve('ftp/', file))

⋮┆----------------------------------------

javascript.lang.security.audit.path-traversal.path-join-resolve-traversal.path-join-resolve-

traversal

Detected possible user input going into a `path.join` or `path.resolve` function. This could

possibly lead to a path traversal vulnerability, where the attacker can access arbitrary

files stored in the file system. Instead, be sure to sanitize or validate user input first.

Details: https://sg.run/OPqk

33┆ res.sendFile(path.resolve('ftp/', file))

routes/fileUpload.ts

javascript.lang.security.audit.path-traversal.path-join-resolve-traversal.path-join-resolve-

traversal

Detected possible user input going into a `path.join` or `path.resolve` function. This could

possibly lead to a path traversal vulnerability, where the attacker can access arbitrary

files stored in the file system. Instead, be sure to sanitize or validate user input first.

Details: https://sg.run/OPqk

29┆ const tempFile = path.join(os.tmpdir(), filename)

⋮┆----------------------------------------

39┆ const absolutePath = path.resolve('uploads/complaints/' + fileName)

⋮┆----------------------------------------

javascript.express.security.audit.express-libxml-vm-noent.express-libxml-vm-noent

Detected use of parseXml() function with the `noent` field set to `true`. This can lead to

an XML External Entities (XXE) attack if untrusted data is passed into it.

Details: https://sg.run/n8Ag

80┆ const xmlDoc = vm.runInContext('libxml.parseXml(data, { noblanks: true, noent: true, nocdata: true

})', sandbox, { timeout: 2000 })

⋮┆----------------------------------------

javascript.lang.security.audit.vm-injection.vm-runincontext-context-injection

Make sure that unverified user data can not reach vm.runInContext.

Details: https://sg.run/9oey

80┆ const xmlDoc = vm.runInContext('libxml.parseXml(data, { noblanks: true, noent: true, nocdata: true })',

sandbox, { timeout: 2000 })

routes/keyServer.ts

javascript.express.security.audit.express-res-sendfile.express-res-sendfile

The application processes user-input, this is passed to res.sendFile which can allow an

attacker to arbitrarily read files on the system through path traversal. It is recommended

to perform input validation in addition to canonicalizing the path. This allows you to

validate the path against the intended directory it should be accessing.

Details: https://sg.run/7DJk

14┆ res.sendFile(path.resolve('encryptionkeys/', file))

⋮┆----------------------------------------

javascript.express.security.audit.express-path-join-resolve-traversal.express-path-join-resolve-

traversal

Possible writing outside of the destination, make sure that the target path is nested in the

intended destination

Details: https://sg.run/weRn

14┆ res.sendFile(path.resolve('encryptionkeys/', file))

⋮┆----------------------------------------

javascript.lang.security.audit.path-traversal.path-join-resolve-traversal.path-join-resolve-

traversal

Detected possible user input going into a `path.join` or `path.resolve` function. This could

possibly lead to a path traversal vulnerability, where the attacker can access arbitrary

files stored in the file system. Instead, be sure to sanitize or validate user input first.

Details: https://sg.run/OPqk

14┆ res.sendFile(path.resolve('encryptionkeys/', file))

routes/likeProductReviews.ts

javascript.express.mongodb.express-mongo-nosqli.express-mongo-nosqli

Detected a `../data/mongodb` statement that comes from a `req` argument. This could lead to

NoSQL injection if the variable is user-controlled and is not properly sanitized. Be sure to

properly sanitize the data if you absolutely must pass request data into a mongo query.

Details: https://sg.run/Qqxq

18┆ db.reviews.findOne({ \_id: id }).then((review: Review) => {

⋮┆----------------------------------------

25┆ { \_id: id },

⋮┆----------------------------------------

31┆ db.reviews.findOne({ \_id: id }).then((review: Review) => {

⋮┆----------------------------------------

42┆ { \_id: id },

routes/logfileServer.ts

javascript.express.security.audit.express-res-sendfile.express-res-sendfile

The application processes user-input, this is passed to res.sendFile which can allow an

attacker to arbitrarily read files on the system through path traversal. It is recommended

to perform input validation in addition to canonicalizing the path. This allows you to

validate the path against the intended directory it should be accessing.

Details: https://sg.run/7DJk

14┆ res.sendFile(path.resolve('logs/', file))

⋮┆----------------------------------------

javascript.express.security.audit.express-path-join-resolve-traversal.express-path-join-resolve-

traversal

Possible writing outside of the destination, make sure that the target path is nested in the

intended destination

Details: https://sg.run/weRn

14┆ res.sendFile(path.resolve('logs/', file))

⋮┆----------------------------------------

javascript.lang.security.audit.path-traversal.path-join-resolve-traversal.path-join-resolve-

traversal

Detected possible user input going into a `path.join` or `path.resolve` function. This could

possibly lead to a path traversal vulnerability, where the attacker can access arbitrary

files stored in the file system. Instead, be sure to sanitize or validate user input first.

Details: https://sg.run/OPqk

14┆ res.sendFile(path.resolve('logs/', file))

routes/login.ts

javascript.sequelize.security.audit.sequelize-injection-express.express-sequelize-injection

Detected a sequelize statement that is tainted by user-input. This could lead to SQL

injection if the variable is user-controlled and is not properly sanitized. In order to

prevent SQL injection, it is recommended to use parameterized queries or prepared

statements.

Details: https://sg.run/gjoe

36┆ ... `SELECT \* FROM Users WHERE email = '${req.body.email || ''}' AND password =

'${security.hash(req.body.password || '')}' AND deletedAt IS NULL`, { m ...

[shortened a long line from output, adjust with --max-chars-per-line]

routes/order.ts

javascript.lang.security.audit.path-traversal.path-join-resolve-traversal.path-join-resolve-

traversal

Detected possible user input going into a `path.join` or `path.resolve` function. This could

possibly lead to a path traversal vulnerability, where the attacker can access arbitrary

files stored in the file system. Instead, be sure to sanitize or validate user input first.

Details: https://sg.run/OPqk

46┆ const fileWriter = doc.pipe(fs.createWriteStream(path.join('ftp/', pdfFile)))

routes/profileImageFileUpload.ts

javascript.express.express-fs-filename.express-fs-filename

The application builds a file path from potentially untrusted data, which can lead to a path

traversal vulnerability. An attacker can manipulate the file path which the application uses

to access files. If the application does not validate user input and sanitize file paths,

sensitive files such as configuration or user data can be accessed, potentially creating or

overwriting files. To prevent this vulnerability, validate and sanitize any input that is

used to create references to file paths. Also, enforce strict file access controls. For

example, choose privileges allowing public-facing applications to access only the required

files.

Details: https://sg.run/0B9W

28┆

fs.open(`frontend/dist/frontend/assets/public/images/uploads/${loggedInUser.data.id}.${uploadedFileType.ext}

`, 'w', function (err, fd) {

routes/profileImageUrlUpload.ts

javascript.express.request.ssrf-deepsemgrep.ssrf-deepsemgrep

Untrusted input might be used to build an HTTP request, which can lead to a Server-side

request forgery (SSRF) vulnerability. SSRF allows an attacker to send crafted requests from

the server side to other internal or external systems. SSRF can lead to unauthorized access

to sensitive data and, in some cases, allow the attacker to control applications or systems

that trust the vulnerable service. To prevent this vulnerability, avoid allowing user input

to craft the base request. Instead, treat it as part of the path or query parameter and

encode it appropriately. When user input is necessary to prepare the HTTP request, perform

strict input validation. Additionally, whenever possible, use allowlists to only interact

with expected, trusted domains.

Details: https://sg.run/Wxn4

23┆ .get(url)

⋮┆----------------------------------------

javascript.express.security.audit.express-ssrf.express-ssrf

The following request request.get() was found to be crafted from user-input `req` which can

lead to Server-Side Request Forgery (SSRF) vulnerabilities. It is recommended where possible

to not allow user-input to craft the base request, but to be treated as part of the path or

query parameter. When user-input is necessary to craft the request, it is recommeneded to

follow OWASP best practices to prevent abuse.

Details: https://sg.run/0PNw

23┆ .get(url)

⋮┆----------------------------------------

javascript.express.express-fs-filename.express-fs-filename

The application builds a file path from potentially untrusted data, which can lead to a path

traversal vulnerability. An attacker can manipulate the file path which the application uses

to access files. If the application does not validate user input and sanitize file paths,

sensitive files such as configuration or user data can be accessed, potentially creating or

overwriting files. To prevent this vulnerability, validate and sanitize any input that is

used to create references to file paths. Also, enforce strict file access controls. For

example, choose privileges allowing public-facing applications to access only the required

files.

Details: https://sg.run/0B9W

31┆

imageRequest.pipe(fs.createWriteStream(`frontend/dist/frontend/assets/public/images/uploads/${loggedInUser.data

id}.${ext}`))

routes/quarantineServer.ts

javascript.express.security.audit.express-res-sendfile.express-res-sendfile

The application processes user-input, this is passed to res.sendFile which can allow an

attacker to arbitrarily read files on the system through path traversal. It is recommended

to perform input validation in addition to canonicalizing the path. This allows you to

validate the path against the intended directory it should be accessing.

Details: https://sg.run/7DJk

14┆ res.sendFile(path.resolve('ftp/quarantine/', file))

⋮┆----------------------------------------

javascript.express.security.audit.express-path-join-resolve-traversal.express-path-join-resolve-

traversal

Possible writing outside of the destination, make sure that the target path is nested in the

intended destination

Details: https://sg.run/weRn

14┆ res.sendFile(path.resolve('ftp/quarantine/', file))

⋮┆----------------------------------------

javascript.lang.security.audit.path-traversal.path-join-resolve-traversal.path-join-resolve-

traversal

Detected possible user input going into a `path.join` or `path.resolve` function. This could

possibly lead to a path traversal vulnerability, where the attacker can access arbitrary

files stored in the file system. Instead, be sure to sanitize or validate user input first.

Details: https://sg.run/OPqk

14┆ res.sendFile(path.resolve('ftp/quarantine/', file))

routes/redirect.ts

javascript.express.security.audit.express-open-redirect.express-open-redirect

The application redirects to a URL specified by user-supplied input `query` that is not

validated. This could redirect users to malicious locations. Consider using an allow-list

approach to validate URLs, or warn users they are being redirected to a third-party website.

Details: https://sg.run/EpoP

19┆ res.redirect(toUrl as string)

⋮┆----------------------------------------

javascript.express.open-redirect-deepsemgrep.open-redirect-deepsemgrep

The application builds a URL using user-controlled input which can lead to an open redirect

vulnerability. An attacker can manipulate the URL and redirect users to an arbitrary domain.

Open redirect vulnerabilities can lead to issues such as Cross-site scripting (XSS) or

redirecting to a malicious domain for activities such as phishing to capture users'

credentials. To prevent this vulnerability perform strict input validation of the domain

against an allowlist of approved domains. Notify a user in your application that they are

leaving the website. Display a domain where they are redirected to the user. A user can then

either accept or deny the redirect to an untrusted site.

Details: https://sg.run/BDbW

19┆ res.redirect(toUrl as string)

routes/search.ts

javascript.sequelize.security.audit.sequelize-injection-express.express-sequelize-injection

Detected a sequelize statement that is tainted by user-input. This could lead to SQL

injection if the variable is user-controlled and is not properly sanitized. In order to

prevent SQL injection, it is recommended to use parameterized queries or prepared

statements.

Details: https://sg.run/gjoe

23┆ ... `SELECT \* FROM Products WHERE ((name LIKE '%${criteria}%' OR description LIKE

'%${criteria}%') AND deletedAt IS NULL) ORDER BY name`) // vuln-code- ...

[shortened a long line from output, adjust with --max-chars-per-line]

routes/showProductReviews.ts

javascript.express.mongodb.express-mongo-nosqli.express-mongo-nosqli

Detected a `../data/mongodb` statement that comes from a `req` argument. This could lead to

NoSQL injection if the variable is user-controlled and is not properly sanitized. Be sure to

properly sanitize the data if you absolutely must pass request data into a mongo query.

Details: https://sg.run/Qqxq

34┆ db.reviews.find({ $where: 'this.product == ' + id }).then((reviews: Review[]) => {

routes/trackOrder.ts

javascript.express.mongodb.express-mongo-nosqli.express-mongo-nosqli

Detected a `../data/mongodb` statement that comes from a `req` argument. This could lead to

NoSQL injection if the variable is user-controlled and is not properly sanitized. Be sure to

properly sanitize the data if you absolutely must pass request data into a mongo query.

Details: https://sg.run/Qqxq

18┆ db.orders.find({ $where: `this.orderId === '${id}'` }).then((order: any) => {

routes/updateProductReviews.ts

javascript.express.mongodb.express-mongo-nosqli.express-mongo-nosqli

Detected a `../data/mongodb` statement that comes from a `req` argument. This could lead to

NoSQL injection if the variable is user-controlled and is not properly sanitized. Be sure to

properly sanitize the data if you absolutely must pass request data into a mongo query.

Details: https://sg.run/Qqxq

18┆ { \_id: req.body.id }, // vuln-code-snippet vuln-line noSqlReviewsChallenge

forgedReviewChallenge

routes/userProfile.ts

javascript.browser.security.eval-detected.eval-detected

Detected the use of eval(). eval() can be dangerous if used to evaluate dynamic content. If

this content can be input from outside the program, this may be a code injection

vulnerability. Ensure evaluated content is not definable by external sources.

Details: https://sg.run/7ope

36┆ username = eval(code) // eslint-disable-line no-eval

⋮┆----------------------------------------

javascript.express.security.express-insecure-template-usage.express-insecure-template-usage

User data from `req` is being compiled into the template, which can lead to a Server Side

Template Injection (SSTI) vulnerability.

Details: https://sg.run/b49v

56┆ const fn = pug.compile(template)

routes/videoHandler.ts

javascript.lang.security.audit.unknown-value-with-script-tag.unknown-value-with-script-tag

Cannot determine what 'subs' is and it is used with a '<script>' tag. This could be

susceptible to cross-site scripting (XSS). Ensure 'subs' is not externally controlled, or

sanitize this data.

Details: https://sg.run/1Zy1

57┆ challengeUtils.solveIf(challenges.videoXssChallenge, () => { return utils.contains(subs,

'</script><script>alert(`xss`)</script>') })

⋮┆----------------------------------------

69┆ ... subs + '</script>') ...

[shortened a long line from output, adjust with --max-chars-per-line]

routes/vulnCodeFixes.ts

javascript.express.express-fs-filename.express-fs-filename

The application builds a file path from potentially untrusted data, which can lead to a path

traversal vulnerability. An attacker can manipulate the file path which the application uses

to access files. If the application does not validate user input and sanitize file paths,

sensitive files such as configuration or user data can be accessed, potentially creating or

overwriting files. To prevent this vulnerability, validate and sanitize any input that is

used to create references to file paths. Also, enforce strict file access controls. For

example, choose privileges allowing public-facing applications to access only the required

files.

Details: https://sg.run/0B9W

79┆ if (fs.existsSync('./data/static/codefixes/' + key + '.info.yml')) {

⋮┆----------------------------------------

80┆ const codingChallengeInfos = yaml.load(fs.readFileSync('./data/static/codefixes/' + key +

'.info.yml', 'utf8'))

routes/vulnCodeSnippet.ts

javascript.express.express-fs-filename.express-fs-filename

The application builds a file path from potentially untrusted data, which can lead to a path

traversal vulnerability. An attacker can manipulate the file path which the application uses

to access files. If the application does not validate user input and sanitize file paths,

sensitive files such as configuration or user data can be accessed, potentially creating or

overwriting files. To prevent this vulnerability, validate and sanitize any input that is

used to create references to file paths. Also, enforce strict file access controls. For

example, choose privileges allowing public-facing applications to access only the required

files.

Details: https://sg.run/0B9W

93┆ if (fs.existsSync('./data/static/codefixes/' + key + '.info.yml')) {

⋮┆----------------------------------------

94┆ const codingChallengeInfos = yaml.load(fs.readFileSync('./data/static/codefixes/' + key +

'.info.yml', 'utf8'))

server.ts

javascript.express.security.audit.express-check-csurf-middleware-usage.express-check-csurf-

middleware-usage

A CSRF middleware was not detected in your express application. Ensure you are either using

one such as `csurf` or `csrf` (see rule references) and/or you are properly doing CSRF

validation in your routes with a token or cookies.

Details: https://sg.run/BxzR

96┆ const app = express()

⋮┆----------------------------------------

javascript.express.security.audit.express-check-directory-listing.express-check-directory-

listing

Directory listing/indexing is enabled, which may lead to disclosure of sensitive directories

and files. It is recommended to disable directory listing unless it is a public resource. If

you need directory listing, ensure that sensitive files are inaccessible when querying the

resource.

Details: https://sg.run/DX2G

245┆ app.use('/ftp', serveIndexMiddleware, serveIndex('ftp', { icons: true })) // vuln-code-snippet

vuln-line directoryListingChallenge

⋮┆----------------------------------------

250┆ app.use('/encryptionkeys', serveIndexMiddleware, serveIndex('encryptionkeys', { icons: true, view:

'details' }))

⋮┆----------------------------------------

254┆ app.use('/support/logs', serveIndexMiddleware, serveIndex('logs', { icons: true, view: 'details'

})) // vuln-code-snippet vuln-line accessLogDisclo ...

[shortened a long line from output, adjust with --max-chars-per-line]

views/promotionVideo.pug

javascript.express.security.audit.xss.pug.explicit-unescape.template-explicit-unescape

Detected an explicit unescape in a Pug template, using either '!=' or '!{...}'. If external

data can reach these locations, your application is exposed to a cross-site scripting (XSS)

vulnerability. If you must do this, ensure no external data can reach this location.

Details: https://sg.run/3xbe

79┆ if (splitted.length != 2) {

┌──────────────┐

│ Scan Summary │

└──────────────┘

Some files were skipped or only partially analyzed.

Partially scanned: 30 files only partially analyzed due to parsing or internal Semgrep errors

Scan skipped: 101 files larger than 1.0 MB, 123901 files matching .semgrepignore patterns

For a full list of skipped files, run semgrep with the --verbose flag.

Ran 401 rules on 1799 files: 86 findings.