9 Post-Auth Blind NoSQL Injection in the users.list API leads to Remote Code Execution

Share:

TIMELINE

sonarsource submitted a report to Rocket.Chat. Mar 19th (2 years ago)

Summary:
The `users.list` API endpoint is vulnerable to NoSQL injection attacks. It can be used to take over accounts by leaking password reset tokens and 2FA secrets. Taking over an admin account leads to Remote Code Execution.

Description:
The `users.list` API endpoint takes a custom query via the `query` URL query parameter. Although the returned fields are restricted, the query is not validated or sanitized properly and can thus be used to perform a blind NoSQL injection that can leak any field's value of any document in the `users` collection.

By using MongoDB's `$where` operator, an attacker can build arbitrary oracles that can leak the value of any field of any user document. The query can be tailored to leak only the values of a specific account which makes it easy to target an admin account. Most notably an attacker can leak password reset tokens and 2FA secrets.

Example: in order to check if the password reset token of an admin user begins with a specific letter, e.g. `A`, the attacker would send the JSON object `{"$where":"this.roles.includes('admin') && /^A/.test(this.services.password.reset.token)"}` as the `query` parameter. The response contains the matching admin user when the guess was correct, or no users otherwise. This can be repeated for all possible characters and for each position in the token, until the whole token is known. See the `users_nosqli_blind_leak` function in the attached exploit for an implementation of this.

In order to take over another account, an attacker would perform the following high-level steps:

1. Leak the user's email address
2. Request a password reset for the target user's account
3. Leak the password reset token
4. Leak the TOTP 2FA secret or email 2FA token hash if necessary
5. Reset the target user's password to an attacker known one using the password reset token and any leaked 2FA tokens/secrets if necessary

To gain Remote Code Execution capabilities on the server, an attacker can follow these steps to take over an admin account. The attacker can then use the newly gained admin privileges to create an incoming web hook that has a script. This allows them to execute commands or get a shell on the server, because the script is executed on the server without a security boundary in place (which seems to be intended).

The vulnerable code can be found here: users.js:230

See `post_auth_nosqli.py` for a reference exploit and the attached video for a demonstration of it.

Releases Affected:

- Tested on 3.12.1
- Seems to be affected since 0.49.0 as the vulnerability was introduced in commit 3112d22

Steps To Reproduce (from initial installation to vulnerability):

1. Install Python3 (required by the exploit)
2. Install the Python dependencies required by the exploit: `pip3 install requests bcrypt`
3. Set up an instance of RocketChat 3.12.1, e.g. by cloning the repo and using Docker Compose:
4. `git clone git@github.com:RocketChat/Rocket.Chat.git`
5. `cd Rocket.Chat`
6. `git checkout tags/3.12.1`
7. `docker-compose up -d`
8. Configure the instance with default settings
9. Create a normal (non-admin) user with username `attacker` and password `attacker`
10. Run the reference exploit against the instance: `python3 post_auth_nosqli.py -u attacker -p attacker 'http://localhost:3000'`
11. The exploit should provide an interactive shell on the the server, use it to verify that you can execute commands as the rocketchat user: `whoami`

Supporting Material/References:

The attached proof-of-concept video shows the setup and exploitation of a fresh Rocket.Chat instance.
Please note: The unsuccessful login at the end of the video does not mean that the exploit did not work, it just shows that the original admin password was restored (as stated in the exploits output). The exploit was successful, which can be seen by the output of the shell commands at the end of the exploit.

This is the exploit's output:

Code 1019 Bytes Wrap lines Copy Download

```
 ___  ___  _ __   __ _ _ __ ___  ___  _   _ _ __ ___ ___
/ __|/ _ \| '_ \ / _` | '__/ __|/ _ \| | | | '__/ __/ _ \
\__ \ (_) | | | | (_| | |  \__ \ (_) | |_| | | | (_|  __/
|___/\___/|_| |_|\__,_|_|  |___/\___/ \__,_|_|  \___\___|

[+] Found admin: username=admin id=56gyPQKt8Ff3Weowk
[*] Leaking email...
[+] Leaked email: admin@rocketchat.local
[*] Leaking password hash...
[+] Leaked password hash: $2b$10$ubhEIM/j6qLFNINHVbP.B.CJFCXagK7V5zD0Q8BYzs6UBlbBpiECa
[+] Requesting password reset...
[*] Leaking password reset token...
[+] Leaked password reset token: ET4sx905cF9pTZOsHFu6eRad7MwpYmqs-iTMWQIXAhv
[+] Resetting password to "DEbCf2b0A2BE79bBcDf1"...
```

```
[+] Restoring admin password...
[+] Dropping into shell:
$ whoami
rocketchat
$ id
uid=65533(rocketchat) gid=65533(rocketchat) groups=65533(rocketchat)
$
```

Suggested mitigation

- Properly validate the `query` parameter:
  - Restrict the usage of MongoDB operators using an allowlist, especially top level operators like `$where`
  - Restrict the set of query-able fields using an allowlist (like the restriction on the returned fields)
- Check every API endpoint that uses the `parseJsonQuery()` function for similar vulnerabilities

Disclosure Policy

All reported issues are subject to a 90 day disclosure deadline.
After 90 days elapse, parts of the bug report will become visible to the public.

Don't hesitate to ask if you have any questions or need further help with this issue.

Impact

An attacker can use this vulnerability to target an admin user and take over their account, which is already a high impact. The attacker can then use certain features that are available to admins in order to gain Remote Code Execution capabilities. This is demonstrated in the reference exploit by creating an incoming web hook that executes the attacker's payload in the context of the server process.

This gives them complete control over the Rocket.Chat instance and exposes all attached components, e.g. the database or any external system whose credentials are stored within Rocket.Chat settings. An attacker can read, change, or delete all items in the database, impacting confidentiality, integrity, and availability.

markus-rocketchat changed the status to **Triaged**. Mar 21st (2 years ago)

@sonarsource
thanks a lot for reporting this to us. We will work on a fix and I will keep you posted on the developments.

Best
Markus

sonarsource posted a comment. Mar 26th (2 years ago)

Hi @markus-rocketchat,

We have found another endpoint that is vulnerable in the same way.
The `users.autocomplete` API endpoint takes the `selector` query parameter which is then JSON-decoded.
It is then passed to the `findUsersToAutocomplete()` method where the `conditions` property is taken and passed to `Users.findActiveByUsernameOrNameRegexWithExceptionsAndConditions()`. There the `conditions` parameter (coming from the user input) is merged with the query object in a way that allows to overwrite the whole query. This allows an attacker to use the `$where` operator like we explained in the original report.

The user input flows like this:

1. app/api/server/v1/users.js:801
2. app/api/server/v1/users.js:807
3. app/api/server/lib/users.js:25
4. app/models/server/raw/Users.js:257

We have also found another way of exploiting both vulnerable endpoints. It involves extracting values from the database via error messages, which is why we propose another general remediation in addition to the ones listed in the original report:
Error messages from internal components, such as the database, should bot be included in API responses because they might include sensitive information. They should instead be replaced with generalized error messages.

Sorry for the late addition! If you have any questions or need more details you can ask anytime.
Kind regards

markus-rocketchat posted a comment. Mar 26th (2 years ago)

Hi @sonarsource

no need to apologize, in fact THANK YOU a lot! This is extremely valuable information and I have already passed it on to the devs for evaluation.

Best
Markus

markus-rocketchat closed the report and changed the status to **Resolved**. Apr 15th (2 years ago)

Hi @sonarsource

thanks a lot again for submitting the report, which was very useful for our team. We applied a fix in the recent hotfix releases: 3.13.2, 3.12.4, 3.11.4
Please let us know if you come any way to circumvent our fix or if you want to share additional feedback.

Best
Markus

sonarsource posted a comment. Apr 16th (2 years ago)

Hi @markus-rocketchat

The fix looks good!
We would like to assign a CVE for this vulnerability, should we handle it or do you want to do it?

thank you for your feedback!

about CVE: both ways would work for us, We just would like to ask to maintain a responsible disclosure timeline for at least 30 days since the fix went live. If you want us to request the CVE, please request a disclosure of the report and the CVE will be published together with the report after the responsible disclosure.

sonarsource requested to disclose this report. May 10th (2 years ago)

Hi,

We would like to request disclosure for this report so that a CVE gets assigned.
We are going to cover the bug in a blog post next Tuesday, so it would be nice for us to have the CVE ready then. Since the 30 days after the fix will be over this Friday (14th of May), is it possible to disclose the report on that date?

Please remove the exploit script from the report before the disclosing it.

Thanks!

sonarsource posted a comment. May 12th (2 years ago)

Hi,

We found that the variant of the vulnerability (in the `users.autocomplete` endpoint) we mentioned earlier is still vulnerable because of a flaw in the fix.

The `clean` function in app/api/server/lib/cleanQuery.ts that is used to remove forbidden operators from a query object uses a regex to check if a property name starts with `$` :

Code 375 Bytes Wrap lines Copy Download

```
export function clean(v: Query, allowList: string[] = []): Query {
  const typedParam = removeDangerousProps(v);
  if (v instanceof Object) {
      /* eslint-disable guard-for-in */
      for (const key in typedParam) {
          if (/^$/.test(key) && !allowList.includes(key)) {
              delete typedParam[key];
          } else {
              clean(typedParam[key], allowList);
              }
          }
      }
      return typedParam;
  }
```

The regex is flawed because `$` has a special meaning. It represents the end of the input and not a literal `$` character when used in regular expressions.
Because of this, the function only removes empty property names, and not the ones starting with `$` . This allows the use of dangerous operators such as `$where` .

To patch this, it is best to use `key.startsWith('$')` instead of a regex, because it is more clear and less error-prone.

As mentioned in our last message, we will release a blog post on Tuesday that will feature the original vulnerability in the `users.list` endpoint, the variant in the `users.autocomplete` endpoint will not be mentioned in it.

markus-rocketchat posted a comment. May 18th (2 years ago)

Hi @sonarsource

the CVE is requested and will be published once the report is disclosed.

Could you please remove your last comment as it is a different vulnerability? I have informed our team already about it and please feel free to submit another Hackerone vulnerability for it.

Once you remove the comment, I can disclose directly.

sonarsource posted a comment. May 19th (2 years ago)

Hi @markus-rocketchat

It seems we cannot remove or edit comments, at least HackerOne's UI does not give us the option to do it.
If you can remove or hide it (like the exploit script), please do so.

We will create a new report about the bypass.

Thanks!

markus-rocketchat cancelled the request to disclose this report. May 19th (2 years ago)

canceled to edit comment

markus-rocketchat posted a comment. May 19th (2 years ago)

i just unlocked from the disclosure process, but cant do comment modification either. We would like to wait with the disclosure until we have a fix for the other endpoint. would that be ok? if you have a blog or external disclosure, you could already disclose the previous one.

sonarsource posted a comment. May 19th (2 years ago)

Yes that is fine, we don't want to disclose unfixed vulnerabilities.
FYI, we have released a blog post on the fixed ones yesterday: https://blog.sonarsource.com/nosql-injections-in-rocket-chat

markus-rocketchat posted a comment. May 28th (2 years ago)

Hi @sonarsource

If you would like, we could start the disclosure process and disclose in 30 days.

Best
Markus

sonarsource requested to disclose this report. May 31st (2 years ago)

Hi @markus-rocketchat

looks good!
Please remove the exploit script before disclosure.

Thanks!

sonarsource posted a comment. Jun 28th (about 1 year ago)

Hi,

As we are close to the disclosure (2 days) please don't forget to remove the exploit script from the report before it goes public.

Thanks!

sonarsource cancelled the request to disclose this report. Jun 30th (about 1 year ago)

Cancelling disclosure because the exploit script was not removed yet. Please remove it and then disclose this report.
Thanks!

markus-rocketchat posted a comment. Jun 30th (about 1 year ago)

@sonarsource wow, apologies! just saw this now and removed the script. should be good to disclose now!

sonarsource requested to disclose this report. Jul 1st (about 1 year ago)

No worries and thanks for removing it!

This report has been disclosed. Jul 31st (about 1 year ago)