Chatbox Engine End User Manual

Valid for: 1.4.5 (but heavily revised)

CELEBRATING ONE YEAR:

[$ git clone https://github.com/4inches-usbstick/chatboxengine.git]

Table of Contents:

(pages are marked by their page number in the corner and NOT by what number your PDF reader or equivalent program indicates.)

Sections 1 through 5 cover things for end users and 6-8 cover things for advanced users and people who choose to mod or write clients for Chatbox Engine.

1 Basics and Setup

CONCEPT:

In any given directory in which a Chatbox Engine instance exists, there are text-encoded, binary, HTML or special data-formatted files that are readable to everyone. These files are called Chatboxes. There’s also the ability to write to these chatboxes and to say who can and cannot write to a certain Chatbox.

Anyone can create them, unless specifically configured against it.

Attached to HTML and text-encoded chatboxes are file upload directories (that can be disabled)

Finally, there are some abilities open to administrators and authorized users only, such as:

- Editing text
- Managing the actual server
- Moving, deleting and creating chatboxes / directories.

Chatbox Engine has been used in a lot of applications, from IRC-like communication tools to remote desktop utilities to voice calling and messaging. It all depends on how the server is configured and how people use it.

By default, it comes with a client (inchat.php/inchat-div.php) that is accessible by joinchatpublic.php.

SPECS:

- Advised to have PHP 7+
- Must have > PHP 5
- For Python wrappers, use the most recent versions of Python and its libraries.
- Do not need root to install unless your directory is protected locally.
- Need text editor and web browser for testing.

SETUP:

When you first get Chatbox Engine, it won't work. You have to set it up. First, create a 'textengine' (can be any name) directory in any subdirectory or at the root. Then, create a subdirectory called 'sitechats' (or any other valid name).

Any Chatbox Engine instance needs two folders because some clip-ons work out of the 'textengine' parent directory, so to maintain interoperability, you have to create two directories.

Now that the directories are created, unzip the ZIP or use a VCS or just copy the files over. You can delete the 'pythonfiles', it won't be of any use to you as a server operator.

Now, take note of the filepath of the file named .htamainpolicy - this is the master configuration file for all of the server. There are a few places that need this path:

1. Open /textengine/sitechats/data/mainlookup.php and then replace line 2's file path with the path you copied (absolute path only, no relative or symlink).
2. Do the same to /textengine/sitechats/media/mainlookup.php
3. These locations are the only hard-coded areas left.

Note that we use textengine as the parent dir and sitechats as the dir where everything is. Your names can be different.

Now, open .htamainpolicy in a text editor: it's config file time.

POLICY ID	PURPOSE	TYPE
1	Server domain name or IP address	Hostname
3	Textengine path (parent directory path, actual dir does not have to be named textengine)	Filepath
5	Offshore path (where Chatboxes are offloaded from the main area)	Filepath
7	Media offshore path (where Media Dirs are offloaded from the media directory)	Filepath
9	Server timezone	Timezone
11	Max file upload (does not bypass the limit imposed by your hosting provider / server) (in bytes)	Integer
13	Whether to allow media upload across the server.	Bool
15	Deprecated, should be NO	Bool
17	Whether to attach timestamps to file uploads' names	Bool
19	Whether or not to use the old inchat_joinpage system (set this to NO)	Bool
21	Whether or not the API is enabled (excludes terminal commands) (set this to YES)	Bool
23	Whether or not timestamps can be enabled using rule.Timestamps(1)	Bool
25	Banhammer command file (.htaccess or similar)	Filepath
27	Banhammer text (deny from or similar) (use %ip in place of the address to ban)	String
29	Phrases that cannot be in messages (non case sensitive) (no regexp) (delimiter = //)	Delimited String
31	Illegal file extensions for the sendmsg endpoint (delimiter = //)	Delimited String
33	Whether or not to do encoding conversion (encoder parameter related) (set this to YES)	Bool
35	Whether or not the onboard config editing utility is allowed to run	Bool
37	Terminal errors (0, E_ALL)	Choice

	(set to 0 in a production environment)	
39	Default client: whether or not to check if a Chatbox has a media dir (YES for assume yes, NO for assume no, CHECK to check every time)	Choice

41	Whether or not the editing endpoint requires the admin password	Bool
43	Whether to allow the HTTP POST sendmsg endpoint	Bool
45	Terminal logging file	Filepath
47	Whether to do terminal logging	Bool
49	Commands that don’t get logged (delimiter = //)	Delimited String
51	Whether or not the udb manager needs admin	Bool
53	Allow anon. newchat endpoint	Bool
55	Whether the newchat command needs admin	Bool
57	CBEDATA requiring a “begin CBEDATA” header	Bool
59	HTTP/HTTPS for URLs	Choice
61	Whether autoencoding ‘&’ in the clients with JS is enabled	Bool
63	Whether WILDCARD-ALL is an acceptable chatbox value in the editing endpoint	Bool
65	Whether CCMD is enabled (if this is NO then the next three must also be NO)	Bool
67	Pre-Exec CCMD enable	Bool
69	Mid-Exec CCMD enable	Bool
71	Post-Exec CCMD enable	Bool
73	Whether lock add and lock del commands require admin	Bool
75	Whether lock add has any effect	Bool
77	Allow CSEND command (set this to YES)	Bool

79	Timestamps structure for ts+name+msg	String
81	Timestamps structure for ts+msg	String

83	Timestamps structure for name+msg	String
85	Timestamps structure for msg only NOTE (for PIDs 79-85) %time = time %date = date %name = name %mess = message These values are replaced in the message, so for example, %name-> %message becomes “Namer-> hi everyone”	String
87	Max size for a message (in bytes or chars depending on 89)	Integer
89	Whether to use bytes or chars to count message size (byte or char)	Choice
91	Max size for a message that is POSTed to the server using the sendmsg_post endpoint (in bytes or chars depending on 89)	Integer
93	Recovery chatbox to enter your recovery password into when you run the command	Chatbox Name
95	The recovery password	String
97	Whether the password recovery tool is enabled	Bool
99	Recovery time (the amount of time you have to enter the password into the Chatbox once you run the command in seconds) To use this utility, run inirecovery command, then open the recovery Chatbox and enter the recovery password. The terminal endpoint will stall until the password is entered in or time runs out, after which the current password (the forgotten one) is returned.	Integer
101	Commands that require admin (delimiter = //) If the command doesn’t require any authentication then you can’t put it here, it just won’t work.	Delimited String
103	The PHP date format for timestamps	PHP formatted date
105	The PHP time format for timestamps	PHP formatted timestamp

107	The name of the sitechats directory (the child directory that everything is in) (NOT THE PATH THE NAME OF THE SUBDIR)	Directory Name
109	CBEDATA path delimiter	Single char
111	Terminal logging format: “Terminal [%time %date]: command=%cmd %params , user=%useduid:%uid , method=%verb)” is the default, try it.	String
113	When a password is entered and sent off for handling by another script (for example: hash.php), the location of that script is assumed to be the Filepath + “hash.php”. For example: “/www/textengine/sitechats/hash” + “hash.php” = “/www/textengine/sitechats/hash/hash.php”	Filepath

You’re not done yet. Open .htaterminalaccess and write the names of any Chatboxes you don’t want deleted by the terminal in there, delimited by the \r\n character.

Then, open .htabannednumbers and change the names of any Chatboxes that you don’t want created by the public. There’s a lot of ridiculous ones in there inherited from the early days, so just delete the ones you don’t want to be protected.

Finally, delete the README. Nobody needs that.

Congrats, your server is ready for its first live test. In a web browser, open joinchatpublic.php and try to join terminal-logger or your terminal logging Chatbox, then try to send a message. It should be blocked from sending; this is what you want. (make sure to protect your terminal logging Chatbox; this process is described in a future section)

Next, create a file named test.txt in sitechats, and try to join that and send a hello message. It should go through and you should see it in the client.

Now, onto the config files that apply per media directory. In a recent update a filename filter was added where filenames had to have a certain pattern or could not have a certain pattern (i.e. no .exe files). In /textengine/sitechats/media/cbname/.htafiletxpolicy, you can set those parameters. Creating this file can be done manually (through SSH for instance). In beta versions, this was the only way. In release versions however, a terminal command would be available. The syntax is pretty simple:

mode:FILTERMODE
This makes it so that most files are allowed and certain keyphrases invalidate the upload.

mode:GATEMODE

This makes it so that most files are invalidated and only ones that meet requirements pass.

pattern:.extension
This does one of two things:

1. If the mode is GATE, it allows the upload to pass as it meets a requirement.
2. If the mode is FILTER, it invalidates the upload because it contains a no-fly phrase.

lock:YES
Locks the media dir entirely.

unlock:YES
Reverts lock:YES.

** If there is more than one instance of lock or unlock: the last instance is taken. For instance, lock, lock, unlock and lock results in a lockdown because lock is the last command.

Also note that if this file is not present, the file upload automatically passes as there is nothing to say otherwise.

2 CBauth, uGroups, Filesafe, Readsafe, Locks, CCMD and C_Policy.

2.0 Functionality
All of these authentication and policy features are used in each file individually. For example, the terminal doesn't need CCMD, locks or FILESAFE, and therefore, no checks are done in that file.

sendmsg_integration.php requires all of these, so it does checks on all of them.

Your only helper will be a PHP includable script that contains callable functions to get certain details out of each section.

Also, things like syntax helpers such as 0::1::2::3; serve a programmatic purpose and thus should not be removed.

When something is a section, we mean this:

```
[BEGIN label_name]
…
[END label_name]
```

This would be the section labeled label_name in the config file.

> ⚠️: ***Because each individual script is responsible for defining how a section should be read and handling any sensitive information, security rests at the level sensitive information is handled at, not at any underlying include-able libraries.***
>
> ***Utmost caution must be used in every script that uses CBauth calls and whatnot to make sure that that individual point is not breakable.***

2.1 CBauth

CBauth, also known as the UID/UKEY system, is Chatbox Engine’s system of authentication and permissions. The new uGroups feature works on top of UID/UKEY, which is how permissions and grouping work.

To begin, let’s have a look at the two areas responsible for this:

```
[BEGIN CBAUTH]
D1::RootOfficial::root1234::sudousers//cantrunhelp;
D2::UserOfficial::root5678::notsudousers;
0::1::2::3;
[END CBAUTH]

[BEGIN GROUPS]
sudousers give sudo
cantrunhelp cantrun help
[END GROUPS]
```

In the CBAUTH section we have the users and their groups, and in the GROUPS section we have the permissions per group. In more detail: a user is defined as

UID::SCREEN_ALIAS::UKEY::GROUP1//GROUP2;

We have these elements in a user in this order, delimited by ::

- UID (the username of sorts)
- Screen Alias (the on screen name that nobody else can have)
- UKEY (the password)
- User Groups (delimited by //) (which groups a user belongs to)
- Semicolon + \n (VERY IMPORTANT)
- *It is strictly forbidden to have a space in the UIDUKEY area.*

In the GROUPS section there are two commands:

GROUPNAME give sudo
GROUPNAME cantrun (command_name || WILDCARD-ALL)
GROUPNAME canrun command_name

Groups do NOT have to be defined unless you want to set perms on them. When you give sudo to a user group, you give that entire group certain admin powers. When you cantrun a certain group, you say that that group can't run a certain command (or all commands).

If you've banned a group from all commands, you can also allow individual commands with canrun.

If a user is a part of multiple groups:

- the group that appears FIRST takes precedent, regardless of whether it is a CAN or CANT.
- if a user is in groups A, B and C, and B can't run help but A is expressly granted perms, A takes hold. If it were B, A and then C, B would take hold and stop execution.

- when accessing sendmsg/display the user has to be part of one or more allowed group to be considered OK to write/read to chatbox.
- the editing terminal does NOT have this feature. all sudo users can edit OR only masterkey access.

In Chatbox Engine, there are four levels of authentication:

- Ordinary user (incorrect, invalid or missing logins)
- Registered user (correct logins but no additional perms besides screen name)
- Sudo user (correct logins and additional perms)
- Admin/localhost/root user (has local access and has access to all commands)

(a protected name is one that nobody else can use).

Sudo and admin users have the capability to modify the server beyond chatboxes. *If a user has multiple groups and one is sudo powered, that user becomes sudo powered.*

The admin user is only accessed by using the master password, which is located in .htapassword

These permission levels and user groups are used in these situations:

- Protected chatboxes and deciding whether or not to let a user send a message to it
- Running most administrator commands
- Editing chatboxes
- Making changes to any critical files

There are some PIDs related to this system of perms:

ID	Type	Notes
41	Bool	If set to YES, the editing endpoint requires the master password instead of just any valid sudo user.
51	Bool	If set to YES, managing the UID/UKEY system will require the master password.
55	Bool	If set to NO, the newchat endpoint is disabled and users are forced to use the terminal endpoint.
73	Bool	If set to NO, the lock add and lock del commands will require the master password.
101	Delimited String	cmd1//cmd2: for every item, delimited by //, that command will require the master password to use, regardless of any other settings or policies. Note that this policy doesn't work on commands that require no authentication in the first place, such as help.

These commands are related to the UID/UKEY system:

lock add - lock out a user or group
lock del - revert lock del
uid add - add a user account
uid del - revert uid add
group add - add a group command
group del - revert any group add
filesafe add - add a filesafe command
filesafe del - delete a filesafe command
readsafe add - add a readsafe command
readsafe del - delete a readsafe command

The following endpoints are related to the UID/UKEY system:

terminalprocess.php - some commands don't run under nonsudo or nonroot
adminedits.php - edits require sudo and sometimes root
admineditsreverse.php - same rationale as adminedits.php
sendmsg_integration.php + variations - some chatboxes are protected and may require registered/sudo/auth

Note how media upload endpoints are not affected by CBauth.

2.1.2 Password Hashing

Added in a recent update was password hashing. Instead of storing passwords as plain text, it is now possible to store them as hashes (with or without salt). Generating salts however is not our problem.

In the password section, instead of a plain text password, use the following syntax:

hash-handler-scriptname//<config information>

It is possible to write custom handlers, but there is one included: "hash.php". To use it:

hash//<salt>%%password=<hash>

The user account "D1" (the default user account number 1) has this in the password slot:

hash//afAGJTYpxFQOPlXwlyNUMQ07NQlBvM1FoFHZfKBP+%%password=ccae0e7e74bcf86466c097490b24b9da02e78dddd9bfb7e9fd9b024633f3efa5642a4aa3879abc68dbe4b140b6a2da86732623deb91f3e473b77768faab66e94

The salt is "***afAGJTYpxFQOPlXwlyNUMQ07NQlBvM1FoFHZfKBP+***" and the hash stored is "***ccae0e7e74bcf864…***". The password for this default account is ***root1234***. If we append ***root1234*** to our salt, we get "***afAGJTYpxFQOPlXwlyNUMQ07NQlBvM1FoFHZfKBP+root1234***".and take the SHA-512 hash, we get "ccae0e7e74bcf864…".

If the salt or password changes, the resulting hash will also change.

2.2 Filesafe/Readsafe

In some cases it may be desirable to require a login in order to write to a Chatbox, or to lock it off entirely in some cases. This is where Filesafe comes in. The section for Filesafe in .htamainpolicy is labeled FILESAFE.

This is different from the locks function; filesafe denies by default and then allow; locks allow by default and then deny.

These are the commands:

chatbox_name::auth_level[login/sudo/local]::
chatbox_name::g:group_name::

The first one takes the Chatbox name and the auth level.
The auth levels in FILESAFE are:

- login: the user must present any valid UID/UKEY pair
- sudo: like login, but the UID/UKEY pair must be associated with a sudo group.
- local: all write perms are disabled for everyone

The second one takes the Chatbox name and the group name, with the prefix "g:"
If you want to allow multiple groups use the // delimiter like such:

g:group1//group2

This section should not contain any spaces. If there is a g: prefix, all other groups won't be allowed and only the specified ones will go through.

It's best practice to only use ONE command per chatbox but in theory nothing is stopping you from adding multiple.

No policies in .htamainpolicy influence the functionality of FILESAFE, however, there are these commands:

filesafe add - add a command
filesafe del - revert filesafe add command

These rules all apply to the readsafe area as well which prevents unauthorized reading. It's literally the same thing but instead of halting a write you halt a read operation instead. Also, local level protections on reading means that nobody can read the file which may be useful sometimes. The associated terminal commands are:

readsafe add - add a readsafe command
readsafe del - delete a readsafe command

⚠️: ***Readsafe doesn't protect from direct access, for example, http://localhost:8000/textengine/sitechats/2884 [where 2884 is read protected to the LOCAL level].***

Readsafe only impacts the display.php endpoint.

2.3 Locks

In some cases you have something against a specific UID writing to a chatbox. The command for this is simple:

chatbox_no deny from UID
WILDCARD-ALL deny from UID

To lock a user out of a single chatbox, use:

- chatbox
- “deny from”
- UID (not screenname)
- *There is no delimiter (like ;) in this section.*

The following policies influence the locks functionality:

ID	Type	Notes
73	Bool	If set to NO, the lock add and lock del commands will require the master password.
75	Bool	If set to NO, the locks will not be checked on message send, rendering this section ineffective.

These commands influence locks:

lock add - add a lock command
lock del - revert lock add command

2.4 CPolicy

Any custom config that doesn’t fall anywhere else can be put into CPolicy. There is a helper function for developers to retrieve this section.

This section can contain anything you want.

No policies or commands affect CPolicy. The section is labeled CPOLICY.

2.5 CCMD

If you've ever made or touched or looked at a Discord bot or its documentation there's something called an autoresponder which executes some form of program on a certain pattern in a message (i.e. IF keyword "cheese" IN message THEN execute: blankity(blank,blank);)

Chatbox Engine has the same concept starting after a recent update.

The CCMD system has the label CCMD in the config file. No extra \r\n characters are allowed anywhere.

A command in CCMD is structured like this:

@Pre::BEGINSWITH::/example::mainlookup.php;

You have:

- The execution point (which point during sendmsg execution does the code run)

Value	Notes
@Pre	Before execution of most code.
@Mid	After authentication related code runs but before the message is written to disk
@POST [for PoST]	After most code has executed, including writing a message to hard disk.

- Condition

Value	Notes
BEGINSWITH	If the substr is at the start of the message content
HAS	If the substr is anywhere within the message content

	(NO regexp allowed).

- Substring (what the sendmsg_endpoint actually searches for
- Include path (which PHP file to include)

As an example:

IF keyword "69" IN message THEN include nice.php AT <MIDPOINT>
Would become
@Mid::HAS::69::nice.php;

In sendmsg_integration and friends there is a callable function that lets you check for executable conditions called sendcmd(). More on this later

2.6 Summary

Section Name	Case Sensitive?	\r\n sensitive?	Delimited?	Purpose
GROUPS	✅	❌	❌	Define permissions for a certain group
CBAUTH	✅	✅	Semicolon between commands	Define user accounts and the groups they belong to
FILESAFE	✅	✅	:: between commands	Only permit certain users or groups to write to a chatbox
UIDUKEY LOCKOUT	✅	❌	❌	Lock certain users out of one or all chatboxes
CCMD	✅	✅	Semicolon between commands	Message-based auto triggers? Idk how to describe this in a short way.
CPOLICY	⚠️	⚠️	⚠️	Custom config space. The sensitivity to newlines and whatnot is determine by the external script that makes use of the custom config.

3 Application Programming Interface (API)

3.1 Endpoints
Chatbox Engine has a set of endpoints that can be GETted or POSTed to make changes to the server.

<> = required
[] = optional
{} = auth

Location	POST GET	Parameters	Purpose	Return Value	Remarks
sendmsg_integration.php	GET	Message: <msg> Destination: <write> Name: [namer] Encoder: [encoder] {uid} {ukey}	Write message to hard disk.	Debug info.	[encoder] can be set to 'none' to avoid automatically encoding in UTF-8 using a protected name in [namer] or accessing a protected chatbox may require authentication %nl can be used in place of the \n character in

					message content. These notes apply to all endpoints with sendmsg in the name.
sendmsg_integration_nobreak.php	GET	Message: <msg> Destination: <write> Name: [namer] Encoder: [encoder] {uid} {ukey}	Write message to hard disk without adding newline automatically.	Debug info.	Read above
sendmsg_integration_nobreak_post.php	POST	Message: <msg> Destination: <write> Name: [namer] Encoder: [encoder] {uid} {ukey}	Write message to hard disk without adding newline automatically. For when the GET url exceeds the limit of the server	Debug info.	Read above. Also: Character limits are increased by default. No newline is added automatically. This endpoint is disableable by PID 43.
sendmsg4html.php					This endpoint has been retired and will throw a warning any time it is used.
newchat_integration.php	GET	New Chatbox Name: <newname> Allow Media	Create new chatbox without using COPEN command	Debug info.	The filename cannot be an illegal filename, which are

		Upload: <allowmed> CB Type: <option>			names that contain these chars: array('<', '>', ':', '"', '/', '\\', '|', '?', '*', ';', 'NUL', 'COM', 'LPT', 'CON', 'PRN', '&'); Allowmed must be allowmed or forbidmed, no exceptions. CB type is: l for regular h for HTML d for CBEDATA This endpoint can be disabled by PID 53. Any media directories created with option H will be named chatbox-med instead of chatbox.html In .htabannednumbers, the list of filenames dictates forbidden filenames, along with the already existing checks.

display.php	GET	Filename: <chatbox> Encode bytes: [encode] Password: {uid} {ukey} (beginning 1.4.5)	Get Chatbox contents	Chatbox Contents as bytes OR debug info on error	Encode is by default UTF-8, use 'none' if you want to get raw bytes.
terminalprocess.php	GET	Command: <cmd> Parameters: [params] Password: {pass} {uid} {ukey}	Server management commands.	Depends on the command being executed.	This is a prime example of the UID/UKEY auth level system. Don't use the master password and uid/ukey at the same time. Parameters are not a thing in all commands.
adminedits.php	GET	Destination: <cb> Find str: <gro> Replace str: <rw> Password: {key} {uid} {ukey}	Overwrite all instances of <gro> with <rw> in <cb>	Debug info.	The webpage named "admineditsutil.php" accesses this endpoint. Don't use UID/UKEY and master key at once. This endpoint can be set to reject all UID/UKEY by

					PID 41. This endpoint can edit all Chatboxes by using WILDCARD-ALL as the <cb> if PID 63 is set to yes. **Ignore any auth errors that comes out unless it's the only thing that comes out; there's been a long standing issue with that.**
admineditsreverse.php	GET	Destination: <cb> Find str: <gro> Replace str: <rw> Replace index: <index> Password: {key} {uid} {ukey}	Replace ONE instance of <gro> with <rw>. Use the specified index (i.e. 8th time it appears).	Debug info.	Read above. PLUS: The index begins at 0 and not 1. An error will be thrown if a non existent index is chosen.
/textengine/sitechats/media/base64upload.php	POST	Filename: <name> Contents in base64: <content> Chatbox:	Upload a file to a chatbox using base64	URL to file and local file path. Format: (invalid file uploaded but still)	The filename cannot be illegal (see newchat_integration.php) The filename may be

		<hidden>		Bytes: 0 Path : C:/wamp64/www/textengine/sitechats/media//uploaded/00.29.23-11.09.21- URL : /textengine/sitechats/media//uploaded/00.29.23-11.09.21-	timestamped if PID 17 is enabled. Empty chatboxes or filenames are be sent to a fallback directory instead of failing.
/textengine/sitechats/data/datacall.php	GET				See CBEDATA section for more details on this API endpoint.

3.1.2 Examples:

```
$ curl -d
"name=goodfilenamingpractices.txt&hidden=04universal-upload&content=aGVsbG8gCmhvdyBhcmUgeW91IAptdWggZGFybGluZwp0b2RheT8=" -X POST
http://localhost:8080/textengine/sitechats/media/base64upload.php
```

```
$ curl
http://localhost:8080/textengine/sitechats/sendmsg_integration?msg=Hello%20Cruel%20World&write=42069&encoder=ASCII&namer=RootOfficial&uid=USER_ID_R1&ukey=root1234
```

3.2 Error Codes

If an error is returned there will usually be an error code followed by any debug information, the error codes are in this format:

[err:XX] Stop: Generic Error
[warn:XX] Warning: Generic Warning
The string “Stop: “ will also appear giving the English error text. The error codes are as follows:

```
00 reserved for generic errors and lazy people
```

01	*executionpoint error (internal error)*
02	*checkcondition error (internal error)*
03	*file IO error (internal error)*
04	*API lockdown status*
05	*This Chatbox Does Not Actually Exist*
06	*Forbidden string*
07	*Forbidden chatbox destination*
08	*Protected file*
09	*Local level needed*
10	*Sudo level needed*
11	*This UID is locked out*
12	*HTTP POST for sendmsg is disabled*
13	*403 Forbidden*
14	*404 Not Found*
15	*Reserved, not used*
16	*No/incorrect auth*
17	*Newchat disabled*
18	*This file exists already*
19	*This filename is illegal*
20	*This user level cannot edit*
21	*Not high enough user authorization*
22	*Reserved, not used*
23	*UID/UKEY is not acceptable and masterkey is needed*
24	*begin CBEDATA is on strict mode*
25	*No data type*
26	*.htamainpolicy stopped execution*
27	*Payload too large on HTTP POST*
28	*No media dir to work with*
29	*.htaterminalaccess stopped execution*
30	*User did not exist in first place (UID DEL)*
31	*Failure to delete (unknown error)*
32	*LOADEXE command stopped (unknown error)*
33	*Generic Authorization Error*
35	*Message exceeds char or byte limit*
37	*Could not find string in admineditsreverse*
39	*Invalid index in admineditsreverse*
41	*Misconfigured or missing settings file*
43	*Caught error or exception, error/exception was because misconfig*

Half the time, nobody uses the right error code, preferring a generic error code such as 00 or 33 Generic Auth Error; it honestly doesn't matter.

In some cases HTTP headers may also give information, namely, display.php and some terminal commands and utilities.

4 CBEDATA

4.1 Introduction

As a small child project turned into something far too large to correct or stop, CBEDATA is now a thing.

CBEDATA is a format by itself, sort of like XML. The application level decides what to do with it. By default, there is a CBEDATA reader loaded onto Chatbox Engine that reads chatboxes with the .cbedata extension. These are created by using COPEN cmd or the 'd' option in newchat_integration.

4.2 Writing CBEDATA

CBEDATA works in a structure of containers and subcontainers. Subcontainers can hold attributes of their own and more subcontainers. CBEDATA also ignores any non-CBEDATA lines by default, making it possible to mix in regular text and CBEDATA-readable text.

To explain it further, imagine you have a directory, named main. You can add files, which are attributes that are linked to the main directory, or you can add subdirectories, which are subcontainers.

Here is an example of a file:

```
class[main>
//the main container has one attribute along with one subcontainer
company==Generic Programming Firm;

//the one subcontainer holds zero attributes and two subcontainers
class[company_management>

//first subcontainer, two attributes
class[charles>
years_of_employment==4;
currentlyemployed?==No;
]

//second subcontainer, another two attributes
class[aj>
years_of_employment==2;
currentlyemployed?==Yes;
]

]
```

Using the filesystem analogy from earlier, this is what the tree would look like:

```
main {main container}
└company.txt {attribute attached to main}
└company_management {subcontainer linked to main container}
      └charles {subcontainer linked to company_management}
            └years_of_employment.txt {attrib. linked to charles}
            └currently_employed.txt {attrib. linked to charles}
      └aj {subcontainer linked to company_management}
            └years_of_employment.txt {attrib. linked to aj}
            └currently_employed.txt {attrib. linked to aj}
```

And this is what the XML equivalent would look like:

```
<main>
<companyname>Generic Programming Firm</companyname>
<companymanagement>
```

```
<charles>
<years_of_employment>4</years_of_employment>
<currently_employed>No</currently_employed>
</charles>

<aj>
<years_of_employment>2</years_of_employment>
<currently_employed>Yes</currently_employed>
</aj>

</companymanagement>
</main>
```

The exact syntax is as follows:

Syntax Element	***Notes***
class[container_name>	Defines a new subcontainer as container_name. Multiple subcontainers cannot hold the same name. A subcontainer can contain both attributes and more subcontainers, **although, attributes ALWAYS come first in a container before any subcontainers.**
]	Close any container off with a closing square bracket.
attributename==attributevalue;	Defines an attribute tied to the container it is in.
begin CBEDATA	Best-practice header, to be placed at the beginning of the data.

begin CBEDATA
 	
//non CBEDATA structured
//lines are ignored by default.
 	
class[main_container_name>
a1==v1;
 	
class[subcontainer1>
a1==v1;
	This example contains all of the major elements.
	
	There is the header, the main big container, an attribute tied to the main big container (that comes before any subcontainers);
	
	and then you have two sub containers, each with two attributes.
	
	You also have a comment that shows that ordinary text

a2==v2;
]
 	
class[subcontainer2>
a1==v3;
a2==v4;
]
 	
]
	that is not structured for CBEDATA is ignored.

4.3 Reading CBEDATA

To access CBEDATA, the standard is you ask for an address and a library designed to handle CBEDATA gives it over. To access a certain attribute, you first need to give the library / program the subcontainers to take to get there, and then you have to tell it the attribute. Using the syntax-highlighted example above and saying we want subcontainer2's a2 attribute, we get the address : **"main_container_name-subcontainer2-a2".** As you can see, the '-' character is the default delimiter, although sometimes that can be changed in the parser's settings.

4.3.1 datacall.php endpoint

Endpoint Location	Method	Parameters	Purpose	Return Values	Remarks
/textengine/sitechats/data/datacall.php	GET	Chatbox name: <src> Data path: <path> Type*: <type>	CBEDATA parser through HTTP GET	Depends on the type, or debug info on error. Types: "attr": get attribute value from path. "var": get attribute value in "att==val" form. "attr-name": get attribute name "class": get	If a file is PID 31 protected, you can't read from it.

				entire subcontainer (if you get attribute1 of subcontainer, you get the entire subcontainer) remember that you have to still reference an attribute in "class" mode. "raw": get entire file. Note that you still must reference an attribute.	

4.3.2 cbedata.py

There is a Python parser for CBEDATA. First, you should import the module with:

import cbedata

Now that you've figured out basic Python, you need the contents, whether it be from a remote server, a string literal or from a file. Let's use our example from earlier.

COMPANYINFO.cbedata:

```
class[main>
//the main container has one attribute along with one subcontainer
company==Generic Programming Firm;

//the one subcontainer holds zero attributes and two subcontainers
class[company_management>

//first subcontainer, two attributes
class[charles>
years_of_employment==4;
currentlyemployed?==No;
]

//second subcontainer, another two attributes
class[aj>
years_of_employment==2;
currentlyemployed?==Yes;
]

]
```

```
MAIN.py
import cbedata
f = open('COMPANYINFO.cbedata','r')
contents = f.read()
f.close()
```

There are now two functions; one to return strings and one to return iterables. The non-iterable one is called get_offline(); the iterable one get_offline_obj().

Function	Parameters	Return Values	Remarks

get_offline	st, path, type, delim = '-'
	
	st = CBEDATA formatted string
	
	path = data address
	
	type = format of returned data
	
	delim = data address delimiter ('-' character by default)
		There are four return formats:
		
		- cls: return entire subclass
		- var: return attribute in varname==varvalue
		- val: just return the value
		- raw: get entire file
			These work the same way as the PHP endpoint. You have to refer to an subcontainer/datapoint even if you are just getting the subcontainer text.

get_offline_obj	Same as above.	There are five return formats: - list-key: get all attribute names in a subcontainer - list-val: get all values of attributes in a subcontainer. - dic: get each attribute	These ones do NOT require you to address a specific attribute, rather, the program wants an address that refers to a subcontainer.

		name as the dictionary key and get each attribute value as the dictionary value. - sbc-list: get a list of all subcontainers in a container. - raw: give the entire file	For example, if you want to get the dic details out of Charles' subcontainer, you would use ***"main-company_management-charles"*** instead of ***"main-company_management-charles-attribute".***

4.4 Remarks

- PID 57 dictates whether or not the 'begin CBEDATA' header is strictly necessary
- Parsers work by finding certain strings recursively (they find the main container, then a subcontainer, and then the attribute). This is what allows you to insert text anywhere and have the parser ignore it.
- <> = required args, [] = optional args, {} = auth

4.5 Escaping

- In both the Python parser and PHP parser, these characters are escapable:

>	;	==	[	]

- Use ^char to escape, for example: ***^;***

- These are used when a value has characters that could cause errors when the parser reads over them. When you escape them the parser will ignore them.

5 Terminal + “DOSKEY”

One of the endpoints is the Remote Management Terminal. This endpoint allows for the remote management and modification of files as well as things such as authentication.

There are three GET parameters plus UID/UKEY:

$_GET[‘cmd’]: the command
$_GET[‘params’]: parameters (what you put here depends on the command)
$_GET[‘pass’]: password (master)

+ UID/UKEY slots (if you use these don’t use the master password slot)

5.1 Default Commands

The current help message is listed as the following:

```
COMMANDS:

*del: deletes chatbox with number (parameter).
*delhtml: same as del, but only for html chatboxes
*xcopy: copies chatbox with number (parameter) to a separate area for
safekeeping. use xcopy --append to not erase any currently existing
copies of that chatbox
*wipe: delete the contents of a Chatbox but not the Chatbox itself

vers: shows the CBE version, no required parameters. use
'showall::YES' to bring up the entire credits file instead of just a
version number
*banhammer: bans IP address with value (parameter)

*mkdir: make a Media Directory where there wasn't one, the media
directory will be named (parameter)
*mcopy: copy Media Directory with name (parameter) to the specified
separate area for safekeeping. Using WILDCARD-ALL as the parameter
allows you to copy all media dirs
*mload: pull Media Directory with name (parameter) from safekeeping to
the main Media dir. The destination dir must already exist. Do not use
WILDCARD-ALL with this command.
*mdel: remove a specific Media Directory with name (parameter) without
deleting the Chatbox. Using WILDCARD-ALL removes all media
directories, so be careful.

*cbroadcast: broadcast message with contents (parameter) to all legacy
and HTML chatboxes. Use DRYRUN to show which files will be written to
without actually writing.
*^_copen: open a Chatbox. parameters slot syntax: FILENAME.FILE-EXT
--MEDIAOPTION
clist: list all Chatboxes and whether or not they are protected
*cload: copy Chatbox from designated dir to main sitechats dir
*csend: write a message to a Chatbox bypassing nogo phrases and
UID/UKEY checking. Use csend --nobreak to not use a newline char when
writing to the Chatbox. Can be disabled by PID 77. Use chatbox;message
in the params box.
help: brings up this help message, no parameters
```

*^ecfg: configure .htamainpolicy. this command can be disabled by PID 35
*^change: change Master Admin Password

*^cmd add: add a custom command in this syntax: @Event;Condition;String;Includepath
*^cmd del: remove a custom command in this syntax: @Event;Condition;String;Includepath
*^udb add: add a user and give them a permission. the parameter should be in this syntax: UID Name Password Groups (space char as delimiter)
*^udb sdel: delete a user with UID as parameters. works when you only have a UID and no other details
*^udbgrab: grab user info. without sudo you'll get their groups and perm level and name, with sudo powers you get all details. takes UID as params
*^udb del: delete a user. you'll need to provide their information in the parameter slot with this syntax: UID Name Password Groups (space char as delimiter)

*^lock add: lock a UID out from a Chatbox in this syntax: [chatbox no.] deny from [UID]
*^lock del: unlock a UID from a Chatbox with the same syntax as LOCK ADD
*^filesafe add: add a file to be write protected. syntax: chatbox::who to restrict to (sudo, login or local):: OR g:group1//group2
*^filesafe del: remove a file to be write protected. syntax: chatbox::who to restrict to (sudo, login or local):: OR g:group1//group2
*^readsafe add: add a file to be read protected. syntax: chatbox::who to restrict to (sudo, login or local):: OR g:group1//group2
*^readsafe del: remove a file to be read protected. syntax: chatbox::who to restrict to (sudo, login or local):: OR g:group1//group2
*^group add: add a user group command, either GROUPNAME give sudo or GROUPNAME cantrun CMDNAME
*^group del: removes a user group command, in the same syntax as group add
*validatormgr: adds a validator command to a media dir (the media dir must exist before hand). param syntax: [chatbox]/[command].

```
*^ccfg: show entire config file to see which parts you want gone and
which parts to add too inirecovery: starts the 'I FORGOT THE PASSWORD'
procedure. Opens a Chatbox then gives you time to enter the backup
code from .htamainpolicy.

No Verbose allows you to suppress the output of the command (unless a
fatal error happens on command execution). This feature is only for
the web client.

* = requires UID/UKEY or password
^ = requires password and cannot take UID/UKEY (by default, it is
possible to change some of these commands in .htamainpolicy)
_ = three options: --allowmed, --allowmedhtml, --forbidmed
If there are dangerous commands, there are people who will find a way
to mess it up. This terminal does not stop you from making bad
decisions. If we are only free to make good decisions, we are not free
at all.
```

A table of the commands is located below:

(authlevels:

- 0 for anyone can run (ie help)
- 1 for requires any sudo by default (ie del-html)
- 2 for requires master password (ie change)
- These levels can be changed by editing PID 101 as well as PIDs 41 51 55 73 .

)

CMD	Auth	Parameters Slot	Purpose/Remarks
del	1	str: chatbox number	Deletes a chatbox and any associated media directory if it exists, unless it is protected by .htaterminalaccess
del-html	1	str: chatbox number	Delete an HTML chatbox and it’s associated media directories (HTML chatbox directories are built different), unless it is protected by .htaterminalaccess
xcopy	1	str: chatbox number	Copy a Chatbox to the safekeeping directory

			as specified by PID 5
xcopy --append	1	str: chatbox number	Does the same thing but appends instead of overwrites.
wipe	1	str: chatbox number	Clear a Chatbox out, unless it is protected by .htaterminalaccess
vers	0	none	Show Chatbox Engine version. Requires CBEDATA endpoints to be active.
banhammer	1	IP address	Ban a certain IP address using the server's built in accept/reject system (for example .htaccess)
mkdir	1	Any valid directory name	Create a media directory despite the current status of the associated Chatbox
mcopy	1	Any existing directory name	Copy the media directory to the safekeeping location as specified by PID 7.
mload	1	Any existing directory name in the safekeeping location.	Reverses mcopy. NOTE: for m** commands, WILDCARD-ALL may be an acceptable option, but don't. Just don't.
cbroadcast	1	The message to write, or DRYRUN to see which chatboxes are affected without writing.	Write a message to *.html files and non extension files.
copen	1/2	Valid filename and media flag (--forbidmed or --allowmed)	Example: copen 1357 --forbidmed This opens a new Chatbox and media dir, even if the name is protected. PIDs 55 and 101 can effect the authlevel required to run this command.
clist	0	none	List all chatboxes.
cload	1	Chatbox name (one in the safekeeping area)	Copy a Chatbox back to the main area from safekeeping.
csend	1	chatbox;message_content	Bypassing filters and sending a message. Adds newline automatically.

csend --nobreak	1	Same thing	Does the sae thing but does not automatically add a newline. Use %nl to fill in for newlines or use the character directly. These two commands can be disabled by PID 77.
help	0	none	Brings up the help message from earlier.
ecfg	2	none	Bring up the mainedit utility which can edit .htamainpolicy. Requires admin password to both read from and write to the main config file. Built into the terminal directly.
ccfg	2	none	Bring up the contents of the .htamainpolicy as a reference (previously you edited completely blind).
change	2	new admin password	Changes the admin password. Requires the admin password. Built into the terminal directly.
cmd add cmd del	2	@Event;Condition;String;Includepath	Adds or deletes a custom command.
lock add lock del	2	[cb#] deny from [uid]	Adds or deletes a lock command.
filesafe add filesafe del	2	chatbox::allowed:: chatbox::g:group::	Add or delete a filesafe command. The top one takes local, sudo or login, the bottom one takes g:groupname.
readsafe add readsafe del	2	chatbox::allowed:: chatbox::g:group::	Add or delete a readsafe command. The top one takes local, sudo or login, the bottom one takes g:groupname. This only impacts the display.php endpoint and read perms.
group add group del	2	GROUPNAME give sudo GROUPNAME cantrun COMMANDNAME	Add or delete a GROUPS command, using those syntaxes.
udb add udb del	2	UID Name Password Groups	Add or delete a user.
udb sdel	2	UID	Delete a user when you only have the UID.

udbgrab	**	UID	Get user info when provided a UID. **: with sudo powers you get all the details, without sudo powers you get everything but their password
portal-login	2	none	Opens the editing portal, capable of editing ALL files. Requires cedit.php to be loaded into the terminal dir and portal.php to be in the sitechats area.
validatormgr	1	[chatbox]/[cmd] 04uni/lock:YES	Appends a command to a .htafiletxpolicy which dictates patterns that cannot be in file names. Refer to Section 1 for syntax of commands. Note that the media dir must exist beforehand.
test	0	none	Tests the DOSKEY function.
testsudo	1	none	Tests if a pair of logins are correct.

5.2 DOSKEY
There are two things to understand before going more in depth:

1. All commands are stored as PHP files in the /terminal directory.
2. Wikipedia defines DOSKEY as the following:

A command for DOS, IBM OS/2, Microsoft Windows, and ReactOS that adds command history, macro functionality, and improved editing features to the command-line interpreters COMMAND.COM and cmd.exe.

If you want to add a custom command, you'll need to do it through the /terminal dir and the "DOSKEY" feature.

Let's say you have a PHP file that holds a command:

test_command_file.php
```
<?php
if ($_GET['cmd'] == 'test command' && $_GET['pass'] == $pass) {
echo 'test command successful';
}
?>
```

Now, if you move that file over to the terminal directory, you can invoke that command by entering in the script name: test_command_file

If you want to use a different name (i.e. test), you can add a key in .htaterminalkeys.
A prominent example of this is the udb del and udb add commands; the file is named udbmanager.php but the command names are udb add and udb del.

To link a command name to a PHP file, use:

*Example**: uid add::%rdir/%sc/terminal/udbmanager.php;***
*Struct**: command name::%rdir/%sc/filepath;***

(where %sc is the sitechats directory name and %rdir is the textengine path)

Note that if you want to access a file such as .htamainpolicy, the terminal cmd executes under the scope of the sitechats dir, not terminal.

⚠️ **There are a few known glitches with the terminal system:**

1. With extra newlines, this area completely breaks.
2. The first command is always ignored, no matter whatever you do with it. Just fill in the top command with a dummy command and start the actual content below.
3. A known glitch with all “file exists?” checkers is that it just straight up reports false info, which may result in the terminal running a PHP file that doesn’t exist, or stating a file doesn’t exist when it clearly does. This glitch appears to get better with PHP 8+.

BEYOND THIS POINT, THE INFORMATION WILL ONLY BE USEFUL TO PEOPLE WHO MAKE CLIENTS OR MODIFY CHATBOX ENGINE (ADVANCED USERS).

6 mainlookup.php wrapper

A PHP wrapper for the config file is located in /sitechats, /sitechats/data and /sitechats/media. These contain helper functions that help you navigate the main config file. Before this all scripts had to have their own method of accessing .htamainpolicy, however, now there is a standardized way to do so for extension creators and modifiers.

To get the functions use **include ‘mainlookup.php’;**

An indepth explanation of each function is located below:

6.1 CBauth wrapper functions

uidlsk(str $uid, str $ukey) → bool
This takes a UID and UKEY set and returns true if they match and false if they don’t match. Any valid UID/UKEY pair will return true.

If password hashing is enabled, it will automatically hash the password and check it against the stored hash.

uid(str $uid, str $ukey, int $attrno) → str
This takes a UID/UKEY pair as user identification, and returns an attribute of that user. The intcodes for each attribute are below:

0: UID
1: On screen name
2: UKEY
3: Whether or not the user has sudo power or not (if they are in any sudo group). **
4: Raw user groups attribute.

**: returns str ‘notsudo’ for false and ‘sudo’ for true. does not return a bool.

uid_db() → str
Returns entire UID/UKEY set as a str.

group_db() → str
Returns entire [GROUPS] section from .htamainpolicy.

6.2 File Protection functions

wr_db() → str
Returns the entire FILESAFE area as a str.

wr_bycb(str $cb, int $attrno) → str
Returns an attribute about a Chatbox. The intcode doesn’t actually do anything but it is still a required argument.

For example, if you call cb:04universal-upload and it has g:sudousers protection, the function will return g:sudousers as a str.

ga() → str
Returns the entire UID/UKEY LOCKOUT list.

6.3 General Parse Operations

plsk(int $pid) → str
Returns the value attached Policy ID (for instance, plsk(int 9) → str “America/New_York”).
Note that all PIDs are odd.

gcpp() → str
Returns the CPOLICY section as a str.

customreturn(str $section) → str
Returns a custom named section (for example, if you have an extension named RADIOBOX, it could tell you to create a section in config named RADIOBOX-CFG specifically for that extension. It would then call customreturn(“RADIOBOX-CFG”)).

7 chatboxengine.py wrapper

I've always wanted to be important enough to have my own Python module. Now granted I did make it in my own name by myself for me (implying that people don't really care) but it's still cool.

First off: create a session object that has all your logins and whatnot. __init__ works as follows:

session = chatboxengine.Session(str <server>, str [masterkey], int [timeout], str [uid], str [ukey], str [url = /textengine/sitechats], str [protocol] = 'http')

Most of these details are self-explanatory, with the following ones being somewhat confusing:

<server> being the domain name or IP of the remote server (port number included)
[timeout] being the point at which the request raises an error instead of holding

[protocol] being whether or not to use HTTPS

[url] is the path to the API endpoints. Best practice dictates that this is /textengine/sitechats, however, this path may be different (/textegine/instance01/sc). This is common in servers with multiple instances of Chatbox Engine.

The following attributes can be read from the session object:

sv = session.server
pw = session.password
uid = session.uid
ukey = session.ukey
url = session.url

To create an individual Chatbox object (the object that lets you mess with a Chatbox individually), use

cbobj = chatboxengine.Chatbox(<sv>, <chatbox name>, [alias], [encoder] = 'UTF-8', [timeout] = 15, [url] = '/textengine/sitechats')
Most of the time it's better to use the logins stored in the session object to fill in spots for the __init__ on cbobj.

Now you have the chatbox object you can perform the following operations:

Function with Params	***Purpose***	***Remarks***
get(uid = '', ukey='')	Gets a Chatbox	Uses display.php endpoint, so may be subject to read perms.
make(option = 'l', allowmed = 'forbidmed'	Creates a Chatbox	Uses newchat_integration and the same options it would normally take (check API section).
write(contents, newline = 1, uid = "", ukey = "")	Write to a Chatbox	Writes to a Chatbox using the logins provided. The name, encoder and server are derived from the Chatbox() object info. Uses HTTP GET. Not binary safe.
wipe(password)	Wipes a Chatbox	Set password to 'UID::UKEY' if you need to use CBAUTH logins instead of the masterkey.
delete(password)	Delete a Chatbox (based	Same remarks as above

delete_html(password)	on what type it is)	regarding the password.
edit(find, replace, password)	Edits using adminedits.php	Edits the Chatbox, replacing every instance of FIND with REPLACE. Same remarks regarding the password as wipe()
editsplice(find, replace, password, splice)	Edits using admineditsreverse.php	Same thing as edit() but uses the new endpoint that can pick and choose the nth instance of a str to remove.

NOTES:
1. The string ‘2’ is returned on failure. Not sure why but this is the way it has been so to maintain interoperability ‘2’ it will stay.
2. There used to be a HS_chatbox object to deal with the old High-Security extension. With read perms now phasing the old, jank HS extension out, this is no longer needed and attempting to run it will result in RuntimeError.
3. On success, the HTTP response code will be returned, even if it’s not a perfect 200.
4. This version of the library currently only supports HTTP and HTTPS.

8 Multiple Instance Management

One of the least awaited yet also most useful features of Chatbox Engine is the ability to have multiple instances of it on one server on one port.

Old versions of Chatbox Engine were locked into the following server structure (yes, even with these names):

```
server_root
└ textengine
        └ sitechats
                └ media
                └ copies
                └ others
```

However, now it is possible to have the following structure (granted you edit the config files correctly):

```
server_root
└ first subfolder
        └ textengine01 [textengine]
                └ instanceA [sitechats]
                        └ media
                        └ copies
                └ instanceB [sitechats]
                        └ copies
        └ textengine02 [textengine]
                └ instanceC [sitechats]
                        └ media
                        └ copies
```

In the main config files you can change the paths to the instance of Chabox Engine using these policies:

PID 3 - path to /textengine, the directory that hosts the subdir that in turn hosts Chatbox Engine

PID 5 - path to an offshore folder for Chatboxes (multiple instances can share one offshore)

PID 7 - path to an offshore folder for media directories (multiple instances can share one offshore)

PID 107 - name of your sitechats, the subfolder contained within your 'textengine' (not the full path, but rather, the folder name)

Throughout the docs we have assumed that the parent directory is textengine and the child directory that houses an instance of Chatbox Engine is sitechats. For our instance, the following edits would be necessary to make instanceC run (as an example):

PID 3 becomes '/mycoolserver/server_root/textengine02/' (parent directory)
PID 5 becomes '/mycoolserver/server_root/textengine02/instanceC/copies' (or other path)
PID 7 becomes '/mycoolserver/server_root/textengine02/instanceC/copies/media' (or other path)
PID 107 becomes 'instanceC' - child directory name (***NOT THE PATH****)*

⚠️ It is still however necessary to have one folder, then another folder, then the instance of Chatbox Engine in this structure:

```
└ textengine01
        └ instanceA
```

(textengine is the parent, instanceA is the child, everything inside instanceA is the actual instance and subdirs).

The path you take to the parent can vary (/first_subfolder/textengine01 in our example), however, instanceA (or the child) MUST be an actual subdir of the parent.

TLDR: Don't place more folders in between instanceA and textengine01, but change the path you take to get to textengine01 all you want.

The most recent example I have is our prod server (which we use to evade school web filters):

```
root
└ textengine
          └ sitechats
               └ copies
               └ media
          └ side project
               └ other dirs
```

The textengine path for both side_project and sitechats is set to "/storage/ssd2/700/16518700/public_html/textengine", however:

- sitechats has 'sitechats' listed for PID 107
- side_project has 'side_project' listed for PID 107

TLDR: Same textengine path, different sitechats names (in this example).

9 Password Handlers

This is what the default user account 1 (D1) password string is:

hash//afAGJTYpxFQOPIXwIyNUMQ07NQlBvM1FoFHZfKBP+%%password=ccae0e7e74bcf86466c097490b24b9da02e78dddd9bfb7e9fd9b024633f3efa5642a4aa3879abc68dbe4b140b6a2da86732623deb91f3e473b77768faab66e94

This is what is responsible for hashed passwords. In this case, “hash” (the first thing in the string) is the name of where the password is sent off for handling). In this case, the password is sent off to another script where it is hashed before being checked against the stored hash. The script’s name is hash.php and is located in the hash subfolder.

However, it is possible to write your own password handler script. It needs to have these two functions:

```
function input_password_handler($userID, $suppliedPassword,
$correctPassword)

function output_password_handler($userID, $suppliedPassword,
$correctPassword)
```

$suppliedPassword is the password that the user enters in, and $correctPassword is what’s in the password slot of the account (in this case, it’s the entire string in red).

The function input_password_handler should return the processed version of the user entered password. In hash.php, input_password_handler returns the SHA-512 hash of the inputted password plus the salt.

The function output_password_handler should return the processed version of the password that the server stores: in hash.php, the end result is the hash "ccae0e7e74bcf86466c097490b24b9da02e78dddd9bfb7e9fd9b024633f3efa5642a4aa3 879abc68dbe4b140b6a2da86732623deb91f3e473b77768faab66e94".

After that, the results of the two functions are checked against each other. In hash.php, assuming the correct password was entered, both functions should return an identical SHA512 hash.