Matt Woodward's Blog

Web development and general geekery

SATURDAY, MARCH 17, 2012

The Definitive Guide to CouchDB Authentication and Security

With a bold title like that I suppose I should clarify a bit. I finally got frustrated enough with all the disparate and seemingly incomplete information on this topic to want to gather everything I know about this topic into a single place, both so I have it for my own reference but also in the hopes that it will help others.

Since CouchDB is just an HTTP resource and can be secured at that level along the same lines as you'd secure any HTTP resource, I should also point out that I will not be covering things like putting a proxy in front of CouchDB, using SSL with CouchDB, or anything along those lines. This post is strictly limited to how authentication and security work within CouchDB itself.

CouchDB security is powerful and granular but frankly it's also a bit quirky and counterintuitive. What I'm outlining here is my understanding of all of this after taking several runs at it, reading everything I could find on the Internet (yes, the whole Internet!), and a great deal of trial and error. That said if there's something I'm missing or not stating accurately here I would LOVE to be corrected.

Basically the way security works in CouchDB is that users are stored in the `_users` database (or elsewhere if you like; this can be changed in the config file), and security revolves around three user roles:

- Server admin
- Database admin
- Database reader

Notice one missing? That's right, there is *not* a defined database reader/writer or database writer role. We'll get to that in a minute. And of course you can define your own roles provided that you write the functionality to make them meaningful to your databases.

Here's how the three basic roles play out:

- Server admins can do anything across the entire server. This includes creating/deleting databases, managing users, and full admin access to all databases, i.e. full CRUD on all documents as well as the ability to create/modify/delete views, run compaction and replication, etc. In short, god mode.
- Database admins have full read/write access (including design documents) on specific databases and can also modify security settings on a specific database. (I don't know if database admins can manage replication because I did not test that specifically.)
- Database readers can only read documents and views on a specific database, and have no other permissions.

Even given all of this, reading and writing in CouchDB needs more clarification so you know what is and isn't allowed:

- By default all databases are read/write enabled for anonymous users, even if you define database admins on a database. Note that this includes the ability to call design documents via GET, but does not include the ability to create/edit/delete design documents. Once you turn off admin party you have to be a server or database admin in order to manage design documents.
- If you define any database readers on a database anonymous reads are disabled, but anonymous writes (of regular documents, not design documents) are still enabled.
- In order to prohibit anonymous writes, you must create a design document containing a validation function in each database to handle this (much more on this below).
- Regardless of any other settings server admins always have full access to everything, with the exception that if you create a validation function the admin user's access is impacted by any rules in that validation function. More on this below, but basically if you create a validation function looking for a specific user or role and the admin user doesn't match the criteria, they'll be blocked just like anyone else.

ABOUT ME

Matt Woodward

- Principal IT Specialist, US Senate
- Web developer using va variety of tools including Groovy/Grails, Java, Python/Django, and CouchDB
- All-around geek

Blocking Anonymous Writes

So now we come to the issue of blocking anonymous writes (meaning create/update/delete), and it's simple enough but I have no idea why this isn't done at the user level. Maybe there's a logical reason that isn't written down anywhere, but why you can't create a reader/writer user or role is a mystery to me.

But enough whining. Here's how you do it.

To block anonymous writes you have to create a design document in the database that contains what's called a validation function. This basically means that your design document must contain a `validate_doc_update` field, and the ID for this document follows the standard pattern for design documents, e.g. something like `_design/blockAnonymousWrites` The value of the `validate_doc_update` field is a function that will be run before all write operations, and it takes the new document, the old document (which would be null on create operations), and the user context in as arguments. This gives you access to everything you need to do simple things like check for a valid user, or more complex things like seeing if specific fields exist in the document that's about to be written or even if there are conflicts on an update operation with the old version of the document that you want to reject.

Here's a sample validation function that simply checks for a specific user name, foo, and rejects the write operation if the user is not foo:

```
function(new_doc, old_doc, userCtx) {   if(userCtx.name != 'foo') {
throw({forbidden: "Not Authorized"});   } }
```

The `userCtx` object has properties of `name` and `roles`. The `name` property is the user name as a string, and `roles` is an array of role strings.

Let's say you wanted to limit write operations to the role `bar`. To accomplish that you'd use JavaScript's `indexOf()` method on the `userCtx.roles` array to see if the required role exists:

```
function(new_doc, old_doc, userCtx) {   if(userCtx.roles.indexOf('bar')
== -1) {     throw({forbidden: "Not Authorized"});   } }
```

Obviously on top of all of this you have access to all the properties of the document being posted as well as the old document if it's a revision, and you can use all that information to do whatever additional validation you need on the document data itself before allowing the document to be written to the database.

Creating Users

As far as creating users is concerned you can either do this in Futon or (as with everything in CouchDB) via the HTTP API. Note that if you create users via Futon you need to be aware that if you are logged in as admin and click the "Setup more admins" link you're creating a **server admin**. That means they have permission to do literally everything on that CouchDB server.

If you want to create a non-admin user make sure you're logged out and click on the "Signup" link, and you can create a user that way. Note that this doesn't work on BigCouch if you're hitting Futon on port 5984 since the `_users` database lives on port 5986 in BigCouch, and that backend port is by default only accessible via localhost; more on that below. And big thanks to Robert Newson on the CouchDB mailing list for pointing that out since I was tearing my hair out a bit after my recent migration to BigCouch.

If you want to create users via the HTTP API, in CouchDB 1.2 or higher you simply do a PUT to the `_users` database via curl or another HTTP tool, or make an HTTP call via your favorite scripting language. I'll show all the examples in curl since it's language agnostic and universally available (not to mention because I find curl so damn handy).

```
curl -X PUT http://mycouch:5984/_users/org.couchdb.user:bob -d
'{"name":"bob", "password":"bobspassword", "roles":[], "type":"user"}'
-H "Content-Type: application/json"
```

That will create a user document with an ID of `org.couchdb.user:bob` and a user name of `bob`, and `bob` is *not* a server admin. In CouchDB 1.2 it will see the password field in the document and automatically create a password salt and hash the password for you.

On versions of CouchDB prior to 1.2, or with servers based on versions of CouchDB prior to 1.2 such as BigCouch 0.4.0 (which is based on CouchDB 1.1.1), the auto-salt-hash bit does not happen. This means you need to salt and hash the password information and store the hashed password and the salt in the user document.

As a reminder in case you weren't paying attention earlier: On BigCouch the `_users` database is on port 5986. This had me banging my head against my desk for the better part of an afternoon. It's probably documented somewhere but you know geeks and reading manuals, so I'm sharing that important tidbit in the hopes it helps someone else.

To create a user on CouchDB < 1.2 or BigCouch 0.4.0 (which again is based on CouchDB 1.1.1) you first need to:

- Create a salt
- Hash the concatenation of the password and the salt using SHA1
- Include the salt used as the `salt` property of your user document, and the hashed password as the `password_sha` property of your user document

There are numerous ways to do all of this and you can see some examples in various languages and technologies on the CouchDB wiki, but since openssl is standard and a quick and easy way to do things I'll recap that method here.

First you need to generate a salt:
```
SALT=`openssl rand 16 | openssl md5`
```

Next echo that out just to make sure it got set properly:
```
echo $SALT
```

Next you concatenate whatever password you want + the salt, and then hash the password using SHA1:
```
echo -n "thepasswordhere$SALT" | openssl sha1
```

One caveat: if when you echo `$SALT` it contains `(stdin)` at the beginning like so:
```
(stdin)= 4e8096c4d0047e8d535df4b356b8d102
```

Make sure NOT to include the `(stdin)=` part in what you're going to put into CouchDB. Ignore `(stdin)=` and the space that follows and use only the hex value.

After generating a salt and hashing the password the end result that you put in CouchDB looks something like this (you'd obviously replace `thehashedpassword` and `thesalt` with the appropriate values):
```
curl -X PUT http://mycouch:5984/_users/org.couchdb.user:bob -d
'{"name":"bob", "password_sha":"thehashedpassword", "salt":"thesalt",
"roles":[], "type":"user"}' -H "Content-Type: application/json"
```

Of course if you know when you're creating the user that you want to grant them a specific role, you'd put that in the roles array. These roles will be contained in `userCtx.roles` in validation functions and you can act on that accordingly (see the above discussion about validation functions for more details).

And again note that if you're on BigCouch ***use port 5986 for the _users database!***

Summary

To sum all this up, here's a handy-dandy chart.

If you want to ...	*You need to ...*
Allow anonymous access to all functionality including creating and deleting databases	Do nothing! Leave admin party turned on. (At your own risk, of course.)
Disable anonymous server admin functionality (create/delete databases, etc.) but continue to allow anonymous read/write access (not including design documents) on all databases	Create at least one server admin user by clicking the "Fix this!" link next to the admin party warning on the lower right in Futon.
Allow a user who is not a server admin to have admin rights on a specific database	Create a non-server-admin user and assign them (by name or role) to be a database admin user on the specific database. This can be done via the "Security" icon at the top of Futon when you're in a specific database, or via the HTTP API.
Block	Create a non-server-admin user in CouchDB and assign

anonymous reads on a specific database	them (by name or role) to be a database reader on the specific database. This can be done via the "Security" icon at the top of Futon when you're in a specific database, or via the HTTP API.
• Block anonymous writes on a specific database	• Create a non-server-admin user in CouchDB and create a design document in the database that includes a validation function, specifically in a `validate_doc_update` property in the design document. The value of this property is a function (that you write) to check for a specific user name or role in the `userCtx` argument that is passed to the function, and you would throw an error in the function if the user or role is not one you want to write to the database.

And that's more or less all I know about CouchDB security. I'll end with some links if you want to explore further.

Any questions, corrections, or suggestions for clarification are very welcome. Hope some of you found this helpful!

Security/Validation Function Links

- http://wiki.apache.org/couchdb/Security_Features_Overview
- http://wiki.apache.org/couchdb/Security_Features_Overview#Salt_and_Password_Generator (steps for creating new non-admin users in a terminal using openssl)
- http://blog.lizconlan.com/sandbox/securing-couchdb.htm
- http://blog.couchbase.com/what%E2%80%99s-new-couchdb-10-%E2%80%94-part-4-security%E2%80%99n-stuff-users-authentication-authorisation-and-permissions
- http://spin.atomicobject.com/2011/12/22/authentication-and-couchdb/
- http://www.hbensalem.com/nosql/couchdb-validation-functions/
- http://custardbelly.com/blog/2011/03/04/jquery-mobile-couchdb-part-7-1-authorization-and-validation/
- http://guide.couchdb.org/draft/validation.html

-
- http://www.jasondavies.com/blog/2009/05/27/secure-cookie-authentication-couchdb/

Posted by Matt Woodward at 3:26 PM
Labels: CouchDB

10 comments:

-b said...

Thanks for writing this up! :-)

April 4, 2012 12:20 AM

jeisenlo said...

Hello Matt,

Thanks for this post!

I have an issue. I was able to install BigCouch, clicked the "Fix This" link at the bottom right to create an admin account. When I submitted the form with my credentials, it errored out, (something about the _users table missing) the _users database is not accessible and I can no longer login to Futon. I am 100% sure I am using the correct credentials. I see _users under recent databases, but when I click on it, it says that the database does not exist... any thoughts or suggestions would be appreciated.

James

April 28, 2012 9:55 AM

jeisenlo said...

BTW, I have tried to access the _user database on port 5986... not working. Am I trying it correctly? http://my-domain-name:5986/_utils/database.html?_users

April 28, 2012 10:00 AM

Matt Woodward said...

Hi James -- you're correct that the _users database on BigCouch is only available over port 5986 but by default that port is only accessible via localhost, so you have to be on the server itself to access anything over that port. In other words you can't use Futon for the stuff that lives on 5986.

April 30, 2012 8:07 AM

jeisenlo said...

Thank you, Matt. So, logging into Futon is no longer a possibility using BigCouch on either port 5984 or 5986? I am just assuming this due to my trouble and that the _users table cannot be accessed from port 5984...

May 2, 2012 1:49 PM

Matt Woodward said...

Right, in BigCouch the _users database exists ONLY on port 5986 and by default that's not accessible remotely. I just ssh into the server and use curl when I need to interact with the _users database.

May 2, 2012 2:17 PM

jeisenlo said...

Okay... got it. Thank you, Matt!

May 2, 2012 4:47 PM

Tiklup said...

Hey Matt,

The default validate_doc_update function for _users in CouchDB 1.2.0 is a pretty good example of how to ensure write security out-of-the-box. Hash & Salt are setup by the system after someone PUTS a request for their own user account.

But even with that ... the only way we can protect the user info of other folks from being read anonymously ... is to block off self-service for user creation completely ... that sucks, would you happen to have any thoughts on that based on work you've done so far?

June 6, 2012 8:53 AM

trisapeace said...

Thanks so much for this post. This is information that I wasn't able to find anywhere else. I couldn't even figure out how to create a new user on Futon before reading this.

That being said, I am running on IrisCouch 1.2.0 and I don't seem to need a design doc/validation function in order to block anonymous writes.

I've set it up so that I have
- one user who is a server admin.
- one user who is a database admin and a database member of the database db
- one user who has no access

If I run this curl command (anonymous user):

curl -X PUT http://mycouch.iriscouch.com/db/abc -d '{"test":"test"}'

or this one (no access user):

curl -X PUT http://noaccess:pass@mycouch.iriscouch.com/db/abc -d '{"test":"test"}'

then the insert fails and I get this message:

{"error":"unauthorized","reason":"You are not authorized to access this db."}

However, if I run this curl command (database admin user):

curl -X PUT http://dbadmin:pass@mycouch.iriscouch.com/db/abc -d '{"test":"test"}'

or this one (server admin user):

curl -X PUT http://serveradmin:pass@mycouch.iriscouch.com/db/abc -d '{"test":"test"}'

then it succeeds.

Am I missing a security hole? Or is Iris Couch's "Database Member" concept slightly different than the "Database Reader" that you discuss?

Thanks!
:-)
Theresa

June 15, 2012 9:26 AM

Paul Lysak said...

I also didn't have to create validation function after correctly setting up DB admins and user in CouchDB 1.2.0 - it just denies any anonymous access.

November 24, 2012 3:35 AM

Post a Comment

Newer Post Home Older Post

Subscribe to: Post Comments (Atom)