THE GREAT RIFT VALLEY SECURITY™

PLATFORM

THIS PAGE INTENTIONALLY LEFT BLANK

OVERVIEW

Security has been a principal consideration of the BAOBAB Server since the first line of code was written. It has a “token-based” security system that establishes “pools” of “access tokens.” If a user does not have a token required to view a data asset, that asset is never even brought into the system.

There are virtually no “user levels” in the BAOBAB Server. The concept of “privileges” doesn’t really exist. It’s all about “tokens.” Data items are secured by imbuing them with tokens. Each data item in the BAOBAB Server can be given a “read token” and a separate “write token,” so there is a difference between access and modification rights. Users log in and their accounts are given a “pool” of tokens.

BAOBAB USERS

There are two fundamental “user levels” in BAOBAB: The Standard User, and the “Manager.”

STANDARD USER LOGINS

All user logins feature a login ID and a password. When you establish a connection to the BAOBAB Server, you provide a login and password. There is a bit of technical back-and-forth involved, where the server returns a temporary “API Key” that is used for subsequent authorized access. Let’s just say that you log in with a login ID and password, and that you stay logged in for the duration of your session (We’ll get into more detail, later).

Each login is given one token (their own unique integer ID), and, possibly, a “pool” of other tokens. These are the available security tokens for this login. They will be authorized to view, and possibly modify, any asset that has specified one of the tokens assigned to the login.

Tokens (including the user’s own ID) are not exclusive to any login. Multiple logins can share a token. You cannot remove a login’s asset ID, so it will always have access to any asset that has its ID as a read or write token.

Logins can be associated with user IDs (“People” assets in the regular database). These “users” have more information that can be applied to the owner of the login. The Security Database has a very simple schema and does not associate very much information with each login.

MANAGER USER LOGINS

These are logins that are exactly the same as standard user logins, except that they can also create and delete other logins, as well as manage tokens. Standard logins can edit other logins (assuming they have the appropriate tokens) but are prevented from being able to create or delete other logins. Manager logins, however, can also edit the tokens for other logins (as long as they have full access to every token in the target login).

No login can edit their own tokens (even managers). This is a simple, standard security practice. If you want access to assets that use a certain token, then you need to ask your manager to give you that token.

“GOD” LOGIN

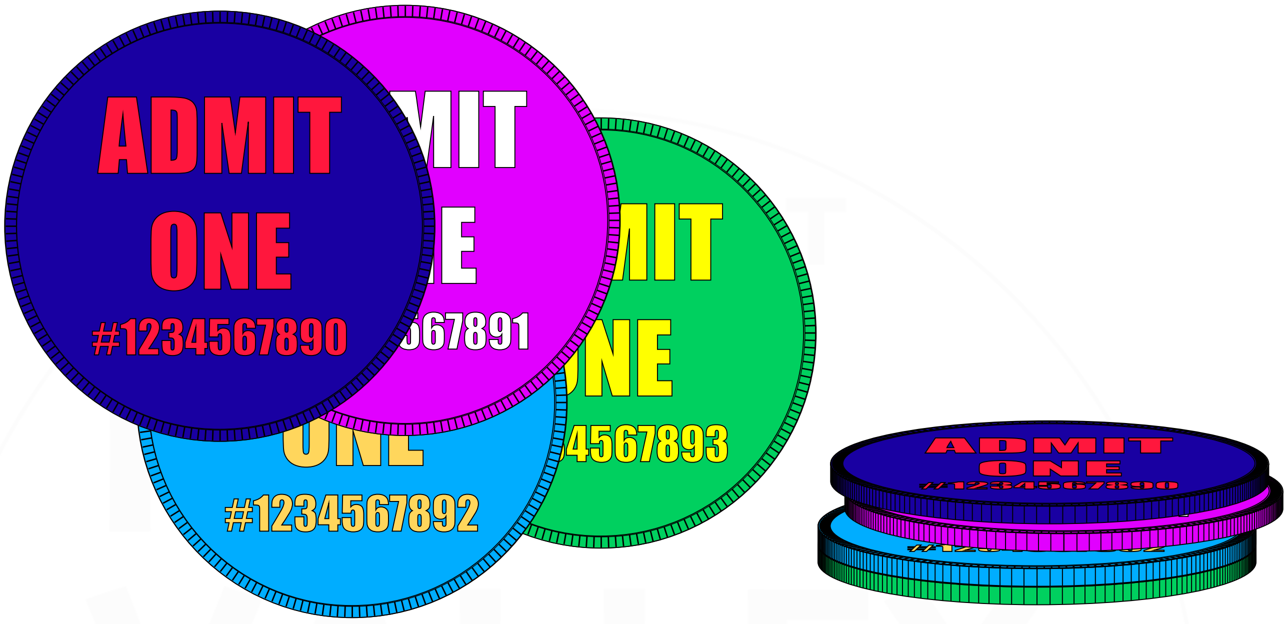
One single login (it does not have to be a manager login) can be “promoted” by the System Administrator to be the “Main” login (the “God” login). This login has full rights to every asset, login, and security token on the server, and should only be used for system administration.

This “promotion” is done at the code level by the Server Administrator. It is not assigned in the database or at runtime.

SECURITY TOKENS

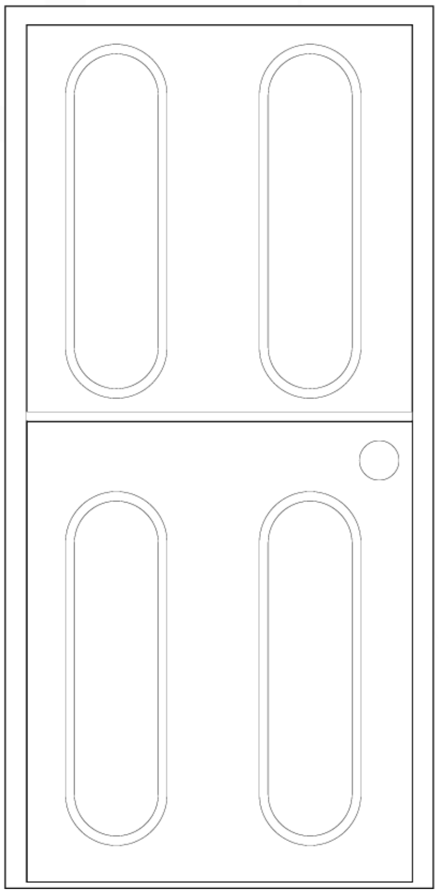
The BAOBAB Server uses what are termed “security tokens” to manage access and modification rights to data in the server. Tokens are simple, unique integers that come from the Security database.

Think of your classic fairground “ride token.”



When we create or edit an asset in the BAOBAB Server, we assign it two tokens: read and write.

THE DUTCH DOOR METAPHOR



We have probably all encountered “Dutch Doors.” These are standard-sized doors to rooms or buildings, that have a horizontal split, about halfway down, so that the top part can be opened, while the bottom stays closed.

This is most common in retail establishments, or “quartermaster” situations, like supply rooms. It allows the person “inside” the door, to control access to the area behind the door, while allowing a certain degree of access to those outside the door.

READ ACCESS

If only the top door is opened, we can think of that as “read” access, where we can see some things, but cannot enter the room. We can only see what is immediately visible via the upper part of the doorway.

WRITE ACCESS

If both doors are opened, then we can think of that as “write” access. We can enter the room, see more things, and even affect change, inside. Write access automatically confers read access. BAOBAB does not have a “write only” security level.

TOKEN ALLOCATION

Tokens are created and allocated by Manager users. Tokens can be created, but not destroyed. If a login is deleted, then their user ID actually gets turned into a standalone token (remember that user IDs are also security tokens).

Every user, whether Manager or Standard, has a “pool” of tokens. They can use these tokens to secure assets, and/or assign to other users (if they are a Manager).

Possession of a token does not automatically confer any rights. That is determined by the owner of an asset, and which tokens they assign to read or write for that asset.

THE “BUY-IN” STACK

When a user is first created by a manager, they are given an initial “stack” of tokens from the Manager’s pool. These tokens are ones that the manager has, as well, so the manager that created the user can also access any items that the user secures with the tokens. When a Manager creates a user, that user’s ID gets added to the Manager’s “token pool.”

GETTING MORE TOKENS

Tokens can be assigned to users by Managers, as long as the Manager has both write access to the user, and “owns” the tokens, in their own “pool.”

That means that a user can be created by a Manager and given a “stack” of tokens. They can then, subsequently, get more tokens from another Manager, that are not available to the creating Manager.

This means that it is entirely possible for users to have greater (or different) privileges than the Manager that created, and can make changes, to the user.

GOD SEES ALL

The “God Admin” user is a special case that has the ability to “see” everything. It is designed to be seldom used. However, it can also be a source for “extremely private” tokens, that no other user on the system has.

A CONCRETE EXAMPLE

Let’s say that we have an asset that represents a hospital (it would probably be a “place” asset).

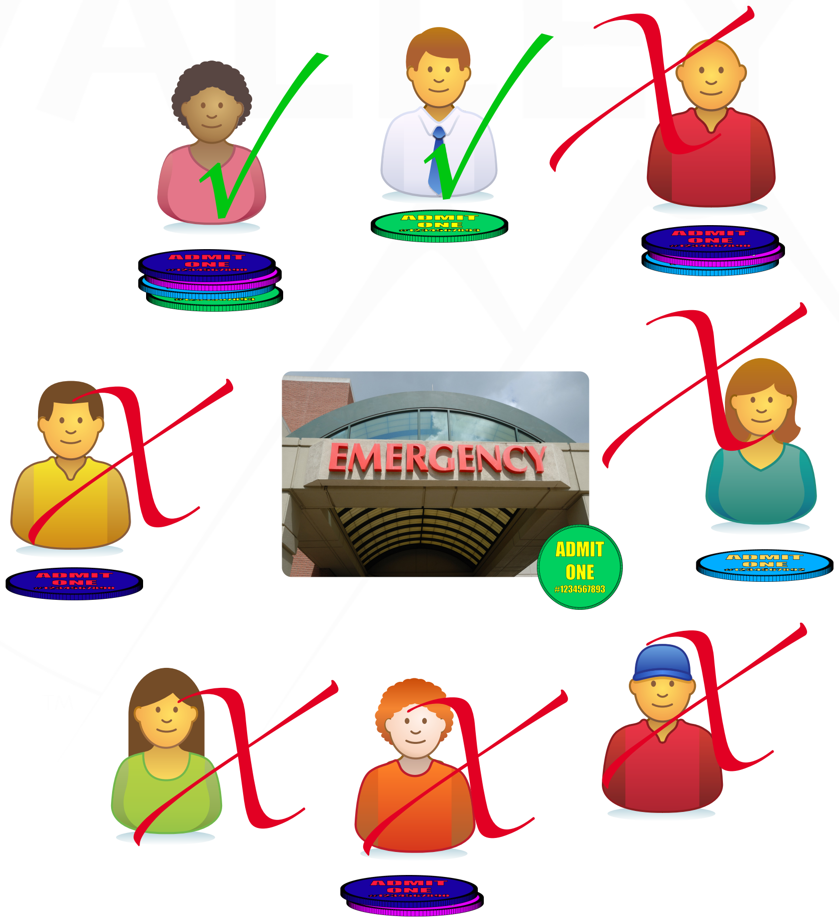
We may give read rights to everyone (no “read” token), but only Hospital Administrator users (these would probably be “people” assets with associated logins) can make changes to the place:

READ: NO TOKEN (0)

WRITE: 1234567893

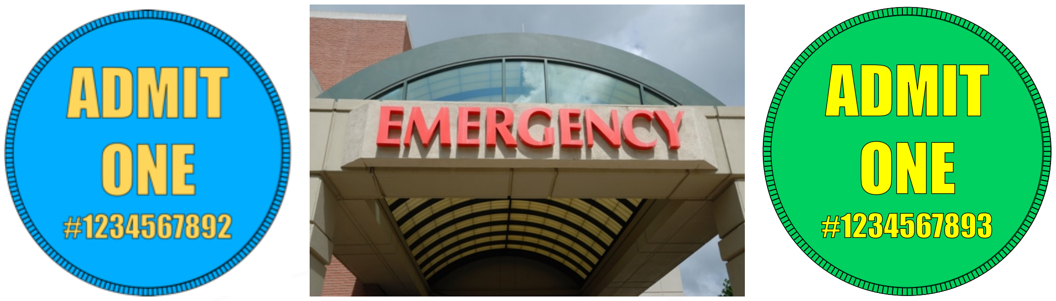
This means that anyone (whether logged in or not) can “see” the hospital, but only users that are logged in, and whose logins have the “1234567893” token (we are using green to represent that token) can make any changes to the hospital “place” in the BAOBAB Server database.

When a user logs in, their account needs to have the “green” token to be able to make changes to the hospital place.



Everyone can see the hospital, but of all the users above, only the two in the top left corner can make changes.

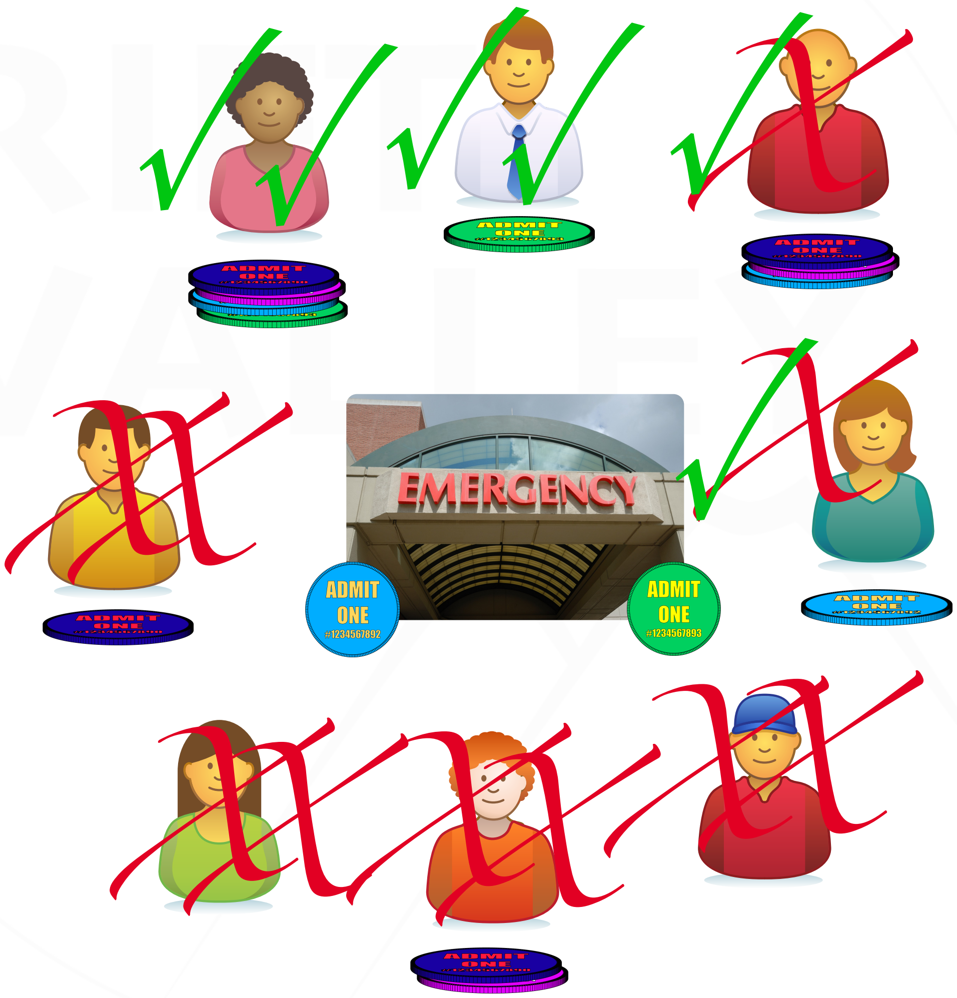
It is also possible to set it up so that only certain logged-in users can even see the hospital:



READ: 1234567892

WRITE: 1234567893

Which gives us this:



As before, the two users in the upper left are the only ones that have edit rights, but all the users in the lower left and bottom are now prevented from even seeing the asset. Having write permissions on an asset also confers read permission. We can only assign one token to each role.

The two users on the right can see, but not modify, the hospital. If either of them had a green token, as opposed to a light blue token, then they would be able to modify the hospital asset, as well.

It also doesn't matter how many tokens you have. You need to have the single correct token to view and/or modify the asset. It only takes one token; but it needs to be the CORRECT token.

NO PERMISSION, NO VISIBILITY

If you don't have read access to an asset, then that asset doesn't exist for you. It's not like you will see a locked door with a sign reading "NO ENTRY." You don't see any door at all.

This is a fairly classic security protocol.

IMPORTANT: *Because of this, just because we can't see an asset or user, does not mean they are not there. It may mean that they exist, but we do not have permission to see them.*

Finally, you can also set the read and/or write token of any asset to "1," which means that any logged-in user can see/change the asset, but no visitor without a login status can see it. Non-logged-in users cannot make changes to any assets; regardless of the assigned write token.

TOKEN MANAGEMENT

Tokens are simple database entries and are really just sequential integer numbers. Think of them like those cheap "raffle tickets" that you can buy at any party store.

The tickets, by themselves, have no intrinsic value. The value is determined by the assets to which they are assigned, as read or write tokens.

If we were to use the above analogy to describe our token system, we could think of it as being a single "source" ticket, with an endless number of "KEEP THIS COUPON" stubs.

Each "stub" could be given to someone, valid for admission to whatever attraction or service is indicated by the main ticket.

If anyone shows up, holding one of those stubs, they are given whatever admission is described by the main ticket (for example, they could be General Admission tickets, or Mezzanine tickets).

TOKEN POOLS

Security tokens are a "limited resource," in a BAOBAB server. They are generated, then copied, aggregated, and passed around. There can be infinite instances of copies of tokens ("stubs"), but a fixed number of original tokens. Tokens represent an entry in the BAOBAB "Security" database, but they aren't like Blockchain. They are simple sequential numbers, relevant only within the BAOBAB server.

Every login is assigned a "pool" of tokens, in addition to their own ID. Since a login is a Security database entity, it has an ID, and, as such, is a security token.

Tokens can be assigned to logins by Managers. Managers cannot assign tokens to themselves, and they can only assign tokens from their own "pools."

In order to assign a token to a login, the Manager must have write access to that login (they must have the same token as the login has in its "write_security_id" database column). When a user is created, it has its own ID set as both the read and write ID.

TOKEN DISTRIBUTION EXAMPLE

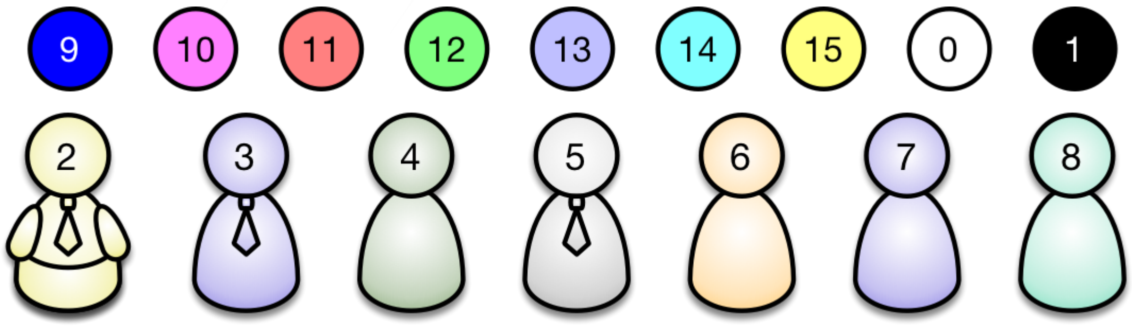
Let's say that we have a BAOBAB server, with a Security database that has the following IDS:

2. Login User; currently assigned the "God" role.
3. Manager Login
4. Login User
5. Manager User
6. Login User
7. Login User
8. Login User
9. Security Token
10. Security Token
11. Security Token
12. Security Token
13. Security Token
14. Security Token
15. Security Token

1 is reserved (all logged-in users), as is 0 (non-logged-in users).

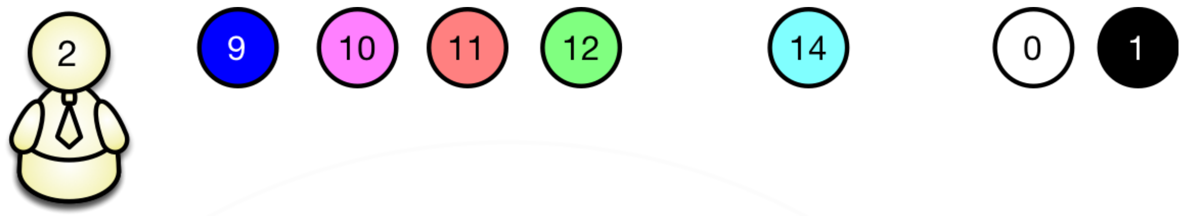
So we have 7 users, and 7 "standalone" security tokens. Remember that each user ID is also a security token, so that means that we have 14 "optional" security tokens, along with the two "built-in" tokens (0 and 1). There is one other "built-in" token: -1. This means that only the God Admin can see the asset. It is rarely used. Two of the users are Managers (can edit other users). One user is the "God Admin" user. It is implemented as a Standard (non-Manager) user, but has been elevated internally, to be the "God Admin," which means that it lives by a different set of rules.

In the following examples, don't worry about sequential numbers.



STEP ZERO: THE GOD ADMIN

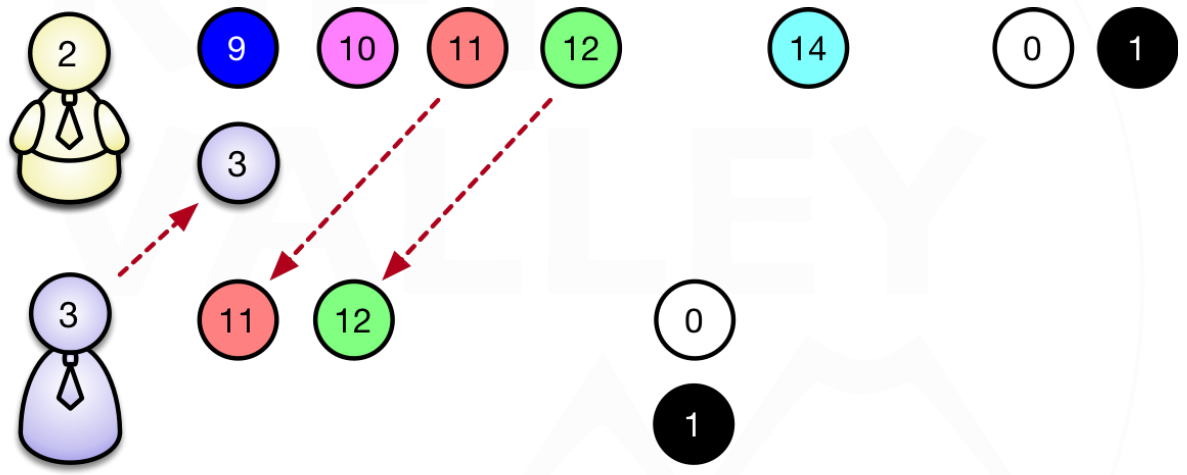
At first, we have only the God Admin (ID 2), and a few initial security tokens. Note that 0 and 1 are "intrinsic." They exist at all times, and are available to all users. Note also, that the God Admin always has access to ALL tokens on the system; regardless of whether or not they created the tokens:



STEP ONE: THE FIRST MANAGER

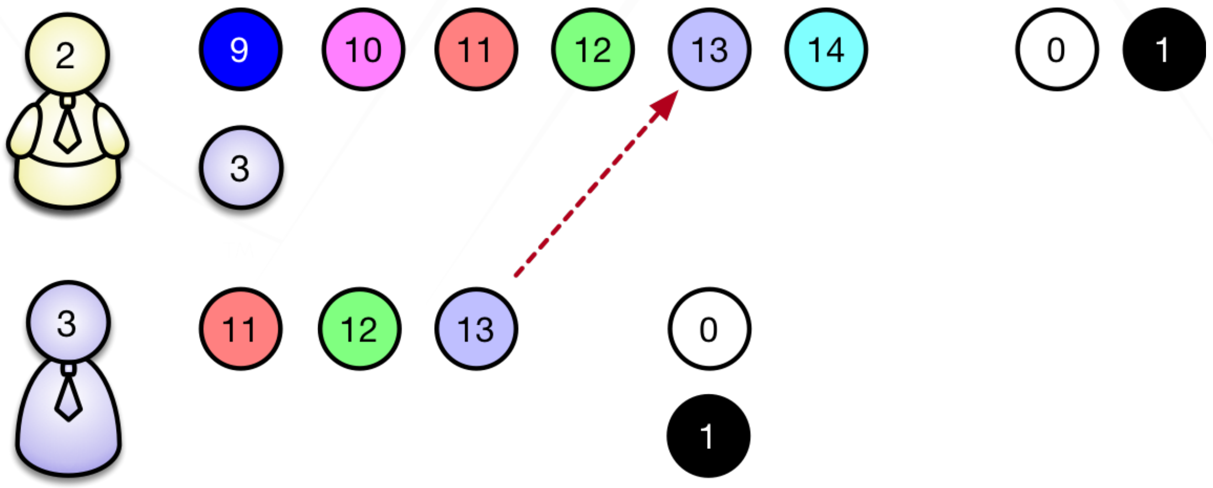
The God Admin creates a Manager user (ID 3), and gives it tokens 11 and 12. The Manager token (3) is automatically "bestowed" to the God Admin. Note that 0 and 1 are automatically available to the new Manager user.

So, at this point, User 3, which also happens to be a Manager, has a token "pool" of 3, 11, and 12, in addition to the two "intrinsic" tokens (0 and 1):



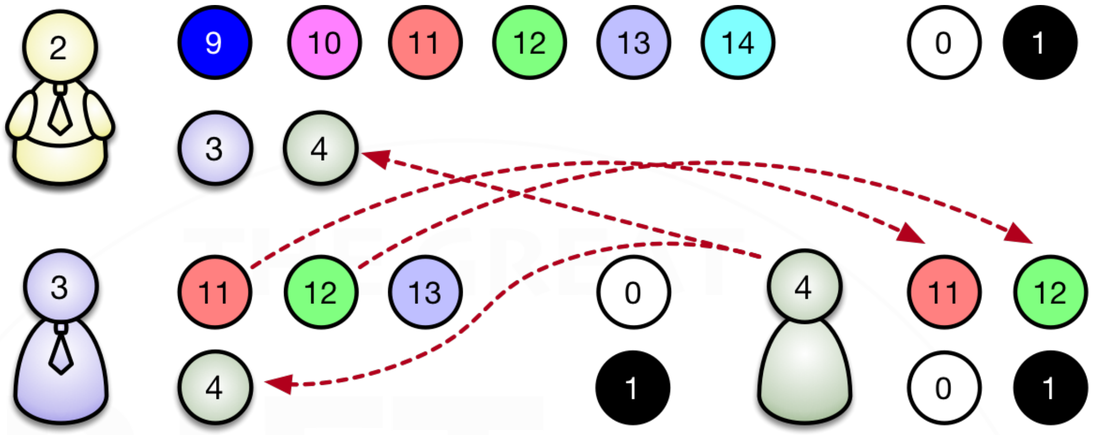
STEP TWO: THE MANAGER CREATES A TOKEN

Let's say that, at this point, the new manager creates a brand new Security token (Token 13), which is automatically made available to the God Admin. Remember that Managers can create new tokens:



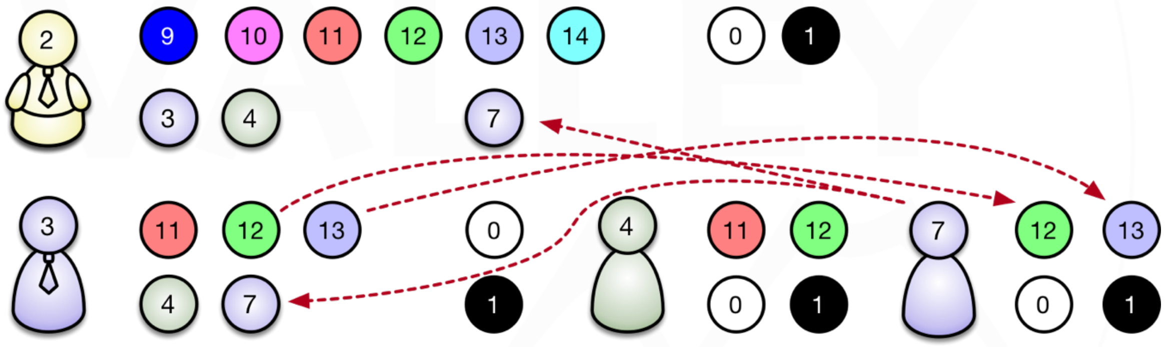
STEP THREE: THE MANAGER CREATES A USER

Now, let's say that the new Manager creates a Standard user. The Manager gives the new user Security Tokens 11 and 12 (from its own pool). The new user ID (4) is automatically assigned to Manager 3 (its creator), and also, the God Admin gets Security Token 4:



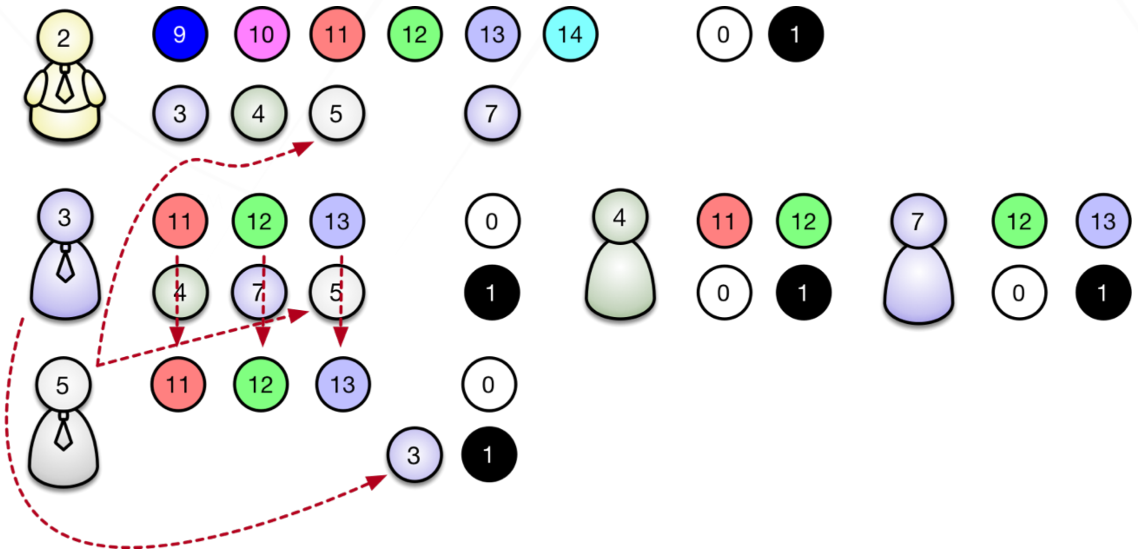
STEP FOUR: THE MANAGER CREATES ANOTHER USER

Let's say that Manager 3 decides to create another Standard user (User 7). It gives the new user Tokens 12 and 13. The new user ID (7) is automatically added to the Manager's "pool," as well as being made available to the God Admin:



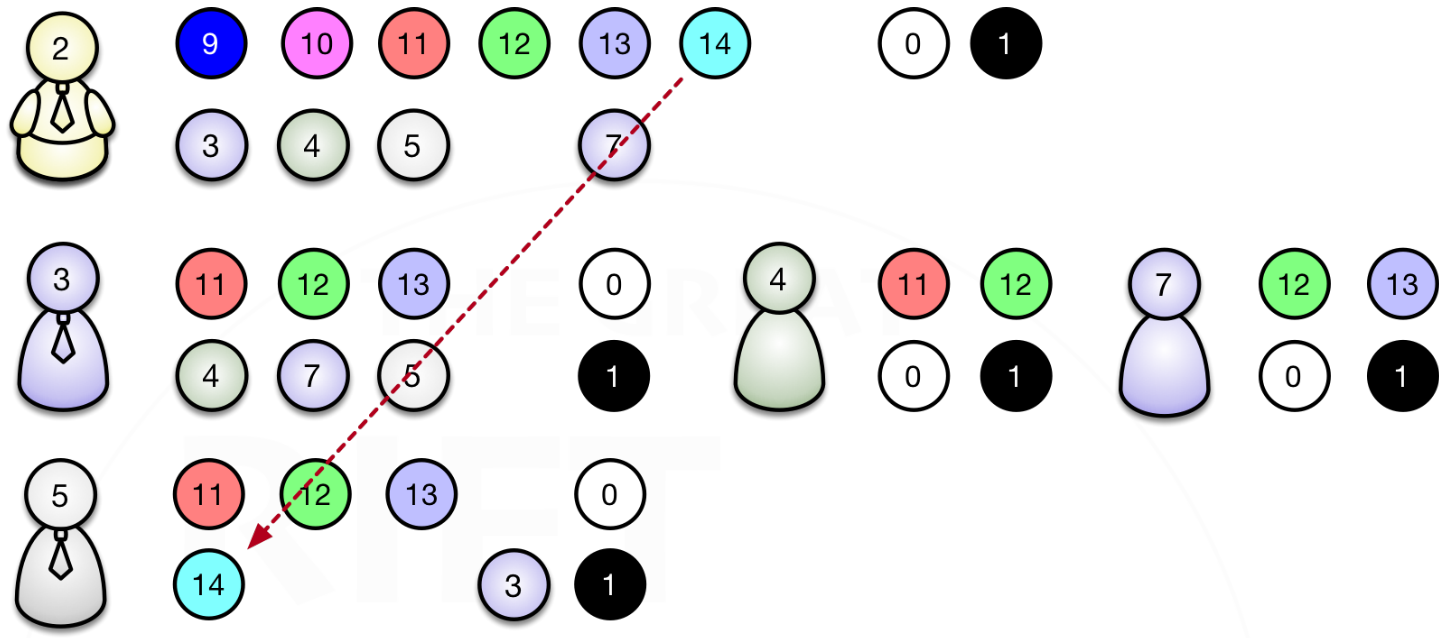
STEP FIVE: THE MANAGER CREATES ANOTHER MANAGER

Now, Manager 3 creates another user. This time, the user is a Manager. Manager ID 3 gives the new user (ID 5) all three of its tokens (11, 12, 13), and, also, take note that it gives the new manager its own token (Token 3). This means that the new Manager will be able to edit its own creator:



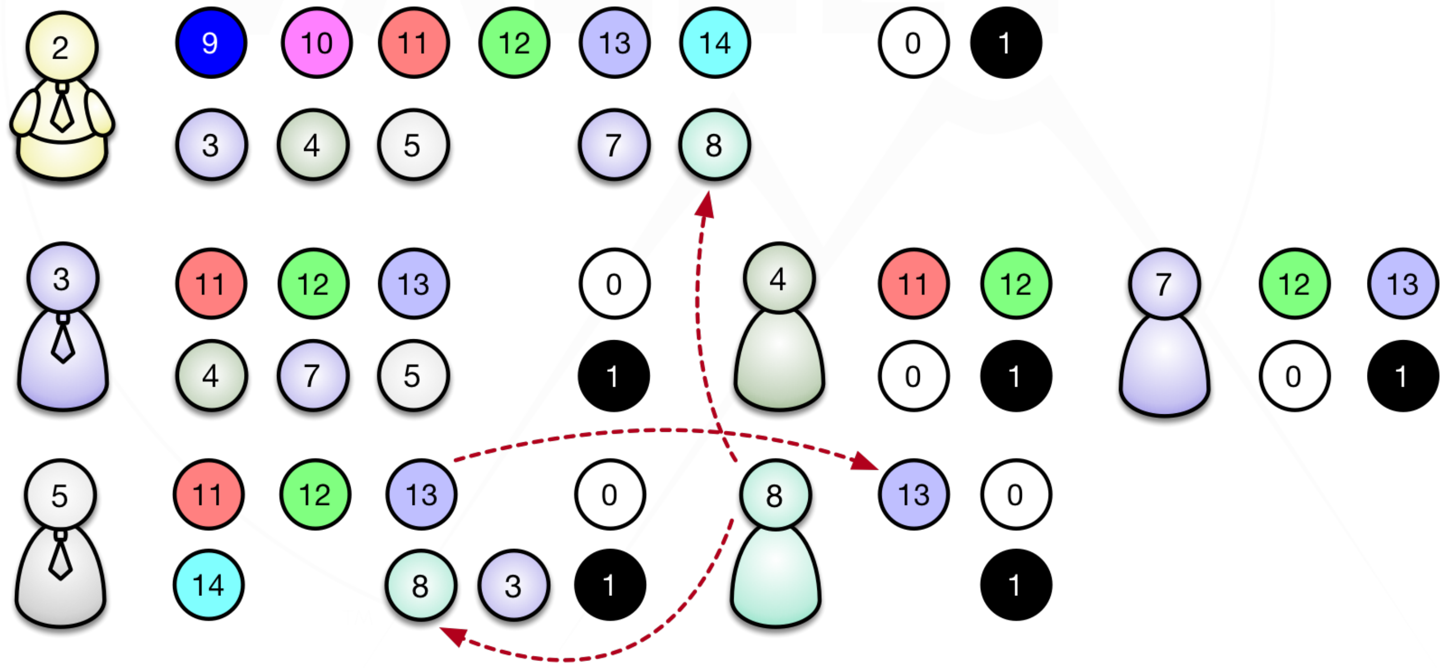
STEP SIX: THE GOD ADMIN GIVES THE NEW MANAGER A SECURITY TOKEN

In this step, the God Admin assigns Token 14 to Manager 5. This means that Manager 5 now has a token that is not available to any other user on the server.



STEP SEVEN: THE NEW MANAGER CREATES A STANDARD USER

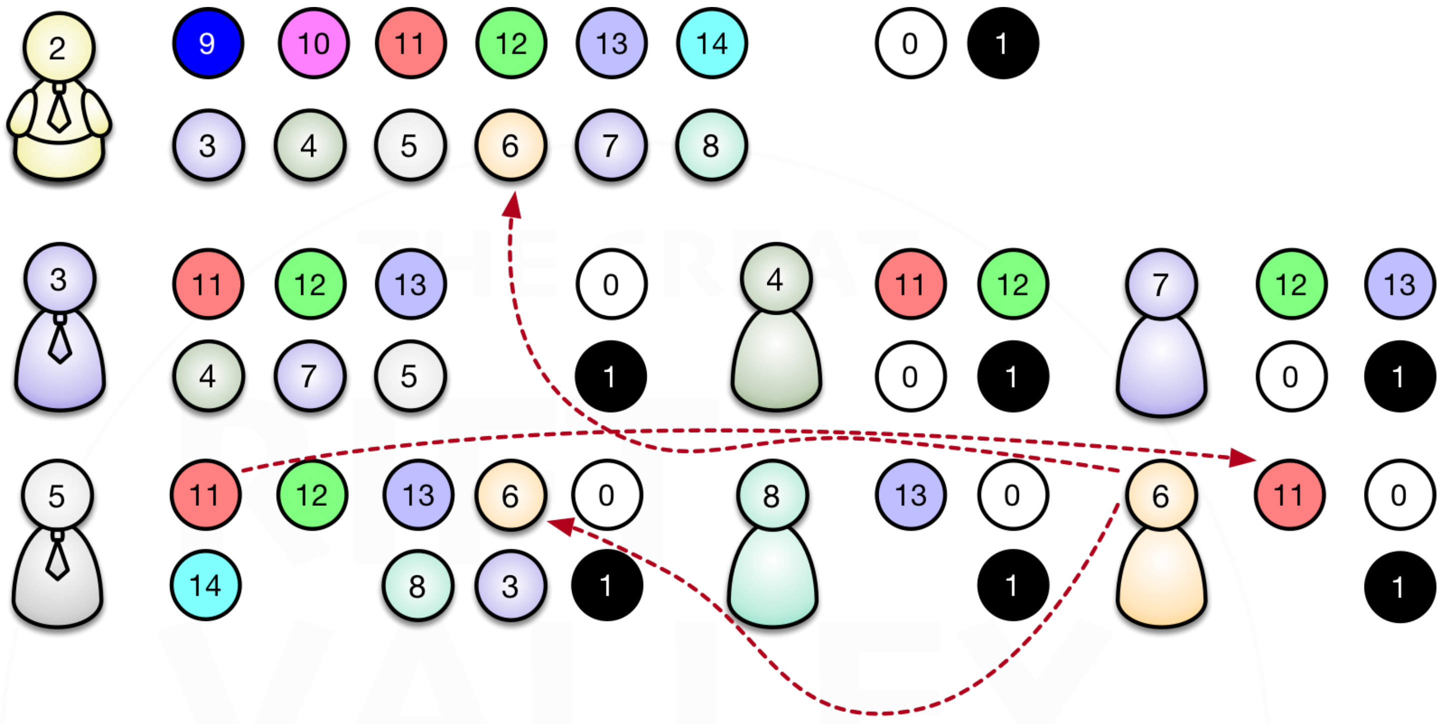
Manager 5 creates User 8, giving it Token 13. Remember that Token 13 was created by Manager 3, but is now available to all users, except User 4.



User 8 is now fairly limited. It can be edited by Manager 5, and the God Admin, but cannot be edited by Manager 3, even though Manager 3 created Manager 5. Also, the only token that User 8 has, in addition to Token 8 (its own ID), is 13, along with the two “intrinsic” tokens (0 and 1).

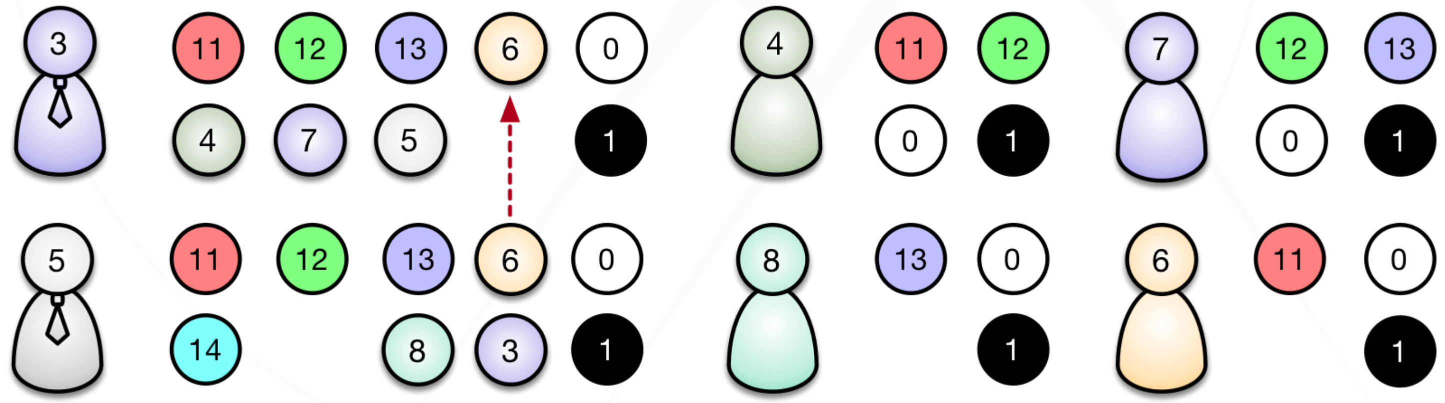
STEP EIGHT: THE MANAGER CREATES ANOTHER USER

At this point, let's say that Manager 5 creates another Standard User (User 6). It supplies User 6 with Token 11. User 6 is registered into Manager 5's token pool, and (of course) is also available to the God Admin:



STEP NINE: MANAGER 5 ALLOWS MANAGER 3 TO EDIT USER 6

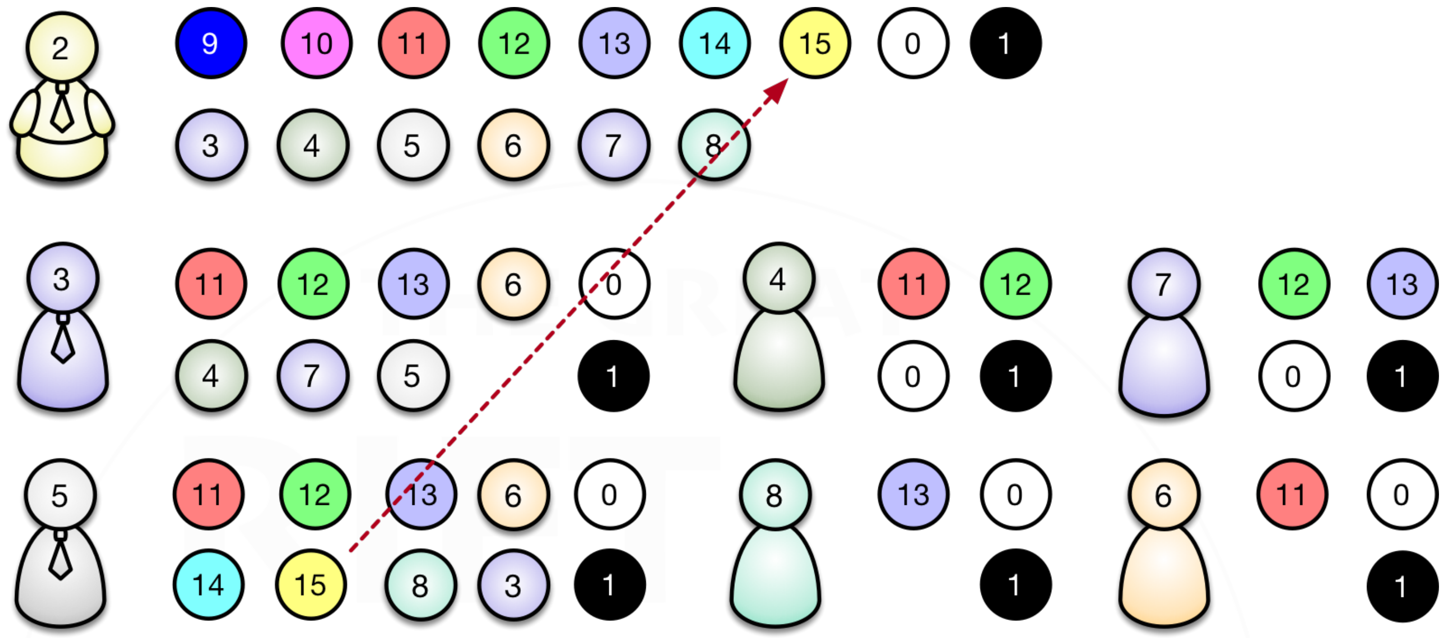
Remember that, in Step 5, Manager 3 gave its ID (Token 3) to Manager 5, when it created Manager 5. This means that Manager 5 can now edit Manager 3. In this step, Manager 5 uses this authority to give Manager 3 edit rights to User 6, by assigning Token 6 to Manager 3's pool:



Before Manager 5 gave the token to Manager 3, Manager 3 could not edit User 6. After the token has been supplied to Manager 3, Manager 3 now has the authority to edit User 6.

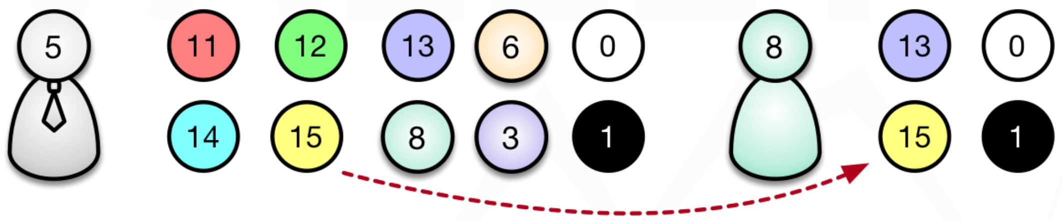
STEP TEN: MANAGER 5 CREATES A NEW TOKEN

In this step, Manager 5 creates Token 15, which is immediately available to the God Admin, but no other users:



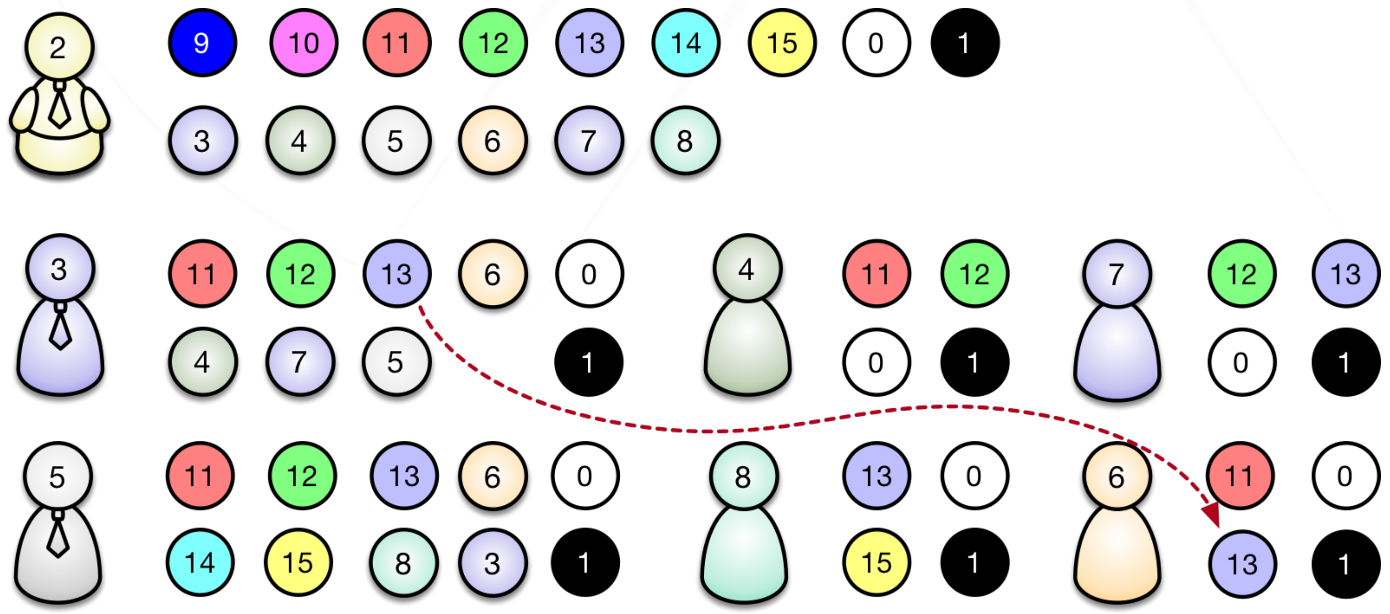
STEP ELEVEN: MANAGER 5 ASSIGNS THE NEW TOKEN TO USER 8

Since Manager 5 has edit rights to User 8 (it is a Manager user, and also has Token 8 in its pool), Manager 5 can assign Token 15 to User 8's token pool:



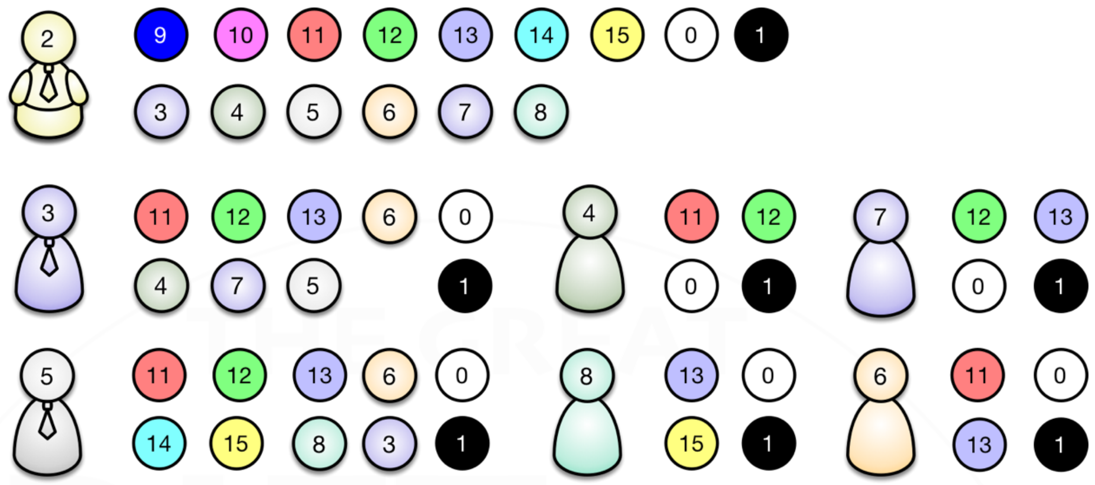
STEP TWELVE: MANGER 3 ASSIGNS TOKEN 13 TO USER 6

Now that Manager 3 has edit rights to User 6, it can give User 6 Token 13:



WHERE WE STAND NOW

At this point, we have seven users, at three different user levels, with a distribution of tokens:



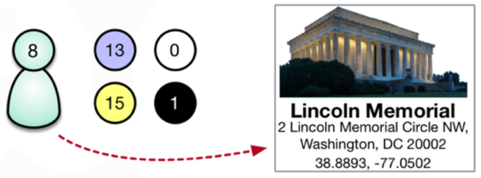
TOKEN ASSIGNMENT EXAMPLE

In the following example, we'll start from the users and tokens above, and show how they are used to secure and control access to various assets.

STEP ONE: USER 8 CREATES A PLACE ASSET

Let's say that User 8, above, creates a "Place" asset. In BAOBAB, a "Place" is a location, usually denoted by a long/lat pair, and an address:

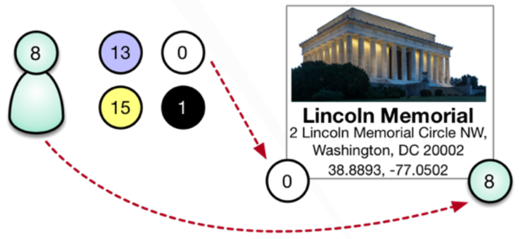
Once the asset has been created, User 8 can then assign tokens to its read and write permissions.



STEP TWO: ASSIGNING READ AND WRITE TOKENS

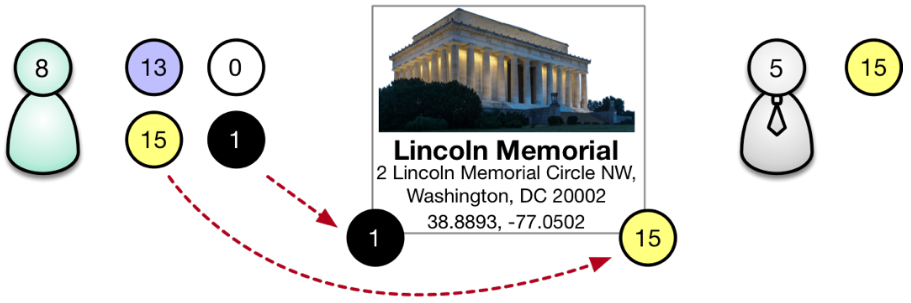
Let's say that the user wants everyone (including non-logged-in users) to be able to see the asset, but only User 8 (and also Manager 5 and the God Admin) can make changes to it.

If we want to do that, we assign 0 as the read token, and 8 as the write token:



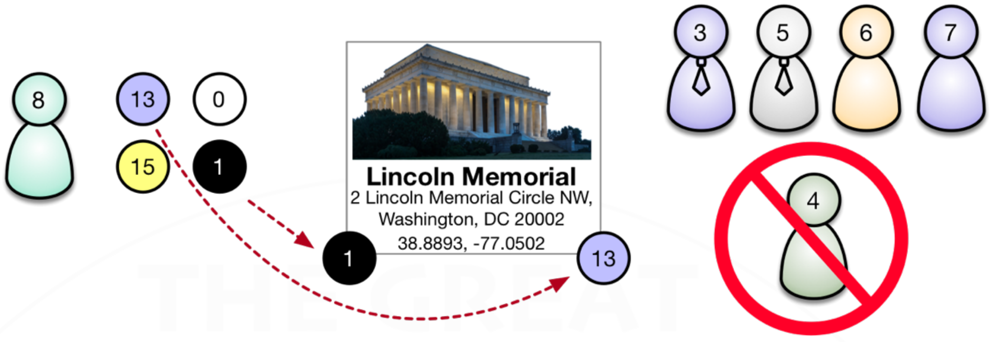
STEP THREE: CHANGING THE VISIBILITY, AND ADDING ANOTHER EDITOR

In this step, User 8 decides to restrict viewing of the location to only logged-in users, so they change the read ID to 1. They also want their Manager to be able to assign others to be able to edit it (NOTE: Even though Manager 5 already has edit rights, User 8 does want to use their own token to enable other users to edit). So User 8 assigns Token 15 to the write ID. This means that Manager 5 can give other users Token 15, and they will be able to edit the asset (Remember that Standard Users can't assign tokens to other users –only Managers). The result is a highly restricted editor permission:



STEP FOUR: MAKING THE ASSET MORE GENERALLY EDITABLE

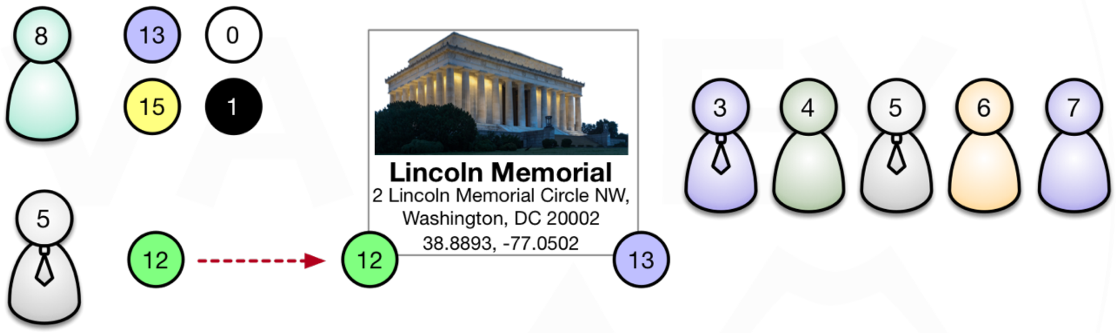
Say that User 8 wants everyone except User 4 editing their location. They can do this by assigning Token 13 to the write ID:



STEP FIVE: A BASIC MISTAKE

Let's say that User 8 wants to allow User 4 to see the asset, but wants to restrict User 6 from seeing it. They might do this by asking that their Manager (Manager 5), apply Token 12 to the asset (remember that User 8 does not have Token 12, so they cannot assign it, themselves).

If User 5 does this, then, instead of User 4 seeing it, and User 6 not seeing it, all users can now see it:



This is because of the write ID still being Token 13. User 4 can now see (but not edit) the asset, but User 6 still has edit rights to the asset (through Token 13), which also automatically confers read permissions.

If User 8 really wanted User 4 to see (but not edit) the asset, and completely hide it from User 6, then none of the existing tokens would work. A new token would need to be created, and applied to the write ID, or Token 13 would need to be removed from User 6.

There is no "blocking" token. Tokens can only *give*, not take.

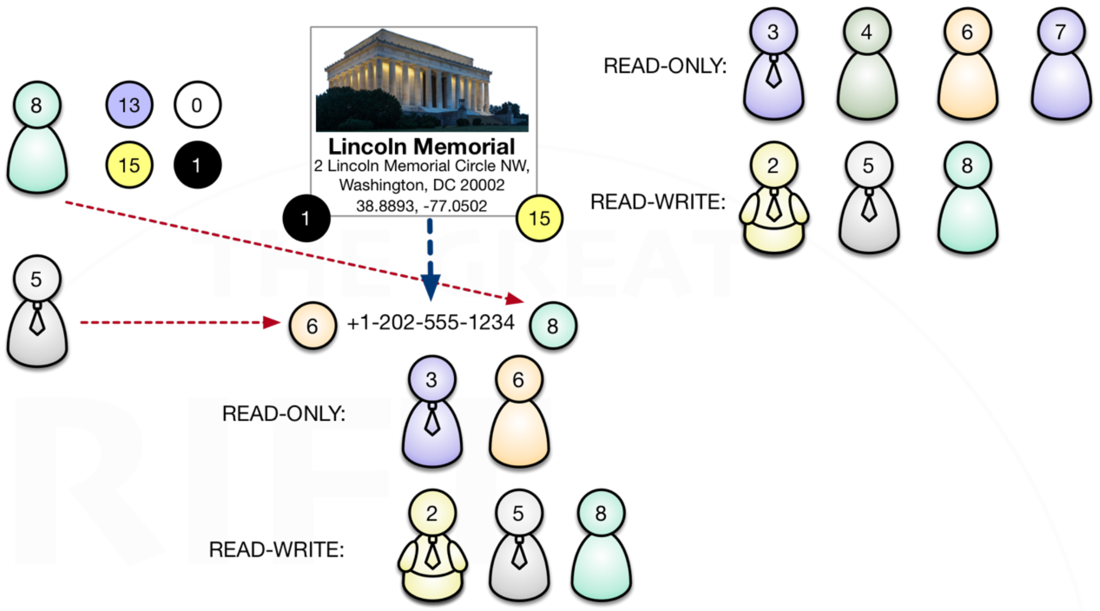
STEP SIX: ATTACHING MORE SECURE ASSETS

It's possible to have a "main" asset with a certain visibility, and attach other assets to it with more restrictive permissions.

Most assets can be expressed as "collections." These are assets that allow other assets to be "attached" to them.

Although the structure is hierarchical, the tokens are not. Each asset's tokens are individual to that asset, and the "hierarchy" of the asset has no effect on the permissions.

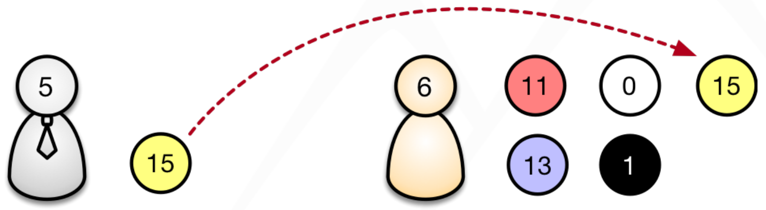
Let's say that User 8 wants the main location to be viewable to all logged-in users, and editable only by themselves, and whomever Manager 5 assigns (the structure after Step Three), but that they want to add a phone number to the location that is only visible to User 6 (and Manager 5). They might do something like ask Manager 5 to assign Token 6 (User 6's ID) to the read ID of the phone number:



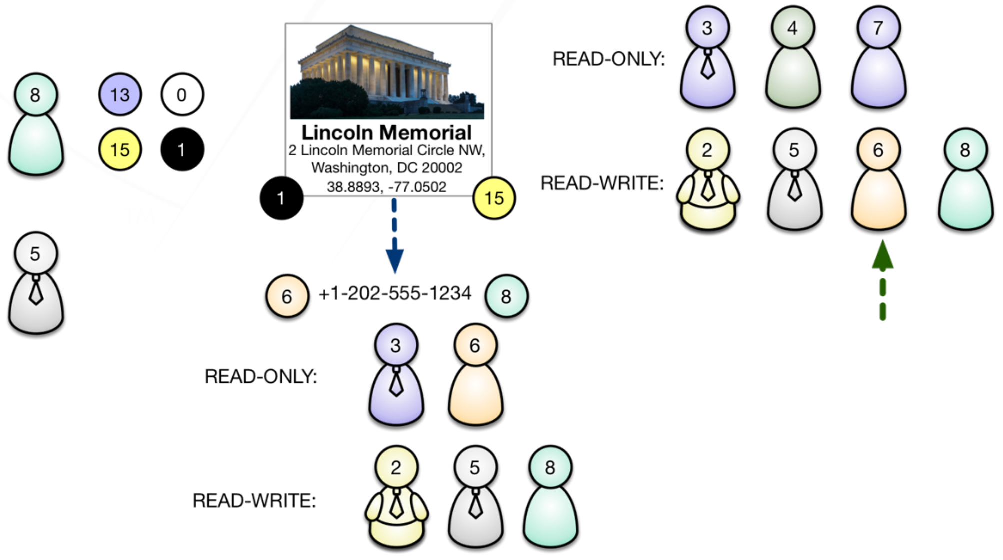
As you can see, the phone number is attached to the asset, as a second asset, but has stricter access requirements. Even though the "read-write" list is the same for the phone number, as it is for the location, the reason is different. The location has Token 15 as the write ID, and the phone number has Token 8 (User 8's ID) as the write token.

STEP SEVEN: CHANGING PERMISSIONS

This means that Manager 5 could assign Token 15 to User 6:



Which would give User 6 read-write access to the location, but not the phone number:



PERSONAL TOKENS

As of version 1.1 of the BAOBAB Server, we now have “personal tokens.”

Personal tokens are special tokens that are private to logins. They can be generated and applied, at the time a Manager creates a new login, but are not even available to that Manager. The only ID, other than the “owner” of the personal tokens, that can see these tokens, is the “God Admin.”

Logins have control of their personal tokens. They can “give” these tokens to other logins, and also “take them back.” Other than that, personal tokens behave exactly like regular tokens. If a login applies a personal token to an asset, they then have complete control of the visibility/modification rights of these assets.

THAT’S PRETTY DAMN COMPLICATED!

Yes, it is. However, it is also extremely powerful, and allows users a very fine-grained control of the security of their assets.

OTHER SECURITY CONSIDERATIONS

LOCATION OBFUSCATION

Location obfuscation is another “classic” security policy. It requires that an asset have an assigned location (a longitude and a latitude). You can then set a “fuzz factor” to that location, which is a number of kilometers.

If a location is “fuzzy,” then any request for its location will return a random long/lat, within a box surrounding the real location.

This is a standard methodology for hiding exact locations (for example, the exact locations of women’s shelters are often confidential), so you know it’s nearby, but not exactly where.

You can set tokens that allow certain logins the ability to view the correct location.

TECHNICAL ARCHITECTURE

DATABASE-LEVEL FEATURES

NOTE: *Most of this is implemented in the BADGER Hardened Database Layer. This is the lowest layer of the BAOBAB Server.*

DUAL DATABASES

One of the most important aspects of security in the BAOBAB Server, is the fact that it uses two databases: Security and Data.

They are completely independent of each other, and can even use different database technologies (BAOBAB comes with built-in support for both Postgres and MySQL).

It is possible to have the Security database “hardened,” and possibly monitored, while the main database is more performance- or capacity-oriented.

QUERY-LEVEL TOKEN SECURITY

In the above section, we went over security tokens. The BAOBAB Server applies token-level security at the level of the SQL query. What this means, is that the tokens are checked by the database, not the server code, and won’t even make it into the server, unless they pass.

Each database query has what is called a “Security Predicate,” which is an SQL statement header that asks the database server to examine the tokens of an asset, comparing it against the current logged-in user’s token pool, before executing the rest of the query.

PHP PDO TRANSACTIONS AND PREPARED STATEMENTS

PDO stands for "PHP Data Objects," and is a PHP language standard library feature that allows us to access different database technologies, and to access the database by using something called "Prepared Statements." Prepared statements allow PDO to "sanitize" database queries, to avoid SQL injection attacks. We also use Transactions, which allow all SQL interactions to occur as "bundled sets." The entire transaction, which may actually be a fairly intense set of SQL commands, is treated as an "atomic," unit, and will not affect the database unless ALL of the statement succeeds. This is another way to avoid SQL injection attacks, as well as avoiding some kinds of bugs, like simultaneous access problems.

OTHER FEATURES

MANAGERS CANNOT CHANGE THEMSELVES

No Manager (even the God Admin) can affect changes to their own permissions. They cannot change their token pool, read/write, etc. All changes must be applied by other Manager logins with write access to the Manager under change.

SINGLE "GOD" ADMIN

Like in "The Highlander," There Can Only Be One. Any of the login IDs (they do not have to be Managers) can be "promoted" to the "God Admin." This is done in the config file.

The God Admin password is also supplied in the config file, and the one set for the ID is ignored.

The God Admin has full rights to everything on the server (with the exception of its own tokens).

GOD CANNOT BE CHANGED BY ANY OTHER MANAGER

The God Admin is even more restricted than other users. It can only change itself (and remember that it can't change its own token pool -which isn't relevant, anyway).

This means that a God Admin cannot use its tenure to "elevate its login," which would apply after it is no longer God Admin; so it can't create another Manager, edit its token pool, or convert its Standard Login to a Manager Login, and then delete the modifying Manager. It has its power only as long as it has been given that power by the config file.

GOD IS LOCKED INTO THE SERVER AT THE CODE LEVEL, NOT THE DATABASE LEVEL

The God Admin cannot be transferred at runtime. The reason that it is assigned in the config file, is so that it is "baked into" the server. Also, the password being in the config file, means that the Server Admin can lock out the God Admin, if necessary.

This also means that the God Admin can't create another user, then transfer itself to that user as God Admin (but it can create a powerful user during its tenure, and that should be considered at the policy –not technical– level).

LOGGING

BAOBAB has support for multiple levels of transaction logging. These are off by default, but can be applied at the API level, or even the database level (really should only be used for debugging).

EXTERNAL LOGIN VETTING

BAOBAB has a "hook," applied at the config file level, for a "login authentication." This could be a function that sends the login to an external authentication authority, and it can override, or enhance, the internal, database-based login authentication.

SSL REQUIREMENT

It is possible to require that all transactions occur through SSL, or even just the login. This is something that is established in the config file.

LAYERED ARCHITECTURE

BAOBAB has a layered architecture, with each layer having a strict functional domain. The ANDISOL layer represents the layer at which all database technology is hidden behind a programmatic façade. It is entirely possible to "swap out" the database technology at that point.

API KEY-BASED AUTHENTICATION

When a user logs in, the login ID and password are sent in in a "standalone" call. No other functionality is allowed in the login call, and the login credentials will not be accepted at any other time.

The server responds with a temporary "API key," which is a randomized, unique (on the server) text token. This token has a finite lifetime, which can be set in the config file. The God Admin has a separate token and lifetime from other users (also configurable).

If the API key passes its lifetime, the login will be considered invalid, and a new login (login ID/password) will be required to obtain a new key.

The server has what's called a "server secret." This is a unique string that is specified in the config file, and must be presented to the server, along with the API key, in order to authenticate the transaction. If either is not valid, the authentication will fail.

This server secret is supplied outside of the server runtime, by the Server Admin, to the entity that wants access to the server. There is no provision in BAOBAB to provide this secret, or change it, other than manually. If someone wants to access the BAOBAB API, they must contact the Server Admin, and request that the server secret be sent to them.

Only one API key can be issued per user. It is possible to configure the server to refuse a new login, if the user already has a valid API key. If the server is not configured to reject redundant logins, a new login will immediately terminate any other login for that user (meaning the API key for that other login will no longer be valid).

BASALT PLUGINS

The API is implemented at the BASALT (top) layer, and there is a plugin architecture. It is possible to write new plugins that may implement even more stringent security policies or API technologies.

CONCLUSION

BAOBAB was written "from the ground up" to be as secure as possible. We can't guarantee that it's "unhackable," and a lot still depends on the policies and process applied by the people running the server.

That said, we're still fairly confident that it's a "tough nut to crack."

All locations can be "obfuscated." This means that they can be set to return randomized "nearby" locations. It is possible to allow certain users to "see past the fuzz," to the true location.

There are two main layers of user: Standard User, and Manager. Managers have the ability to create users, edit users (if authorized), and create tokens.

One (and only one) user can be elevated in the code (not at runtime) to be a "God Admin." This is a superadmin account that has complete runtime authority.

Users have a fairly complex, granular, token-based, toolkit for securing assets, at their disposal.

Each user has a "pool" of tokens, bestowed by Managers. These pools denote the privileges the user has.

Tokens are forever. Once a token has been created, it cannot be destroyed.

User IDs are intrinsic tokens. If a user login is deleted (by a Manager), its ID is converted into a simple token. If it is deleted, its token "pool" is removed (as opposed to being transferred).

There are three "built in" tokens: 0 (all visitors, including unauthenticated visitors), 1 (all authenticated users), and -1 (only the God Admin).

Unauthenticated users can never write to any assets; even if they are allowed to see the asset (Token 0).

There are no "better" tokens. Each token has exactly the same "authority" as any other token, and that authority is determined by the way the token is applied.

Every asset on the BAOBAB Server has a read token and a write token. These denote privileges that various authenticated users may have, for that asset.

There is no "privilege hierarchy." Each asset is atomic in its read/write token. Just because one asset "contains" another asset, does not mean that privileges applied to the "container" asset, also apply to the "contained" asset.

There are a number of technical measures, taken by BAOBAB, to secure the server.

The server uses a stateless REST API, employing random, unique API keys, generated at login time.

Logins are an "atomic" operation. They cannot be combined with any other operation, so there is only a need to send in login credentials once. All other operations require the returned API key.

There is a "server secret," that must be presented, along with an API key, in order to authenticate any transaction. This secret is managed outside the runtime.

BAOBAB uses two separate databases: one for security stuff (logins and tokens), and another for data. These can be hosted separately, and even use different technologies.

There is provision for external authentication (i.e. OAuth or some other validation mechanism).

There is provision for logging.

There is a layered architecture, and there are plugin APIs available for extension. Additionally, the entire database substructure could be replaced at the ANDISOL layer.