Fine penetration tests for fine websites

Dr.-Ing. Mario Heiderich, Cure53
Bielefelder Str. 14
D 10709 Berlin
cure53.de · mario@cure53.de

Pentest-Report GovTech FormSG Web & API 07.2020

Cure53, Dr.-Ing. M. Heiderich, BSc. C. Kean, B. Walny, Dr. N. Kobeissi

Index

Introduction

This report documents the findings of a security assessment of the FormSG web application, including the related SDK and its implemented E2E cryptography. The work was requested by the Government Technology Agency Singapore, also referred to as GovTech, and was executed by Cure53 in July of 2020, precisely in calendar week 30. Cure53 specifically carried out a penetration test and source code audit paired with a cryptography review.

The tests and audit involved a team of five Cure53 team members, each of them a specialist in their respective field of expertise. The testing team examined the entire range of components in scope within an allocated time and budget of twelve days. The test methodology for this project was chosen to be white-box; the auditors were given unencumbered access to the application source code and were able to test against a deployment on a UAT server made available by GovTech. The static IPs used by Cure53 during this assignment were communicated in advance of the actual commencement of the testing activities and allow-listed by the responsible entity.

Dr.-Ing. Mario Heiderich, Cure53
Bielefelder Str. 14
D 10709 Berlin
cure53.de · mario@cure53.de

To best address the scope and coverage in accordance with what has been requested by GovTech, the actual work was split into two distinct and complementary work packages (WPs). The initial WP1 encompassed white-box testing of the provided deployment and a simultaneous audit of the respective source code, consisting of FormSG, its SDK and webhooks. The secondary WP2 was solely focused on reviewing the cryptography employed in the FormSG E2E encryption components. The latter was also done by a dedicated specialist.

The project started on time and progressed efficiently. All communications during this security assessment were done via a dedicated Slack channel provided by GovTech and into which relevant personnel from Cure53 were invited. A secondary channel was made available as well, but predominantly used for the preparation work, which took place ahead of the actual testing activities. Technical conversations were polite and efficient, Cure53 was able to ask questions, get the respective answers quickly and was able to share status updates about the progression of the assignment.

Given the chosen white-box methodology, with unobstructed access to source code and deployed server application, and considering the overall good preparation and supporting materials, Cure53 managed to cover a very wide range of subjects and targets, while simultaneously achieving quite significant depth of research. All this contributed to the very positive penetration test and source code audit results, with a total of four findings, of which two were later re-evaluated as false positives. The remaining issues were classified as general weaknesses of *Low* and informational severity and, by nature, held limited exploitation potential. This is a very good result, especially in connection to the rather large and complex attack surface of the application compound.

In the following sections, the report will first shed some light on the scope of this assessment and the respective test setup, moving on to elaborate on the actual test coverage, test methodology and work packages used to structure this July 2020 exercise. Cure53 precisely documents what exactly was checked, despite the unusually small number of findings in several areas. Subsequently, all valid findings, false positives or not, are discussed in chronological order, alongside their technical descriptions, optional Proof-of-Concept (PoC) displays and mitigation advice, whenever applicable. Finally, the report will close with broader conclusions, subsuming the testing team’s impressions on security and privacy of the software complex while further expanding on some of the more abstract insights gained during these passed penetration tests and source code audits. Cure53 offers additional high-level advice on possible improvements and continued hardening where appropriate.

Dr.-Ing. Mario Heiderich, Cure53
Bielefelder Str. 14
D 10709 Berlin
cure53.de · mario@cure53.de

Scope

- **White-Box Security Tests & Crypto Audits against FormSG Web App & SDK**
  - **WP1**: White-Box Security Tests against FormSG, SDK and Webhooks
    - http://uat.form.gov.sg/
  - **WP2**: Crypto Review against FormSG E2E Encryption Components
  - **Additional Info:**
    - FormSG user guide:
      - https://guide.form.gov.sg
  - **FormSG Sources were shared with Cure53**
  - **Additional Material was shared with Cure53**
  - **Whitelisted Cure53 IPs**
    - 188.165.115.83
    - 119.17.157.228
    - 82.102.25.226
    - 185.173.226.49
    - 81.17.246.108
    - 192.145.124.238
    - 206.189.218.238

Dr.-Ing. Mario Heiderich, Cure53
Bielefelder Str. 14
D 10709 Berlin
cure53.de · mario@cure53.de

Test Methodology

The following section documents the testing methodology applied during this engagement and sheds light on the various areas of the web application subject to inspection and audit. It further clarifies which areas were examined by Cure53 but did not yield any findings.

WP1: White-Box Security Tests against FormSG, SDK and Webhooks

The information below describes the tests and coverage achieved for the web security-related testing of the given FormSG scope. The section comments on which areas were investigated by utilizing the enumerated approaches.

- During the source code review and white-box penetration testing of the given web platform, it was made sure that all common web security practices are followed.
- Additional fuzzing and input-validation testing, along with verification of the appropriate parts of the source code, was done to achieve maximum possible coverage.
- Starting with general checks on the web server, in particular of all the expected security headers, it was noticed that a certain selection was missing. Although this is not a serious issue, it is recommended to make sure the web server responds with the advised configurations. Details can be found in the appropriate ticket at GTA-01-004.
- Although one of the set cookies was scoped to the parent domain, Cure53 was not able to identify any related risks. Session cookies correctly set all recommended flags and even included the *samesite* attribute to prevent CSRF attacks.
- During request parsing checks, it was additionally ensured that recent attack types - such as *request smuggling* - are prevented. Cure53 ran all checks for *TE.CL* and *CL.TE* vectors without success, thus proving that potentially disjunct frontend and backend software agree on the same transfer encoding types.
- Continuing with checks against all currently defined routes and making sure that all request paths within the web application are correctly handled, it was noticed that routing is terminally defined. Static routes via insecure filesystem operations were absent and no accidentally left-behind or still reachable artifacts could be spotted. All other parts of the routing follow recommended formats with strict limitations on what comprises valid request paths.
- Verifying that login functionality works as expected, it was realized that OTP generation is secure and expiration times are consistently checked. Hashed storage of tokens was found to be up to current standards.

Dr.-Ing. Mario Heiderich, Cure53
Bielefelder Str. 14
D 10709 Berlin
cure53.de · mario@cure53.de

- OTP challenges were found to be invincible to brute-force attacks, as the maximum number of attempts is limited and the concurrent generation of multiple valid codes is prohibited.
- Additional testing of login and all other methods receiving user-input (also for shared forms) was completed for injection-type attacks. Since *Mongoose* is the NoSQL solution of choice, it was ensured that all delivered request variables follow the expected content type.
- The audit proceeded to white-box testing of dangerous NodeJS sinks, e.g. *fs.read-*, *writeFile-calls* and similar pitfalls like *child_process* invocations, resulting in the discovery of a limited SSRF vulnerability, which is described in GTA-01-002. While the problem's impact is rather limited, Cure53 still feels this is a flaw which is worth-resolving to limit possible consequences.
- Deep checks were done against all attachment and upload features, in particular for form submissions. File handling and checks for potential directory traversal vulnerabilities and symlink attacks against *zip* uploads (as this is an explicitly allowed file format) were executed in vain. The integration with AWS was evaluated and is considered flawless.
- A similarly good impression (apart from one issue) was left by the webhooks functionality. All checks are deemed clean and it was verified that the chosen *Axios* client library is specifically configured to not follow HTTP redirects. Hostnames are consistently resolved and checked against private address spaces, even though it may be possible that this functionality is vulnerable to DNS rebinding. Judging by the coding style, it may very well be possible that the timeframe between the host validation and the actual *POST* request offers a race window, as described in GTA-01-003. Such impact would most likely be quite limited and highly dependent on attackers being able to conduct extensive network reconnaissance before being able to successfully attempt exploitation.
- The FormSG web application and API made a robust impression with regard to client-side vulnerabilities. Not a single XSS vulnerability could be identified, which should be attributed to the developer's correct use of the AngularJS framework.
- All access controls for creating, updating and deleting items across the FormSG application were found to be secure and all tested endpoints correctly verify the current user's permissions before granting access.

Dr.-Ing. Mario Heiderich, Cure53
Bielefelder Str. 14
D 10709 Berlin
cure53.de · mario@cure53.de

WP2: Crypto Review against FormSG E2E Encryption Components

A list of items below seeks to detail the tasks completed during the cryptography-centered portion of the security testing phase of this project. This is to underline what the Cure53 testers covered during their analysis, particularly in regard to mobile application security within FormSG.

- FormSG's form encryption primitives were traced back to their original implementation (TweetNaCl) and reviewed for proper and secure usage.
- Care was taken to assess whether forgery of submissions was possible, as well as in terms of additional information gleaned based on their cipher-text (aside from file-size and information about the submitter.)
- *"Encoding for encrypted data"* was verified for correctness and abuse potential; specifically instances constructing encodings were checked for the possibility to create colliding or misleading encodings.
- The generation and communication of public keys using X25519 between servers and clients was checked. The standard submission methodology, which employs the local secret signing key for authentication and a communicated server public key for encryption, was examined in order to verify against scenarios where clients could be tricked into encrypting payloads incorrectly. A critical assumption in this regard is that the server is not compromised and that TLS is employed on all secure channels.
- The usage of TweetNaCl rules out issues that plague other cryptographic APIs, such as nonce reuse/misuse and the usage of low-level cryptographic functions without message authentication or flawed Diffie-Hellman operations.
- Password management and hashing was checked for resistance against GPU-based attacks and other kinds of attempts to undermine the security gains from stretching user-provided passwords or key material.
- The controllers for email form encryption and submission were checked to ensure that encryption functions were bound correctly.

Dr.-Ing. Mario Heiderich, Cure53
Bielefelder Str. 14
D 10709 Berlin
cure53.de · mario@cure53.de

Identified Vulnerabilities

The following sections list both vulnerabilities and implementation issues spotted during the testing period. Note that findings are listed in chronological order rather than by their degree of severity and impact. The aforementioned severity rank is simply given in brackets following the title heading for each vulnerability. Every vulnerability is additionally given a unique identifier (e.g. *GTA-01-001*) for the purpose of facilitating any future follow-up correspondence.

GTA-01-001 WP2: Key derivation function vulnerable to certain vectors *(Low)*

Note: *This issue was evaluated as false-positive, since the mentioned functions are used for hash integrity checks and not for key stretching.*

It was found that FormSG uses *PBKDF2-HMAC-SHA512* in order to stretch encryption key information. On the one hand, the chosen hash function - namely *SHA512* - is relatively computation-expensive. On the other hand, 10.000 rounds of *PBKDF2* is below recommended minimums, and the *P2BKDF* approach in general is still vulnerable to a multitude of attacks including parallelization[1], optimization and GPU-based cracking[2].

Affected File:
dist/backend/app/controllers/email-submissions.server.controller.js

Affected Code:

```
function createHash(response, salt) {
    let saltLength = 32;
    salt = salt || crypto.randomBytes(saltLength).toString('base64');
    let iterations = 10000;
    let keylen = 64;
    let digest = 'sha512';
    return new Promise((resolve, reject) => {
        crypto.pbkdf2(response, salt, iterations, keylen, digest, (err, hash) =>
{
            if (err)
                reject(err);
            else {
                resolve({
                    hash: hash.toString('base64'),
                    salt,
                });
            }
        });
    });
```

[1] https://www.usenix.org/conference/woot16/workshop-program/presentation/ruddick
[2] https://link.springer.com/chapter/10.1007/978-3-319-26823-1_9

Dr.-Ing. Mario Heiderich, Cure53
Bielefelder Str. 14
D 10709 Berlin
cure53.de · mario@cure53.de

It is recommended to replace *PBKDF2* with a password-hashing function that is more resistant to these attacks, such as *scrypt*[3]. Unlike *PBKDF2, scrypt* depends on memory performance and is therefore more difficult to parallelize or crack by utilizing GPU computation. *scrypt* could be deployed with the parameters: $N = 2^{18}$, $r = 8$, $p = 1$.

Miscellaneous Issues

This section covers the noteworthy findings that did not lead to an exploit but might aid an attacker in achieving their malicious goals in the future. Most of these results are vulnerable code snippets that did not provide an easy way to be called. Conclusively, while a vulnerability is present, an exploit might not always be possible.

GTA-01-002 WP1: Blind SSRF via SNS signature verification *(Low)*

During the source code audit, a small input validation error resulting in a SSRF vulnerability via HTTP *GET* was discovered. It lies in the fact that the *SigningCertURL* URL is extracted from the request body without any allow-listing in place. This is visible in the following part of the application's source code.

Affected File:
formsg-master/src/app/utils/sns.js

Affected Code:

```
const isValidSnsSignature = async (body) => {
  const { data: cert } = await axios.get(body.SigningCertURL)
  const verifier = crypto.createVerify('RSA-SHA1')
  verifier.update(getSnsBasestring(body), 'utf-8')
  return verifier.verify(cert, body.Signature, 'base64')
}
```

The only limitation is the request URL being forced to HTTPS and having to end with "*.pem*". The file extension does not appear to be a problem though, since it can actually be moved to the request query. The following request demonstrates this against an attacker-controlled web server.

Example Request:

```
POST /emailnotifications HTTP/1.1
Host: uat.form.gov.sg
Content-Length: 204
Content-Type: application/json;charset=UTF-8
```

[3] https://www.tarsnap.com/scrypt.html

Dr.-Ing. Mario Heiderich, Cure53
Bielefelder Str. 14
D 10709 Berlin
cure53.de · mario@cure53.de

```
{"Message":"asd","MessageId":"asd","Timestamp":"asd","TopicArn":"asd","Type":"as
d","Signature":"asd","SigningCertURL":"https://apple.com.mmap.space/arbitrary-
rest-api?discard=.pem","SignatureVersion":"1"}
```

Observed server log entry:

```
13.250.48.167 - - [17/Jul/2020:17:03:50 +0200] "GET /arbitrary-rest-api?
discard=.pem HTTP/1.1" 404 178 "-" "axios/0.19.2"
```

It is consequently possible for attackers to abuse this vulnerability to send arbitrary *GET* requests to internal web servers not open to the public. Such an attempt requires additional information gathering and port scanning, however, thus only rating this issue miscellaneous. The risk does exist though and should still be treated as an input-validation problem. It is recommended to verify the *SigningCertURL* against an allow-list of URLs before sending the *GET* request.

Note: *This issue was fixed by the maintainers and the fix was reviewed by Cure53 in late July 2020. Cure53 had access to the PR and accepted the fix as working and valid.*

GTA-01-003 WP1: Webhooks potentially vulnerable to DNS rebinding *(Low)*

During the source code audit and verification of webhook functionality, an issue similar to GTA-01-002 was found. This time around it stems from the fact that the passed webhook URL may resolve to an unexpected resource after it has been verified. This is visible in the following excerpt of the application's source code.

Affected File:
formsg-master/src/app/controllers/webhooks.server.controller.js

Affected Code:

```
function post(req, _res, next) {
  const { form, submission } = req
  if (form.webhook.url) {
    const webhookUrl = form.webhook.url
[...]
    validateWebhookUrl(webhookParams.webhookUrl)
      .then(() => postWebhook(webhookParams))
```

While the code depicted above correctly verifies the passed webhook URL (in terms of having it only point to HTTPS URLs and prohibiting private IP ranges), the DNS server might actually resolve *webhookUrl* to another resource during *postWebhook*. This way of verification and posting leaves room for a race condition within which the DNS entry might get updated.

Dr.-Ing. Mario Heiderich, Cure53
Bielefelder Str. 14
D 10709 Berlin
cure53.de · mario@cure53.de

Because the attack is quite impractical and the timeframe rather small, Cure53 did not attempt to exploit it, but resolved to warn of the potential impact. The risk is similar to what was already described in GTA-01-002, meaning that this issue should be treated as another potential SSRF vulnerability. It is recommended to store the resolved IP address as the request target (along with the passed hostname) when initiating the webhook.

Note: *After discussing this issue with the client, it became apparent that a working fix in the current NodeJS implementation is not easily possible. Manually pinning the resolved IP to outgoing requests results in TLS certificate verification problems where the only solution appears to be to disable cert validation entirely.*

This, however, is an unacceptable trade-off because a hard to exploit SSRF is not reason enough to weaken the security posture of NodeJS' TLS configuration. Additionally, GovTech does not appear to host sensitive endpoints inside the internal network, thus generally lowering the impact of a potential SSRF through DNS Rebinding.

GTA-01-004 WP1: Inconsistent use of HTTP security headers *(Info)*

It was found that the FormSG platform is missing certain HTTP security headers in some HTTP responses. This does not directly lead to a security issue, yet it might aid attackers in their efforts to exploit other problems. The following list enumerates the headers that need to be reviewed to prevent this and similar flaws.

- **X-Frame-Options**: This header specifies whether the web page is allowed to be framed. Although this header is known to prevent Clickjacking attacks, there are many other attacks which can be achieved when a web page is frameable[4]. It is recommended to set the value to either SAMEORIGIN or DENY.
- Note that the CSP framework offers similar protection to X-Frame-Options in ways that overcome some of the shortcomings of the aforementioned header. To optimally protect users of older browsers and modern browsers at the same time, it is recommended to consider deploying the *Content-Security-Policy: frame-ancestors 'none';* header as well.

Missing security headers are generally bad practice and should be avoided. It is recommended to add the following headers to every server response, including error responses like *4xx* items. It is recommended to more broadly reiterate the importance of having all HTTP headers set at a specific, shared and central place rather than setting them randomly. This should either be handled by a load balancing server or similar

[4] https://cure53.de/xfo-clickjacking.pdf

Dr.-Ing. Mario Heiderich, Cure53
Bielefelder Str. 14
D 10709 Berlin
cure53.de · mario@cure53.de

infrastructure. If the latter is not possible, mitigation can be achieved by using the web server configuration and a matching module.

Note: *This issue was ultimately judged as a false-positive, since the omission of the XFO header is a business requirement.*

Conclusions

The FormSG web application, SDK and E2E cryptography implementation tested during this 2020 project made a very good impression, especially when taking a look at the actual tickets this penetration test and source code audit yielded. Since the only identified actual vulnerability eventually proved to be a false positive, and all other findings have been classified as being general weaknesses of maximum *Low* severity, the security standing of the software complex can only be described as solid. Having spent twelve days on the project in the summer of 2020, five members of the Cure53 reached a positive verdict about the security premise of the examined scope.

The observed unexpectedly good security posture can partly be attributed to the correct application of the AngularJS framework, as no potentially exploitable injections into the website, the forms or the emails could be achieved. While some general recommendations about header security were made in GTA-01-004, they had to be retracted as being false positives by design to satisfy the given business requirements of *X-Frame-Options* needing to be omitted. The general deployment of the web application with other services like AWS appears rock-solid and properly implemented, as no immediate issues could be identified.

The OTP implementation left a robust impression on the testers, since the generation, validation and invalidation of tokens left no room for brute-force attacks or similar mischievous action, for example due to leaked emails. Input-validation checks managed to identify no flaws except for the ones of little significance, as documented in the respective tickets GTA-01-002 and GTA-01-003. Both of the mentioned discoveries concern SSRFs, allowing potential attackers to initiate requests into the local network. Notably, these have unknown consequences, as Cure53 has not been given any insight into GovTech's internal network architecture.

FormSG uses TweetNaCl as the basis for all individual operations of its E2E encryption implementation. This is an absolutely praiseworthy choice, being a well-specified and publicly available cryptographic library. All relevant operations were found to be appropriately bound to the high-level application controllers on both the client and the server. The code responsible for managing form submissions on the server was observed to be extra careful in avoiding known pitfalls and any incorrect application of

Dr.-Ing. Mario Heiderich, Cure53
Bielefelder Str. 14
D 10709 Berlin
cure53.de · mario@cure53.de

cryptographic principles. The initial finding GTA-01-001 was later re-evaluated as being inapplicable, therefore the cryptographic implementation was concluded to be free of errors.

The audited source code is quite clean and of overall high quality. It was defensively programmed and follows most of today's industry standards for modern and secure web development. It was refreshing to observe that even some of the more exotic attributes like *samesite* were correctly applied to sessions, acting as a good additional defense mechanism. In summary, after re-evaluating the false positives in this report and possibly mitigating the remaining general weaknesses, it can be conclusively stated that GovTech has built a solid web application which does not leave much room for security-relevant errors.

Cure53 would like to thank Sonjia Yan, Yuanruo Liang, Leonard Loo, Darrell Wee and the rest of the Government Technology Agency (GovTech) team for their excellent project coordination, support and assistance, both before and during this assignment.