Working with Firebase Rules

Overview

Firebase provides a powerful real-time database system that enables extremely quick read and write operations. Unlike SQL or other conventional database services, Firebase database does not employ a username/password based system to validate modifications to data. Instead, Firebase uses a highly intuitive rules-based system that ensures your data can be used only by those who you authorize.

How does it work

Firebase Rules are written in JSON and JavaScript. They work alongside Firebase Authentication to restrict read/write access to specific users. It is also possible to validate data before writing it to the database. In this guide, we'll take a look at the different rules one can use to secure their data, and some best practices while writing rules.

Use cases for Firebase Rules

Testing mode

Testing your app is easy when you have full access to your database. To enable read/write access to all tags, use:

```
{
  "rules": {
    ".read" : true,
    ".write" : true,
  }
}
```

> **Info**
>
> All your rules should be wrapped inside the `"rules"` tag. Read rules are always set using the `".read"` tag, and write with `".write"`.

The JSON code above specifies two rules for your entire database. Each rule can be thought of as a condition that has to return `true` if the operation is to succeed. In the case above, both `.read` and `.write` always return `true`, meaning no checks are required before accessing your database.

> **Warning**
>
> **Never** set read and write to `true` in a production environment. Your database can be hacked easily and worse, can be wiped clean by unauthorized users.

Authenticated-only mode

You can enable only authenticated users to read and write tags in your database. Users can be authenticated using the Firebase Authentication component.

```
{
  "rules": {
    ".read" : "auth.uid != null",
    ".write" : "auth.uid != null",
  }
}
```

Every authenticated user has a unique "uid". If the uid of the current user is `null`, then they haven't been authenticated. Such users will not be allowed to read/write data. This is the default Firebase setting.

The uid of a user can be accessed from the `user ID` variable in the Login Success and Current User Success blocks in the Firebase Authentication component.

> **Caution**
>
> These rules are barely more secure than the testing mode rules. Note that your users can still access **all** tags and values (including other users' personal data), and can still potentially clear your entire database.

Uid-Tag mode

You can also write rules to restrict read and write access to specific tags. With such rules, you can ensure that the user can edit only the data which they have created.

Considering this database:

```
- users
  - U43gnonNP9joboLnv
      - ranking : 1
        - email : "example@example.ex"
        - age : 20
    - greuo80NOpwqM8100
      - ranking : 67
        - email : "something@someone.com"
        - age : 15
    - kpmkmnn2YIHVI542a
      - ranking : 2
        - email : "user@product.io"
        - age : 45
- challenges
    - FindingNemo
    - name : "Finding Nemo"
        - recommended_age: 15
    - Marathon
      - name : "25mi Marathon"
        - recommended_age: 20
```

The structure of the database is:

```
- users
    - uid
        - ranking
        - email
        - age
- challenges
```

```
    - challengeId
        - name
        - recommended_age
```

Every user has their own tag (which is identified by their uid), and each tag contains data about the user. We want users to be able to edit only their data, meaning a user with uid `U43gnonNP9joboLnv` should be able to edit the ranking, email, and age only for the tag with the name `U43gnonNP9joboLnv`.

However, we wish to let all users see the challenges, but not let anyone edit them (challenges are edited directly from the Firebase console/CLI).

Here is how you would write the rules for such a system.

Start off with read/write rules for the `users` tag.

```
{
  "rules" : {
    "users" : {
            "$user_id" : {
                ".read" : "auth.uid == $user_id",
                ".write" : "auth.uid == $user_id"
            }
        }
  }
}
```

The `$user_id` tag is a wildcard reference to all keys that are inside the `users` tag. We allow read and write operations only if the name of the tag (as specified in `$user_id`) equals of the uid of the authenticated user. This way, a user will not be able to access tags that are not theirs as `auth.uid` will not be the name of the tag.

On the other hand, `challenges` can be read by all and written by none.

```
"challenges" : {
    ".read" : true,
    ".write" : false
}
```

The final JSON object will look like

```
{
  "rules" : {
    "users" : {
            "$user_id" : {
                ".read" : "auth.uid == $user_id",
                ".write" : "auth.uid == $user_id"
            },
            "challenges" : {
                ".read" : true,
                ".write" : false
            }
        }
  }
}
```

Data validation

With Firebase Rules, you can validate user data before it is sent to the database. Firebase provides two variables for this purpose: `data` and `newData`.

- `data`

This variable holds the current value of the database tag. You can use `data.val()` to fetch the value, or use `data.hasChild(child)` to check if it has the specified child. Firebase also provides several other functions you can use on your data. More information can be found here

- `newData` This variable holds the value the user is attempting to write to the database. `newData` also supports `val`, `hasChild`, and other functions that `data` has.

Best practices

Avoid overlap

Due to the nature of how Firebase database works, a rule set on a certain tag applies to all of its children. In the example used in Uid-Tag mode, we've set read/write conditions to the "challenges" tag. These rules apply to all challenges

inside the tag, and to every `name` and `recommended_age` inside each challenge. It is not possible to override rules for child tags if a parent tag already has a rule.

In essence, Firebase Rules are applied top to bottom. Shallower rules overwrite deeper ones. It is thus advised to structure your database such that you group tags that have similar rules. For example, your database can have two main tags: `public` and `private`, and deny read access to all tags inside `private`. Thus, you will be writing fewer rules and avoiding redundant ones. The more complex your database structure is, the harder it is to write coherent rules that do not have loopholes or security flaws.

Avoid unauthenticated access

It is always better to set a rule to `auth.uid != null` instead of `true`. While both are nearly the same, the former prevents unauthorized users from accessing your data. So in case something goes wrong, you will know for certain that it was one of your authenticated users who caused it.

Give limited access

Write your rules such that your users can access only that data which they can view. It is not recommended to give your users access to your entire database.

Lock access in development

Set all rules to `false` in a development environment where you would not want anyone to edit your database. It is also suggested you lock your database if your product is shutting down.

> **Note**
>
> You will still be able to access and edit your database either from your Firebase console or from the command-line interface (CLI).

Conclusion

Firebase Rules are great for securing your database and channeling users to the right tags and values. With the right set of rules, your database will be practically unhackable!

Further reading

- Firebase Security Rules
- Basic Security Rules
- Firebase Security & Rules - Medium
- Common Firebase Rules

Last update: January 24, 2020