BUG BOUNTY
HUB

Community Pentest Results

for Republik

November 2021

TABLE OF CONTENTS

General summary

A security audit targeting several republik.ch applications has been ordered with an evaluation of the general level of security via an intrusion approach from the Internet without user accounts provided and with access to the source code (open source) of the applications.

- Analysis of the applications in black box and grey box (via a classic user account)
- Source code analysis
- Automated and mainly manual tests from 06/11/2021 to 07/11/2021

We consider the risk level of the audited platform as low

- 0 critical vulnerabilities
- 0 high vulnerabilities
- 1 moderate vulnerabilities
- 1 low vulnerabilities

We consider the correction effort as low

- Only one vulnerability and one weakness need to be fixed
- Some hardening can still be done

Summary of good practices observed :

- Good management of CORS which is quite rare
- A password less connection system which is also rare and therefore prevents account theft
- The application has many checks and filters that are well done

---

The following elements considered as "critical" have been the object of particular attention :

- Connection system
- Partitioning between accounts (horizontal and vertical)
- Subscription system

No vulnerabilities or weaknesses could be detected on these endpoints

---

Important note : The GraphQL which is at the center of the application represents a critical brick, this one is quite complex and could require more time to understand it well and try to discover vulnerabilities that we could not find during these two days

Detailed report

Analysis methodology

The methodology of analysis took place in black box by collecting as much information as possible from the outside, then in gray box with the creation of a user account and also in white box with the analysis of the source code of applications and the APK.

The black box / grey box approach focuses mainly on the application aspects of the target with the following elements:

- Network mapping
- Identification of available services, software and versions
- Analysis of potential vulnerabilities on these services
- Analysis of the global configuration of the application servers
- Automated and especially manual vulnerability research
- Analysis of authentication, session management and partitioning best practices

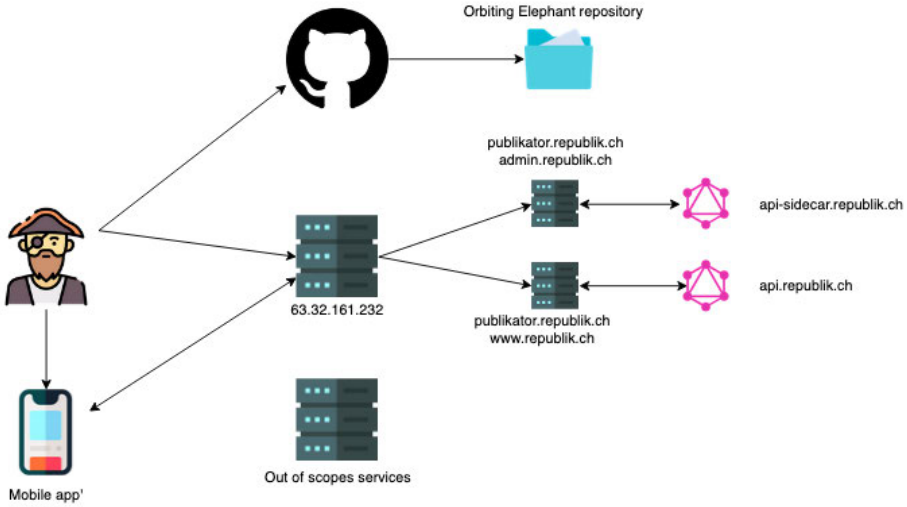
Harvesting / Discovery phase

- Collection of subdomains
- Leak collection
- URL collection

Note : This preliminary step is totally automated on our side, it allows us to obtain initial information on the target which will be useful for me for the next step which is manual, the analysis of these data.

- 1228 reachable URLs (excluding 404) are referenced for https://www.republik.ch/
- 3 reachable URLs (excluding 404) are referenced for https://admin.republik.ch/
- 2 reachable URLs (excluding 404) are referenced for https://publikator.republik.ch/
- 7 reachable URLs (excluding 404) are referenced for https://api.republik.ch/
- 6 leaks of accounts containing but without a clear text password
- 2 Github potential (unconfirmed) logins with in plaintext

Vision of the services from our point of view :


Orbiting Elephant repository
publikator.republik.ch
admin.republik.ch
api-sidecar.republik.ch
api.republik.ch
63.32.161.232
publikator.republik.ch
www.republik.ch
Out of scopes services
Mobile app'

Identified domains

republik.love
republik.ch
matomo.republik.ch
publikator.republik.ch
assets.api.republik.ch
mail.republik.ch
pop.republik.ch
admin.republik.ch
autoconfig.republik.ch
assets.republik.ch
ultradashboard.republik.ch
assets-api2.republik.ch
styleguide.republik.ch
cdn.republik.ch
smtp.republik.ch
lists.republik.ch
imap.republik.ch
api.publikator.republik.ch
api.republik.ch
sandstorm.republik.ch
assets.staging.republik.ch
mail2.republik.ch
assets-api.republik.ch
api-sidecar.republik.ch
mandrill.republik.ch
ux.republik.ch
autodiscover.republik.ch
assets.publikator.republik.ch
wiki.republik.ch
wwww.republik.ch
www.republik.ch
api-5027121202214205556-160060.firebaseio.com
api-5027121202214205556-160060.appspot.com
cdn.repub.ch

Accessible republik.ch subdomains :

https://api-sidecar.republik.ch:443
https://admin.republik.ch:443
https://api.republik.ch:443
https://api.publikator.republik.ch:443
https://assets.publikator.republik.ch:443
http://assets.publikator.republik.ch:80
https://assets.republik.ch:443
http://admin.republik.ch:80
http://api-sidecar.republik.ch:80
http://api.publikator.republik.ch:80
http://api.republik.ch:80
http://assets.republik.ch:80
http://assets.staging.republik.ch:80
http://assets.staging.republik.ch:443
https://autodiscover.republik.ch:443
http://autodiscover.republik.ch:80
https://autoconfig.republik.ch:443
http://autoconfig.republik.ch:80
http://cdn.republik.ch:443
https://lists.republik.ch:443
http://lists.republik.ch:80
https://republik.ch:443
https://publikator.republik.ch:443
http://publikator.republik.ch:80
http://republik.ch:80
http://styleguide.republik.ch:80
https://styleguide.republik.ch:443
https://ux.republik.ch:443
https://ultradashboard.republik.ch:443
https://wwww.republik.ch:443
https://www.republik.ch:443
http://ultradashboard.republik.ch:80
http://ux.republik.ch:80
http://www.republik.ch:80
http://wwww.republik.ch:80

SSL/TLS Analyse & Headers analyse

The domains point to the same server and use similar certificates (Let's encrypt) and a similar configuration.Overall the configuration is good and allows to have a grade of B on SSLabs, the configuration can however be improved with :

- Disabling cipher suites deemed weak for TLS1.2
- The implementation of the Forward Secrecy

For the headers, there are small improvements that differ according to the domain

- CSP (Content Security Policy is an effective measure to protect your site from XSS attacks) are not activated on www.republik.ch & publikator.republik.ch

- Permissions Policy (a new header that allows a site to control which features and APIs can be used in the browser) are not activated on admin.republik.ch & publikator.republik.ch & api.republik.ch

The configuration is therefore considered to be robust overall, the activation of the CSPs on the main domain will still allow to apply a useful additional hardening

The targets always use a connect.sid cookie with the same configuration.Although the HttpOnly (measure preventing access to the cookie via JavaScript and therefore XSS) and Secure (this flag means that the browser only sends the cookie on the means of communication it considers secure, i.e. https) flags, the SameSite(let servers require that a cookie not be sent with cross-site requests) flag could also be set, it is a quick win that mitigates CSRF (Cross-Site Request Forgery).

Leaks

Email	Username	Hashed password	Plain text	Comment
[redacted]		[redacted]	/	[redacted]
[redacted]		[redacted]	/	[redacted]
[redacted]		[redacted]	/	[redacted]
[redacted]		[redacted]	/	[redacted]
[redacted]	/	[redacted]	/	[redacted]

[redacted]	[redacted]	[redacted]	/	[redacted]

Analysis of the APK

A static analysis of the APK was performed.This step consists in searching for secrets and other interesting information in the APK code and also in observing the different services, activities, recivers and providers in search of dangerous functions that can be manually exploited.

No juicy information could be detected, we note however these two API keys detected in the code (they are however public and therefore legitimate) :

Usage	Key	Comment
[redacted]	[redacted]	[redacted]
[redacted]	[redacted]	[redacted]

Source code analyse

Reasons for doing code review :

- Because some issues like weak encryption or building a SQL query in an insecure manner can be detected more quickly
- Have a better vision of the application before the offensive approach
- If a bug is detected, it is possible to track it in the code to possibly increase its impact due to a better understanding

Possibles methodologies:

- String matching/Grep for bugs
  - Static analysis to refine the code
- Following user input
  - Perfect when you detect a suspected bug to understand how to exploit it
- Reading source code randomly / Read all the code
  - Not what I do/promote
- Check one functionality at a time
  - Recommended on critical features like login, password resets, payment features, ...

Dependency analysis

This step allows to analyze the different integrated packages. It is quite common that a package has at least 1 vulnerability, this does not mean that it is exploitable in the project that uses the package.

Mainly because in some cases the vulnerability is only exploitable if a call to a function of the package is callable.

The following summary shows the vulnerabilities that would be potentially exploitable/interesting for an attacker

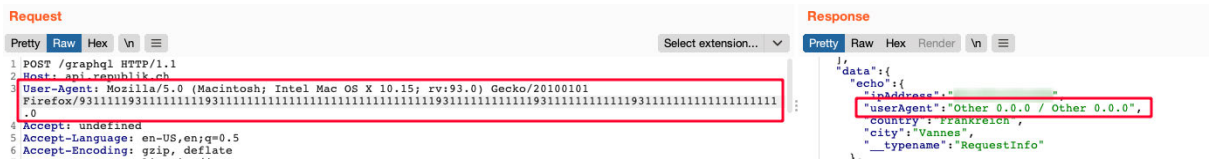
Name	Critical	High	Medium	Low
publikator-frontend	1	16	20	1
orbiting/ backends	1	15	18	1
admin-frontend	0	15	11	0
republik-frontend	0	2	6	0

Most of these vulnerabilities are Denial of Service via regex exploitation or prototype pollution.

- Example 1 : Regular Expression Denial of Service (ReDoS) affecting ua-parser-js => https://security.snyk.io/vuln/SNYK-JS-UAPARSERJS-610226
- Example 2 : Affected versions of this lodash package are vulnerable to Prototype Pollution in zipObjectDeep => https://security.snyk.io/vuln/SNYK-JS-LODASH-590103

Remediation: To remedy these potentially exploitable weaknesses, as the project is open source, it is possible to use Snyk.io to monitor the various repositories

We emphasize that just because a package has a vulnerability does not mean it is exploitable. In the case of the ReDOS via the Regex in the User-Agent, it is not exploitable because after a certain number of characters, instead of displaying "other" in response.



Search for hardcoded secrets

This step consists in searching for secrets in the Github repository using regex by analyzing all the commits

- Default Elastic password - https://github.com/orbiting/backends/blob/09834a0a19ce5efb98dfd6747d2bb873d67fa189/docker-compose.yml
- user / some-password | The original may have been replaced but just in case Github - https://github.com/orbiting/backends/blob/5742e7cd629b24de8074924eae165f0cf3db673f/packages/mailbox/script/backup.js

code analysis

In order not to indicate too many false positives, we indicate only the points which seem to us of interest, which requires a hardening or which could be exploited

Some points are also not indicated because we checked manually and considered unexploitable like the following example :

- Inside republik-frontend/components/Footer/index.js line 179, the username of the user if filled in is inserted in the footer without any particular filtering, which could lead to an XSS. Although this code seems vulnerable, it is not exploitable, because the addition of a username prevents the use of characters other than [a-zA-Z0-9\.] and if by some maner means we manage to insert an unwanted character it is replaced by a .. Moreover, at this place, the injection of HTML tags would naturally have been filtered by the framework

```
179                    <FooterNavLink href={`/~${me.username || me.id}`}>
180                      <a {...navLinkStyle}>{t('footer/me/profile')}</a>
181                    </FooterNavLink>
```

orbiting/backends :

- [LOW] Incomplete URL substring sanitization
  - File : packages/base/lib/Redis.js line 12
  - Description :'amazonaws.com' can be anywhere in the URL, and arbitrary hosts may come before or after it.
  - Remediation : Sanitizing untrusted URLs. This can be done by checking that the host of a URL is in a set of allowed hosts.

```
packages/base/lib/Redis.js:12
 9    const { REDIS_URL = 'redis://127.0.0.1:6379' } = process.env
10
11    const url = new URL(REDIS_URL)
12    const isHerokuRedis = url.hostname.indexOf('amazonaws.com') > -1
```

- [LOW] Use of a broken or weak cryptographic algorithm
  - File :
    - packages/republik-crowdfundings/lib/payments/postfinance/payPledge.js line 30
    - packages/republik-crowdfundings/lib/payments/postfinance/sha.js line 66
  - Description : SHA1 is now considered insecure because several collisions have been detected on this algorithm by Google
  - Remediation : Use SHA3 for example or something more robust like SHA256
  - Notes : Other uses of SHA-1 or MD5 were observed in the code, these implementations are not necessarily a problem that is why we did not indicate all files. For example for the storage of the profile image, an MD5 is calculated, although it would be better to have a more robust hash, the use of MD5 in this case is not considered unsafe

packages/republik-crowdfundings/lib/payments/postfinance/**sha.js**:66

```
63
64    return crypto
65      .createHash('sha1')
66      .update(paramsString)
```

- [HIGH] Possible Header Injection: Improper Neutralization of HTTP Headers for Scripting Syntax
  - File : packages/assets/express/render.js line 59
  - Description : Untrusted user input in response header will result in HTTP Header Injection or Response Splitting Attacks.
  - Remediation : /

```
57        if (cacheResult.ok) {
58          ;['Content-Type', 'Content-Length'].forEach((key) => {
59            res.set(key, cacheResult.headers.get(key))
60          })
```

orbiting/republik-frontend :

- [MED] XSS: HTTP data used in render function without sanitization
  - File : components/Profile/Contact.js line 178
  - Description : User controlled data is used in rendering href or src attribute without sanitization. Automatic sanitization offered by React is not quite effective in such cases.
  - Remediation : Sanitize and validate HTTP data before passing it back to the user via the render function.
  - Notes : Vulnerability exploited and detailed in : "*XSS-1 - Stored XSS in the user profile*"

```
175           <IconButton
176             Icon={LanguageIcon}
177             style={{ marginRight: electionBallot && 0 }}
178             href={user.publicUrl}
179           />
```

orbiting/republik-admin-frontend : Nothing interesting

orbiting/publikator-frontend :

- [LOW] Inefficient regular expression
  - File : components/editor/modules/embed/index.js line 213
  - Description : This part of the regular expression may cause exponential backtracking on strings starting with http://youtu.be/& and containing many repetitions of &. Such regular expressions can negatively affect performance, or even allow a malicious user to perform a Denial of Service ("DoS") attack by crafting an expensive input string for the regular expression to match.
  - Remediation : /

components/editor/modules/embed/**index.js**:213

```
210  const TWITTER_REGEX = /^https?:\/\/twitter\.com\/(?:#!\/)?\w+\/status(?:es)?\/(\d+)$/
211
212  // One capturing group at match[1] that catches the video id
213  const YOUTUBE_REGEX = /^http(?:s?):\/\/(?:www\.)?youtu(?:be\.com\/watch\?v=|\.be\/)([\w\-_]*)(&[^\s]*)*$/
```

Useful resources

Useful resources for the technical elements that may have been discussed in this report :

- Understand XSS
- Understand CSRF
- Understand NoSQL Injection
- Understand Prototype Pollution
- Cookie Security

Vulnerabilities

REP-1 - Over permissive Employee/User Object

With the GraphQL console available along with queries that a "member" level user has access to it is possible to query the employees and user objects to return employee data using the below query, we can additionally access some additional properties in the user Object but to keep the report clean we have opted some of the more interesting properties

Query:

```
query getEmp($suffle: Int=100) {
  employees(shuffle : $suffle ) {
    name
    greeting
     user {
      id
      slug
      username
      email
      credentials {
           description
      }
      isSuspended
    }
  }
}
```

Reproduction steps:

1. Navigate to https://api.republik.ch/graphiql
2. Copy and paste the above query into the console and press the play button

Sample dataset:


```
{
  "data": {
    "employees": [
      {
        "name": "Katharina Hemmer",
        "greeting": null,
        "user": {
          "id": [redacted],
          "slug": "khemmer",
          "username": "khemmer",
          "email": null,
          "credentials": [
            {
              "description": "Community+"
            }
          ],
          "isSuspended": null
```

```
    }
  },
  {
    "name": "Laurent Burst",
    "greeting": null,
    "user": {
      "id": [redacted],
      "slug": "laurentburst",
      "username": "laurentburst",
      "email": [redacted]
      "credentials": [],
      "isSuspended": null
    }
  },
```


Resolution : Limit access to sensitive objects and/or properties. You already do this for such properties as adminNotes.

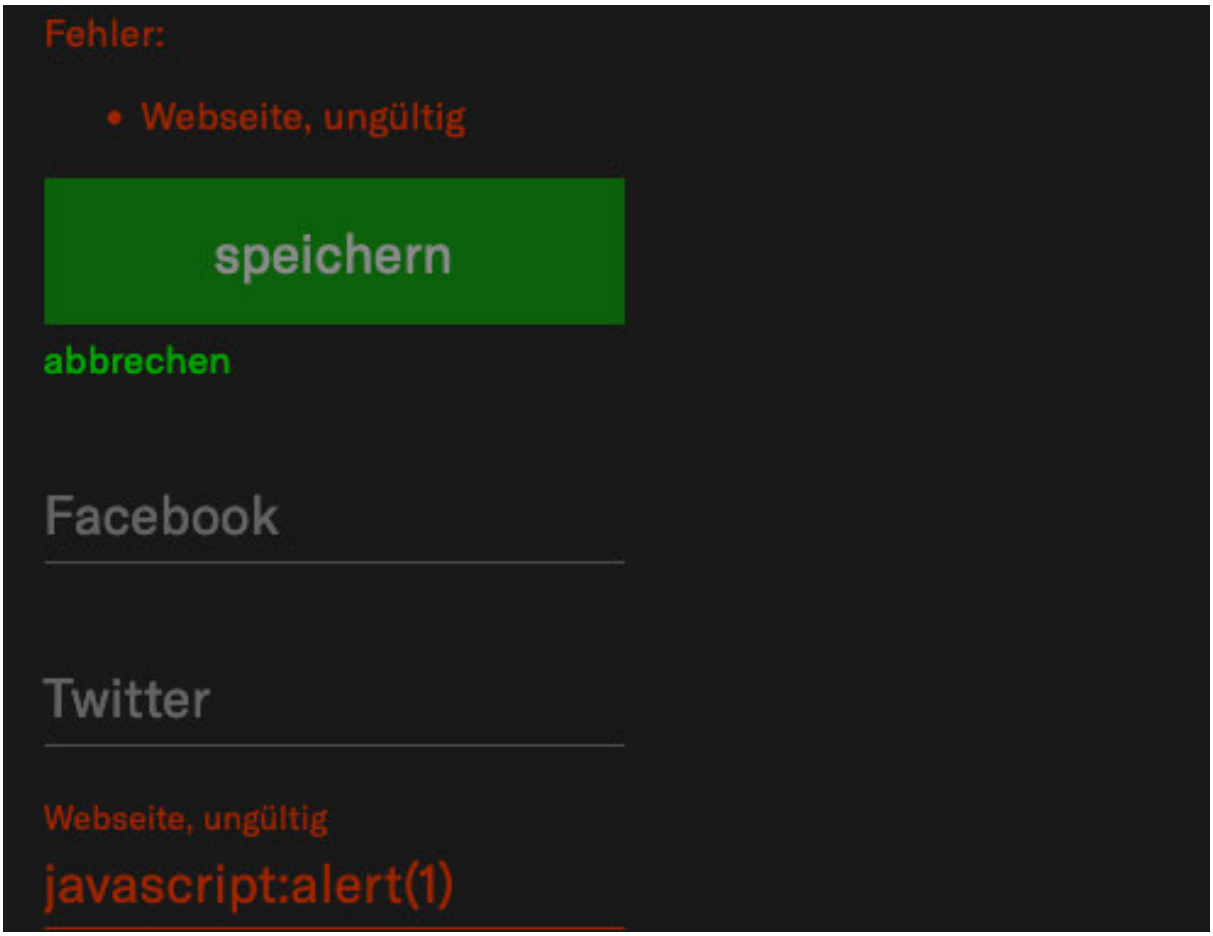
Restrict the access to the employee object to staff only. In particular for the email address which is considered as "sensitive" according to the GDPR.

XSS-1 - Stored XSS in the user profile

Description : The user profile allows you to add the URL of your website, unfortunately this entry lacks filtering and allows you to insert an arbitrary URL leading to a stored XSS

Cross-Site Scripting (XSS) attacks are a type of injection, in which malicious scripts are injected into otherwise benign and trusted websites. XSS attacks occur when an attacker uses a web application to send malicious code, generally in the form of a browser side script, to a different end user.

1. On the client side, a filtering is in place and checks that the user input looks like a URL :

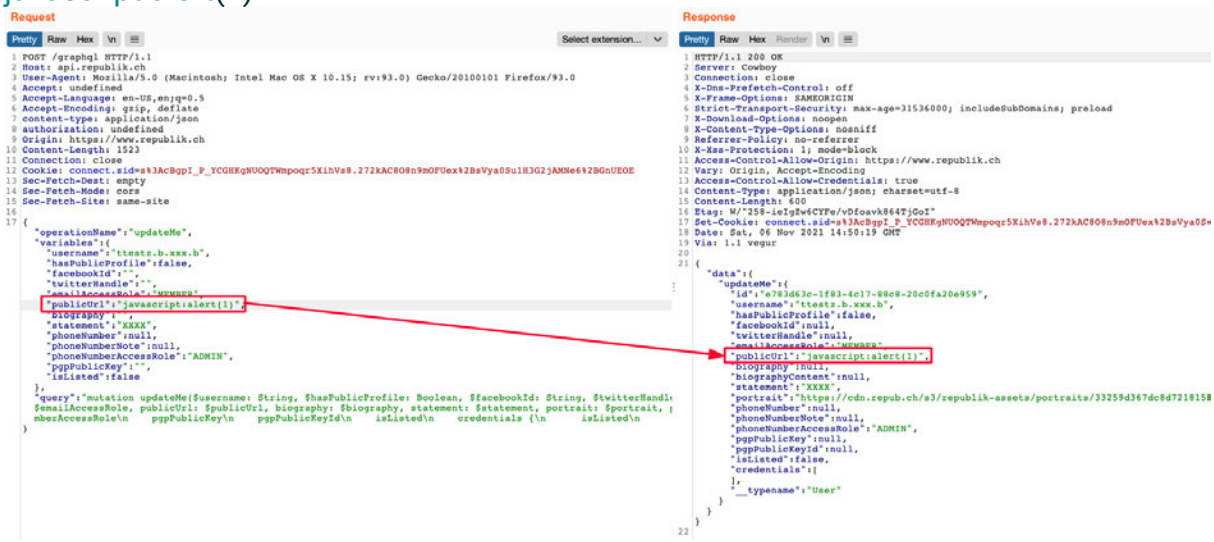


2. But by intercepting the request with BurpSuite, it is possible to modify the request on the fly and modify a valid url such as

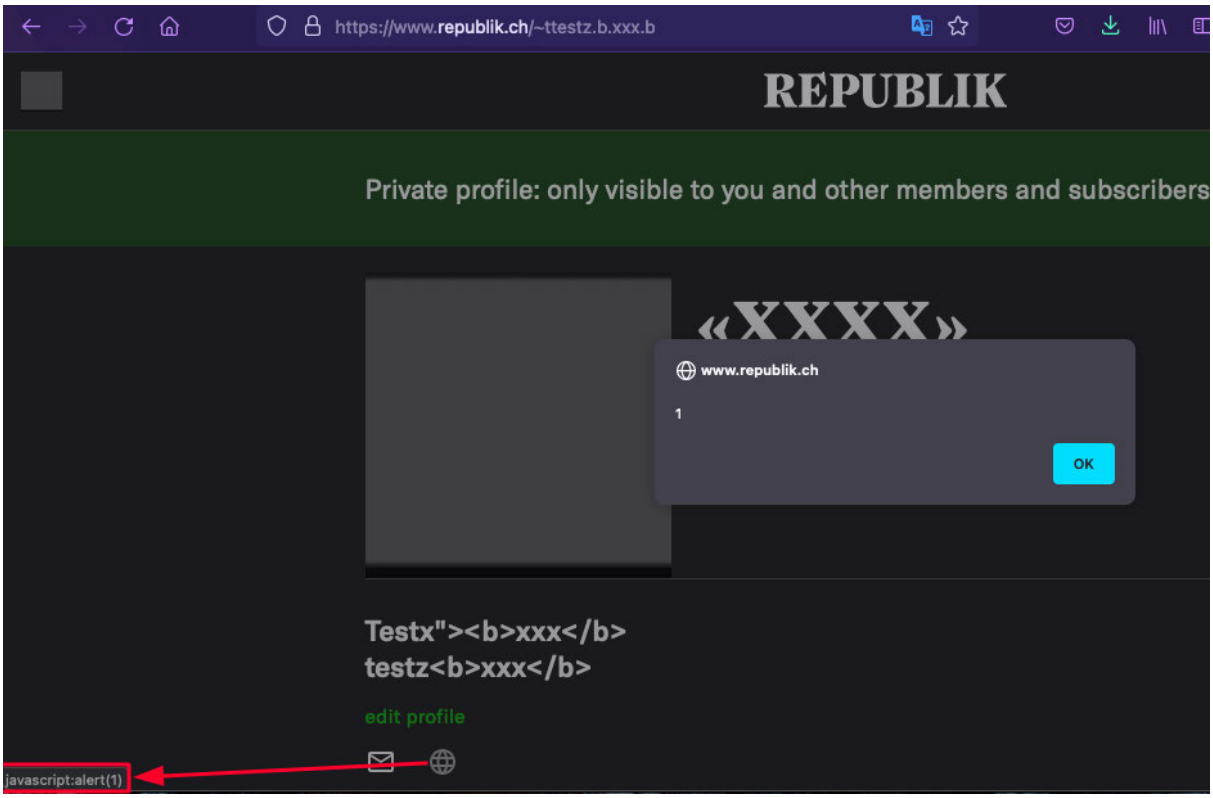
   https://www.google.com

   by

   javascript:alert(1)



3. By clicking on the website icon on the user profile, the JS code is executed



By making its profile public, this stored XSS (which still requires user interaction) can therefore impact several people

Resolution : Filter the user input on the server side as well as on the client side by checking that the URL starts with /https?:\/\//**Resources** : Cross Site Scripting Prevention Cheat Sheet

---