SUBGRAPH

FINDINGS REPORT (FINAL)

Shira

July 31, 2023

Prepared for: Horizontal

Subgraph Technologies, Inc.
642 Rue de Courcelle, Suite 309
Montreal, Quebec
https://subgraph.com

Contents

Overview

Scope

The scope of activities conducted covered testing the Shira application for vulnerabilities and exposures. Shira is an application that hosts quizzes with customizable simulated phishing attacks; it is intended to help prevent successful targeted attacks through user education.

Shira includes:

- Shira API (NestJS) - The backend that serves the dynamic content and has basic management API endpoints.
- Shira Private (React): The management frontend used by operators of a Shira instance.
- Shira Public (React): The application frontend used by the end-users.

The key risks in scope for the security assessment are:

- Shira Private/Shira Public: DOM based XSS and other DOM tampering that could be used to target users of the application in a way where they are mislead.
- Shira API: Implementation, design, and/or configuration issues in the API backend that could lead to breach of the administration interface.

The key simulated attack scenario used in this engagement will be of a determined adversary who wishes to breach the backend application server (Shira API). A secondary attack scenario targets the two different classes of users of the application (Shira Public and Shira Private).

The threat model assumes the following:

1. An adversary of moderate to high skill level that has unauthenticated access to the API backend. Their objective is to obtain access to the server.
2. An adversary of moderate to high skill level that has authenticated access to the API backend. Their objectives are two-fold: gain access to the underlying server, or, target the user through the use of malicious content.

Artifacts Examined

There are three repositories that comprise Shira. Subgraph both deployed and reviewed the following:

- Shira API: commit cf549e1d0a421cb94002b6b152c1f35274ca5c01
- Shira public: commit 871fccebf1340e98b79f0868f6faafa37dd869d7
- Shira private: commit 4b6c048f34500841043f5ab4b30e2bc8df1d3b17

Duration of testing

All activities related to this engagement occured within a total span of 56 hours. Testing comprised approximately 40 hours of this.

Functional Components and Key Features

- Question management frontend (shira private)
- API backend (shira api)
- Quiz frontend (shira public)

Tools used

- BurpSuite Pro
- VSCode
- Various additional Linux open source tools
- Chrome, Firefox
- npm

Security Evaluation Criteria

Subgraph used a methodology that included test cases that covered the following:

- Input validation errors such as SQL injection, template injection, and cross-site scripting
- Server and client-side prototype pollution
- Examination of the authentication mechanism
- Authorization mechanism
- Third-party dependency vulnerability management
- Internal information leakage through error conditions exceptions, etc
- Any use of cryptography

Subgraph also performed limited vulnerability assessment of the observed dependencies and infrastructure.

Summary

No.	Title	Severity	Remediation
V-001	Account Existence Oracle	Medium	Resolved
V-002	No Logout Capability	Low	Resolved
V-003	Vulnerable Javascript Dependences: Shira Public	Low	Unresolved
V-004	Vulnerable Javascript Dependences: Shira Private	Low	Unresolved
V-005	Vulnerable Javascript Dependences: Shira API	Low	Unresolved
V-006	Denial of Service Conditions due to Unhandled Exceptions	Informational	Unresolved

General Observations

Authentication

The following only applies to Shira private/Shira API:

Authentication is performed using passwords, which are set in advance when users are created. Users are created with the user commander CLI interface. Passwords are hashed using bcrypt with a number of salt generation rounds that can either be provided in an environment variable, or the numeric value 10, which is the default for node.bcrypt.

Individual client requests are authenticated using a token that is minted with a 1-day expiration when the user authenticates successfully. The token is signed with a secret that is expected to be passed into the runtime environment of Shira API as environment variable *JWT_SECRET*.

See *src/modules/auth/services/generate-token.auth.service.ts* and *src/modules/auth/strategy/jwt.auth.strategy.ts*.

At the time of testing, there was no way to log out in either the API backend or the Shira private frontend. Therefore users may inadvertently be exposed for the duration of token validity. This was corrected in an update subsequent to submission of this report.

Shira public does not have any authentication and no concept of a user session is applicable.

Authorization

The authorization model in Shira is very simple: there are only two classes of user. The user, which can be considered an administrator for quizzes, and anonymous users. Authenticated users can create and edit quiz questions. There is no distinction between users from an access control perspective, i.e., user *ab@ab.com* can edit content created by *fd@fd.com*. This is presumed to be by design.

Authorization is enforced using a decorator for methods that require authentication.

See *src/utils/decorator/auth-controller.decorator.ts*.

Input Validation

The framework protections are very strong for handling of input. HTML escaping is applied consistently. The There were no instances of SQL injection identified, and no command injection vulnerabilities identified, such as those that might be exposed by server-side template injection issues identified in dependencies.

Details

V-001: Account Existence Oracle

Severity	Remediation
Medium	Resolved

Discussion

It is possible to use timing as a side channel to verify that an email address is used for a user account in the private API backend. This is due to the way that the password authentication is implemented.

See the following code from *./src/modules/user/applications/check-password.user.application.ts*:

```
export class CheckPasswordUserApplication
  implements ICheckPasswordUserApplication {
  constructor(
    @Inject(TYPES.services.IFindByUsernameUserService)
    private readonly findByUsernameUserService: IFindByUsernameUserService,
  ) {}

  async execute(userCredentials: CredentialUserDto): Promise<ReadUserDto> {
    const errors = await validate(userCredentials);
    if (errors.length > 0) throw new InvalidCredentailsUserException();

    const user = await this.findByUsernameUserService.execute(
      userCredentials.email,
    );
    if (!user) throw new InvalidCredentailsUserException();

    const isValid = await comparePassword(
      userCredentials.password,
      user.password,
    );

    if (!isValid) throw new InvalidCredentailsUserException();

    return plainToClass(ReadUserDto, user);
  }
}
```

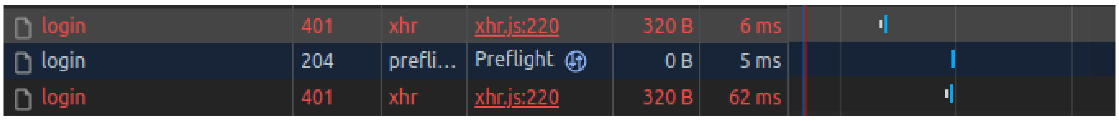
The *execute()* method first validates that the user exists, using the *findByUsernameUserService.execute()* method. If the uer does not exist, an exception is thrown and the authentication flow ends. If the submitted email address is associated with a user, the password is compared.

Shira hashes passwords using bcrypt to hash passwords. To authenticate a user, the submitted password is hashed and the two hashes are compared. This has cost in computation that is measurable as response time by the client.

Because *bcrypt()* is only invoked if the e-mail address is located, this serves as an oracle for distinguishing valid/invalid usernames.

This was tested, with the following response times observed when making POST requests to the */login* endpoint:

- Invalid email address: 3ms
- Valid email address, but invalid password: 62ms



login	401	xhr	xhr.js:220	320 B	6 ms
login	204	prefli...	Preflight	0 B	5 ms
login	401	xhr	xhr.js:220	320 B	62 ms

Impact Analysis

Adversaries can guess email addresses used to administer a Shira deployment.

Compounding this issue is that there are no limits on attempted authentication.

Remediation Recommendations

Calling *bcrypt()* on the supplied password regardless of whether or not the user exists may make it more difficult to distinguish whether or not accounts exist.

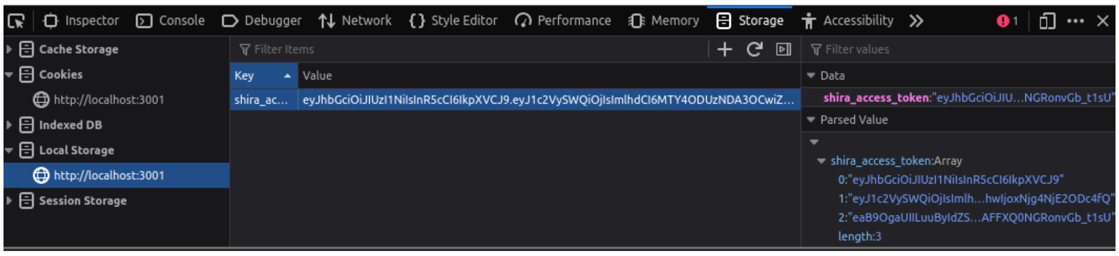
Update: This was remediated in an update that performs the password comparison even if the user does not exist.

V-002: No Logout Capability

Severity	Remediation
Low	Resolved

Discussion

There is no way to invalidate an authenticated user session within the frontend (nor via the API). With token based authentication, it is common for there to at least be a way to remove the token from the local storage so that the user is actually *logged out* of an active session when they desire to be so.



Key	Value
shira_ac...	eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ1c2VySWQiOjIsImlhdCI6MTY4ODUzNDA3OCwiZ...

```
Data
  shira_access_token:"eyJhbGciOiJIU...NGRonvGb_t1sU"
Parsed Value
  shira_access_token:Array
    0:"eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9"
    1:"eyJ1c2VySWQiOjIsImlh...hwljoxNjg4NjE2ODc4fQ"
    2:"eaB9OgaUIlLuuByldZS...AFFXQ0NGRonvGb_t1sU"
    length:3
```

Impact Analysis

A valid session token will persist in the browser local storage without any app-supported method to remove it.

Adversaries with physical access to the desktop may be able to access an authenticated seession.

Remediation Recommendations

Horizontal should create a logout function in the frontend that removes the token from the local storage.

Update: This was remediated in an update that added a logout function to the *shira-private* frontend.

V-003: Vulnerable Javascript Dependences: Shira Public

Severity	Remediation
Low	Unresolved

Discussion

Running *npm audit* reports many issues in dependencies. While possibly all of these issues are not exploitable within the fairly simple Shira public frontend, it is still recommended that third-party dependencies be kept current.

Below is a table of some of the issues reported within the dependencies with an estimation of whether or not these issues could affect Shira public based on a quick assessment.

Vulnerability	Severity	Impact	Affected
GHSA-phwq-j96m-2c2q	Critical	RCE	Unlikely
GHSA-w573-4hg7-7wgq	High	DoS	Unlikely
GHSA-3wcq-x3mq-6r9p	High	Infoleak	Unlikely
GHSA-ww39-953v-wcq6	High	Various	Unlikely
GHSA-9c47-m6qq-7p4h	High	Various	Unlikely
GHSA-f8q6-p94x-37v3	High	DoS	Unlikely
GHSA-rp65-9cf3-cjxr	High	DoS	Unlikely
GHSA-4wf5-vphf-c2xc	High	DoS	Unlikely
GHSA-w5p7-h5w8-2hfq	High	DoS	Unlikely
GHSA-7p7h-4mm5-852v	High	DoS	Unlikely
GHSA-c2qf-rxjj-qqgw	Moderate	DoS	Unlikely
GHSA-j8xg-fqg3-53r7	Moderate	DoS	Unlikely

Impact Analysis

Subgraph quickly reviewed these issues and was unable to reproduce any within the context of the Shira public frontend application. That does not mean they are not exploitable, just that no vector was located within the duration of the engagement. For example, the issue marked critical is likely not exploitable at all in this context due to it manifesting during processing of server side templates. However, this should not discourage the maintainer from ensuring upgrades are implemented for all third-party dependencies.

Remediation Recommendations

- Schedule an update of the dependency and associated testing for promotion to production
- Review the tracking third-party dependencies

Additional Information

N/A

V-004: Vulnerable Javascript Dependences: Shira Private

Severity	Remediation
Low	Unresolved

Discussion

Running *npm audit* reports many issues in dependencies. While possibly all of these issues are not exploitable within the fairly simple Shira administration frontend, it is still recommended that third-party dependencies be kept current.

Below is a table of some of the issues reported within the dependencies:

Vulnerability	Severity	Impact	Affected
GHSA-76p3-8jx3-jpfq	Critical	Various	Unlikely
GHSA-hhq3-ff78-jv3g	High	DoS	Unlikely
GHSA-3rfm-jhwj-7488	High	DoS	Unlikely
GHSA-f8q6-p94x-37v3	High	DoS	Unlikely
GHSA-rp65-9cf3-cjxr	High	DoS	Unlikely
GHSA-9c47-m6qq-7p4h	High	Various	Unlikely
GHSA-hc6q-2mpp-qw7j	High	Various	*Possibly*
GHSA-v339-96qg-c8rf	High	Various	Unlikely
GHSA-cwx2-736x-mf6w	High	Various	Unlikely
GHSA-8v63-cqqc-6r2c	High	Various	Unlikely
GHSA-j8xg-fqg3-53r7	Moderate	DoS	Unlikely
GHSA-c2qf-rxjj-qqgw	Moderate	DoS	Unlikely

Impact Analysis

Subgraph reviewed these issues and was unable to reproduce any within the context of the Shira application. That does not mean they are not exploitable, just that no vector was located within the duration of the engagement.

Remediation Recommendations

- Schedule an update of the dependency and associated testing for promotion to production
- Review the tracking third-party dependencies

Additional Information

Awesome Javascript Realms Security

V-005: Vulnerable Javascript Dependences: Shira API

Severity	Remediation
Low	Unresolved

Discussion

Running *npm audit* reports many issues in dependencies. While possibly all of these issues are not exploitable within the fairly simple Shira public private, it is still recommended that third-party dependencies be kept current.

Below is a table of some of the issues reported within the dependencies:

Vulnerability	Severity	Impact	Affected
GHSA-fj58-h2fr-3pp2	Critical	DoS	Unknown
GHSA-wm7h-9275-46v2	High	DoS	Unknown
GHSA-9c47-m6qq-7p4h	High	Various	*Possibly*
GHSA-hrpp-h998-j3pp	High	Various	*Possibly*
GHSA-4wf5-vphf-c2xc	High	DoS	Unlikely
GHSA-hc6q-2mpp-qw7j	High	Various	Unlikely
GHSA-776f-qx25-q3cc	High	Various	Unlikely
GHSA-8cf7-32gw-wr33	Moderate	Various	Unlikely
GHSA-hjrf-2m68-5959	Moderate	Various	Unlikely
GHSA-qwph-4952-7xr6	Moderate	Auth	*Possibly*
GHSA-j8xg-fqg3-53r7	Moderate	DoS	Unlikely
GHSA-c2qf-rxjj-qqgw	Moderate	DoS	Unknown

Impact Analysis

Subgraph reviewed these issues and was unable to reproduce any within the context of the Shira application. That does not mean they are not exploitable, just that no vector was located within the duration of the engagement. Subgraph tested for scenarios involving:

1. Prototype pollution at the login submission and during creation of questions and other data elements
2. JWT related issues, such as stripping signatures

These issues were considered to be the most serious of those reported by *npm audit*. In the case of JWT tampering, no method of bypassing authentication or authorization was identified during the course of the engagement. Also, no vector for exploitation of the prototype pollution issues were identified, nor were any useful gadgets. That does not mean the issues are not exploitable, as a thorough investigation could reveal vectors that were missed or are buried in third-party dependencies. It is strongly recommended that updates be kept current, especially for Shira API.

Remediation Recommendations

- Schedule an update of the dependency and associated testing for promotion to production
- Review the tracking third-party dependencies

Additional Information

N/A

V-006: Denial of Service Conditions due to Unhandled Exceptions

Severity	Remediation
Informational	Unresolved

Discussion

Exceptions generated by malformed client input cause the application to fail. It is quite trivial to cause the server to crash if an exception is generated. There are likely many instances of this and in a real world context they may not result in a persistent crashed state. However, during testing, each such exception required a restart of the application server runtime.

For example:

```
curl -d 'fdfds' -H 'Authorization: Bearer TOKEN' -i http://localhost:3000/question
```

Will crash the server with the following exception:

```
shira-api-dev | /usr/src/app/src/modules/question/services/create.question.service.ts:35
shira-api-dev |       where: { id: newQuestion.question.fieldOfWork },
shira-api-dev |                                         ^
shira-api-dev | TypeError: Cannot read properties of undefined (reading 'fieldOfWork')
shira-api-dev |     at CreateQuestionService.create (/usr/src/app/src/modules/question/
services/create.question.service.ts:35:41)
shira-api-dev |     at CreateQuestionController.handler (/usr/src/app/src/modules/
question/controllers/create.question.controller.ts:13:32)
shira-api-dev |     at /usr/src/app/node_modules/@nestjs/core/router/router-
execution-context.js:38:29
shira-api-dev |     at processTicksAndRejections (node:internal/process/task_queues:96:5
shira-api-dev |     at /usr/src/app/node_modules/@nestjs/core/router/router-
execution-context.js:46:28
shira-api-dev |     at /usr/src/app/node_modules/@nestjs/core/router/router-
proxy.js:9:17
```

In another instance, field fuzzing produced this unhandled exception:

```
shira-api-dev | /usr/src/app/src/error/TypeORMError.ts:7
shira-api-dev |         super(message);
shira-api-dev |         ^
shira-api-dev | QueryFailedError: Data too long for column 'name' at row 1
shira-api-dev |     at QueryFailedError.TypeORMError [as constructor] (/usr/src/app/src/
shira-api-dev |     at new QueryFailedError (/usr/src/app/src/error/QueryFailedError.ts:9
shira-api-dev |     at Query.onResult (/usr/src/app/src/driver/mysql/MysqlQueryRunner.ts
shira-api-dev |     at Query.execute (/usr/src/app/node_modules/mysql2/lib/commands/comma
shira-api-dev |     at PoolConnection.handlePacket (/usr/src/app/node_modules/mysql2/lib/
shira-api-dev |     at PacketParser.onPacket (/usr/src/app/node_modules/mysql2/lib/conne
shira-api-dev |     at PacketParser.executeStart (/usr/src/app/node_modules/mysql2/lib/pa
shira-api-dev |     at Socket.<anonymous> (/usr/src/app/node_modules/mysql2/lib/connecti
shira-api-dev |     at Socket.emit (node:events:526:28)
shira-api-dev |     at addChunk (node:internal/streams/readable:315:12)
```

Impact Analysis

In a non-monolithic server deployment this is unlikely to be an exploitable vulnerability, even if the server process crashes and hangs. However, the response may reveal that the exception has been triggered. This may aid in some other attacks or assist an adversary with understanding the application structure and configuration state.

Remediation Recommendations

Catch exceptions to handle errors gracefully, with error logs / stack traces directed to log streams.

Appendix

Methodology

Our approach to testing is designed to understand the design, behavior, and security considerations of the assets being tested. This helps us to achieve the best coverage over the duration of the test.

To accomplish this, Subgraph employs automated, manual and custom testing methods. We conduct our automated tests using the industry standard security tools. This may include using multiple tools to test for the same types of issues. We perform manual tests in cases where the automated tools are not adequate or reveal behavior that must be tested manually. Where required, we also develop custom tools to perform tests or reproduce test findings.

The goals of our testing methodology are to:

- Understand the expected behavior and business logic of the assets being tested
- Map out the attack surface
- Understand how authentication, authorization, and other security controls are implemented
- Test for flaws in the security controls based on our understanding
- Test every point of input against a large number of variables and observe the resulting behavior
- Reproduce and re-test findings
- Gather enough supporting information about findings to enable us to classify, report, and suggest remediations

Description of testing activities

Depending on the type and scope of the engagement, our methodology may include any of the following testing activities:

1. **Information Gathering:** Information will be gathered from publicly availble sources to help increase the success of attacks or discover new vulnerabilities
2. **Network discovery:** The networks in scope will be scanned for active, reachable hosts that could be vulnerable to compromise
3. **Host Vulnerability Assessment:** Hosts applications and services will be assessed for known or possible vulnerabilities
4. **Application Exploration:** The application will be explored using manual and automated methods to better understand the attack surface and expected behavior
5. **Session Management:** Session management in web applications will be tested for security flaws that may allow unauthorized access
6. **Authentication System Review:** The authentication system will be reviewed to determine if it can be bypassed
7. **Privilege Escalation:** Privilege escalation checks will be performed to determine if it is possible for an authenticated user to gain access to the privileges assigned to another role or administrator

8. **Input Validation:** Input validation tests will be performed on all endpoints and fields within scope, including tests for injection vulnerabilities (SQL injection, cross-site scripting, command injection, etc.)
9. **Business Logic Review:** Business logic will be reviewed, including attempts to subvert the intended design to cause unexpected behavior or bypass security controls

Reporting

Findings reports are peer-reviewed within Subgraph to produce the highest quality findings. The report includes an itemized list of findings, classified by their severity and remediation status.

Severity ratings

Severity ratings are a metric to help organizations prioritize security findings. The severity ratings we provide are simple by design so that at a high-level they can be understood by different audiences. In lieu of a complex rating system, we quantify the various factors and considerations in the body of the security findings. For example, if there are mitigating factors that would reduce the severity of a vulnerability, the finding will include a description of those mitigations and our reasoning for adjusting the rating.

At an organization's request, we will also provide third-party ratings and classifications. For example, we can analyze the findings to produce *Common Vulnerability Scoring System* (CVSS)[1] scores or *OWASP Top 10*[2] classifications.

The following is a list of the severity ratings we use with some example impacts:

Critical

Exploitation could compromise hosts or highly sensitive information

Critical Exploitation could compromise hosts or highly sensitive information

High

Exploitation could compromise the application or moderately sensitive information

High Exploitation could compromise the application or moderately sensitive information

Medium

Exploitation compromises multiple security properties (confidentiality, integrity, or availability)

Medium Exploitation compromises multiple security properties (confidentiality, integrity, or availability)

[1]https://www.first.org/cvss/
[2]https://www.owasp.org/index.php/Category:OWASP_Top_Ten_Project

Low

Exploitation compromises a single security property (confidentiality, integrity, or availability)

Low Exploitation compromises a single security property (confidentiality, integrity, or availability)

Info

Finding does not directly pose a security risk but merits further investigation

Info Finding does not directly pose a security risk but merits further investigation

The severity of a finding is often a product of the impact to general security properties of an application, host, network, or other information system.

The properties that can be impacted are:

Confidentiality Exploitation results in authorized access to data

Integrity Exploitation results in the unauthorized modification of data or state

Availability Exploitation results in a degradation of performance or an inability to access resources

The actual severity of a finding may be higher or lower depending on a number of other factors that may mitigate or exacerbate it. These include the context of the finding in relation to the organization as well as the likelihood of exploitation. These are described in further detail below.

Contextual factors

Confidentiality, integrity, and availability are one dimension of the potential risk of a security finding. In some cases, we must also consider contextual factors that are unique to the organization and the assets tested.

The following is a list of those factors:

Financial Exploitation may result in financial losses

Reputation Exploitation may result in damage to the reputation of the organization

Regulatory Exploitation may expose the organization to regulatory liability (e.g. make them non-compliant)

Organizational Exploitation may disrupt the operations of the organization

Likelihood

Likelihood measures how probable it is that an attacker exploit a finding.

This is determined by numerous factors, the most influential of which are listed below:

Authentication Whether or not the attack must be authenticated

Privileges Whether or not an authenticated attacker requires special privileges

Public exploit Whether or not exploit code is publicly available

Public knowledge Whether or not the finding is publicly known

Exploit complexity How complex it is for a skilled attacker to exploit the finding

Local vs. remote Whether or not the finding is exposed to the network

Accessibility Whether or not the affected asset is exposed on the public Internet

Discoverability How easy it is for the finding to be discovered by an attacker

Dependencies Whether or not exploitation is dependant on other findings such as information leaks

Remediation status

As part of our reporting, remediation recommendations are provided to the client. To help track the issues, we also provide a remediation status rating in the findings report.

In some cases, the organization may be confident to remediate the issue and test it internally. In other cases, Subgraph works with the organization to re-test the findings, resulting in a subsequent report reflecting remediation status updates.

If requested to re-test findings, we determine the remediation status based on our ability to reproduce the finding. This is based on our understanding of the finding and our awareness of potential variants at that time. To reproduce the results, the re-test environment should be as close to the original test environment as possible.

Security findings are often due to unexpected or unanticipated behavior that is not always understood by the testers or the developers. Therefore, it is possible that a finding or variations of the finding may still be present even if it is not reproducible during a re-test. While we will do our best to work with the organization to avoid this, it is still possible.

The findings report includes the following remediation status information:

Resolved
Finding is believed to be remediated, we can no longer reproduce it

Resolved Finding is believed to be remediation, we can no longer reproduce it

In progress
Finding is in the process of being remediated

In progress Finding is in the process of being remediated

Unresolved
Finding is unresolved – used in initial report or when the organization chooses not to resolve

Unresolved Finding is unresolved – used in initial report or when the organization chooses not to resolve

Not applicable
There is nothing to resolve, this may be the case with informational findings