

Storing SpamAssassin User Data in a SQL Database

Michael Parker

[Start Slide]

Welcome, thanks for coming out today.

[Intro Slide]

Like most open source software, heck software in general, SpamAssassin has grown up over time. This means that sometimes we get really cool features that may or may not work and play well with others.

Three of these features are SQL user preferences, auto-whitelist and bayes. Using these features you can move practically all of your user stored data off the disk and into a database. Well, I know it's most likely still on a disk if you're using SQL storage, when I say disk I mean local disk, and when I say local disk I mean user home directories or virtual config directories. In some setups, managing that information is a nightmare, and that's what I'd like to try and get rid of.

[General Database Setup Slide]

Before we get into each feature, I'd like to talk a little bit about general database setup. Each of these features makes use of perl's DBI database access module. This means that you need to install the appropriate driver (DBD) module for whatever database you choose to use. For more information see the DBI/DBD documentation, O'Reilly DBI book, DBI mailing lists, etc

Honestly, there are so many different database setups it would be nearly impossible to cover them all. For consistencies sake I'm going to talk in terms of MySQL for most of my examples. That in no shape or form means that things aren't supported in other database engines. There are some cases where certain database engines may not work well. I will cover those and the reasons why. If you have strong feelings about a particular database engine and find that it doesn't work well with SpamAssassin I encourage you to work with the developers to help make it better. We've only got so much time and so much access to systems to work on the things.

When setting up your database connection in SpamAssassin you basically need three pieces of information, the data source definition and assuming your database requires them a username and password.

[Determine your DSN Slide]

The data source, or DSN, tells DBI what database driver to load and provides connection

information to that driver. You should read the documentation for your particular driver to determine the proper information needed. Here are some examples. Obviously it's important to get this part right, otherwise everything just won't work correctly.

[Database Architecture Slide]

Whole sessions, heck whole conferences, can be spent on how to properly architect your database, so I'm not going to get into it much here. However, there are some considerations you can take into account. If you're architecting for a high traffic site then you might consider several options to help alleviate the load.

One option, is to use multiple databases. This gives you lots of options on how to handle the data. You can put database files on separate disks, give different permissions or make them different types.

Another option, once you've split up your databases, is putting the databases on separate physical machines. This gives you tons of options.

One drastic option is different database engines all together. You can put your user preferences database on PostgreSQL, your auto whitelist on SQLite and bayes on MySQL. Obviously, this would be odd, but you could do it if you wanted to.

[Basic Setup Slide]

Here is the basic setup for a MySQL and PostgreSQL database. This sets up a single database, spamassassin, with access for a single user, sauser.

MySQL

```
mysql -u <adminusername> -p mysql
```

```
Enter password: <adminpassword>
```

```
mysql> create database spamassassin;
```

```
mysql> grant all on spamassassin.* to sauser@localhost identified by 'spamki11er';
```

PostgreSQL

```
psql -U <adminusername> template1
```

```
template1=# CREATE USER sauser PASSWORD 'spamki11er';
```

```
template1=# CREATE DATABASE spamassassin OWNER = sauser;
```

[User Preferences Intro Slide]

Storing of user preferences in a database is a fairly simplistic process. Configuration options are stored in the database in key/value pairs, or more correctly preference/value pairs. These pairs are selected based on username, more on that in a minute, and feed into the config parser.

One thing that can be very confusing, depending on your setup, is how the username is determined by SpamAssassin, and in this case spamd. Simply put, it uses whatever username is passed in via the spamd protocol. If you use spamc, you can either specify the username on the command line (via -u) or let it figure out the username on it's own by looking up the passwd information for the effective uid currently running spamc. Other systems are responsible for using the User header in the spamd protocol to send the username.

[User Preferences Database Setup Slide]

User preferences setup is pretty easy. In theory you've already created the database and user so you can go ahead and create the userpref table:

```
CREATE TABLE userpref (  
  username varchar(100) NOT NULL default "",  
  preference varchar(30) NOT NULL default "",  
  value varchar(100) NOT NULL default "",  
  prefid int(11) NOT NULL auto_increment,  
  PRIMARY KEY (prefid),  
  KEY username (username)  
) TYPE=MyISAM;
```

You can either enter the create table command directly or use the supplied definitions in the sql directory and run them using your database command line tool, for example:

```
mysql -u<adminusername> -p spamassassin < userpref_mysql.sql  
Enter password: <adminpassword>
```

[User Preferences Configuration slide]

You then just need to setup the proper config options, setting the DSN for your database and then username and password, if required.

user_scores_dsn	dbi:mysql:spamassassin
user_scores_sql_username	sauser
user_scores_sql_password	spamkiller

[spamd startup/debug slide]

Once that is done, you can restart spamd, this time using the -q or -Q command line parameters and you are all set. In addition, you will need too run with -x to keep spamd from attempting to look up user_prefs files.

```
/usr/bin/spamd -x -q
```

If you run spamd in debug mode, with -D, then you will be able to see where it attempts to find the user prefs in the database when a message is sent to be scanned.

```
logmsg: connection from localhost [127.0.0.1] at port 34601
debug: Conf::SQL: executing SQL: select preference, value ...
debug: retrieving prefs for joe@example.com from SQL server
debug: user has changed
```

If you have any problems you can plug the SQL statement from the debug into your database and see what sort of information is returned and then adjust your database accordingly.

[Populating userpref Table slide]

Populating your database is very simple, for each user and config option you just need to insert a row into the database.

```
insert into userpref (username, preference, value)
VALUES ('@GLOBAL', 'required_score', '10.0');
```

```
insert into userpref (username, preference, value)
VALUES ('@GLOBAL', 'whitelist_from', '*@apache.org');
```

```
insert into userpref (username, preference, value)
VALUES ('joe@example.com', 'required_score', '5.5');
```

```
insert into userpref (username, preference, value)
VALUES ('joe@example.com', 'whitelist_from', 'sue@betty.org');
```

```
insert into userpref (username, preference, value)
VALUES ('joe@example.com', 'blacklist_from', '*@spamking.com');
```

When scanning a message for joe@example.com then the global score of 10 will be overridden by the user specific 5.5. The email address su@betty.org is added the global whitelist_from list of *@apache.org. Finally, *@spamking.com is added to the users blacklist_from.

[GUI Frontends Slide]

Obviously for large installations you aren't going to be manually inserting rows into the database. There are several different GUI front ends to help in this manner. Honestly, I don't have much experience with them and can not recommend any. There are a collection of links available on the SpamAssassin wiki:

<http://wiki.apache.org/spamassassin/WebUserInterfaces>

I would encourage you to experiment and find the one that works best for you. This may also be something that building your own becomes feasible since the interface is so simple and 9 times out of 10 you'll want to heavily customize the offering anyway.

[Creating a Custom SQL Query Slide]

There are many cases where you might not want to operate in the default configuration. Maybe you need to change the table name to match with an external system, or maybe you would like to join with another table to get some additional information. Creating a custom SQL query is the way to do that.

The default query is pretty simple, select everything from the table where the username matches the given username or @GLOBAL and order by the username. This gives you the ability to have global user preferences than can be overridden by the the individual users preferences.

Really, the sky is the limit here. The feature after all is to help handle all of the cases where you might want to use SpamAssassin that we can't think of and don't want to have to add special handler code for. Some ideas that you might be interested in. Maybe, just changing the name of the user preferences table, or maybe you want to change how global preferences are sorted so they override the individual user preferences. It is also a method for providing domain based global preferences.

[Custom SQL Query Variables Slide]

Here we see the `_USERNAME_` variable which corresponds to the current users username, as determined by SpamAssassin or passed into spamd. `_DOMAIN_` is the portion of the username that comes after the @ (at sign). In the case that the username doesn't have an @ (at sign) then the `_DOMAIN_` value will be blank. Other variables available are `_MAILBOX_` which is the part of the username before the @ (at sign).

If you happen to compare the docs to this slide then you'll notice that the `_TABLE_` variable is missing. To tell the truth, `_TABLE_` is next to worthless, there is no way to change what table name gets inserted there, it will always be `userpref`. So if you need to

change it you just hard code the table name in your query.

[Auto-Whitelist Intro slide]

We all know that the majority of development in open source software is based on someone scratching an itch. That is exactly how I started working on storing auto-whitelist and bayes data in a SQL database. I figured, if I can store user preferences in the database, why not other data and why hadn't someone already thought about it. Sure enough, I checked Bugzilla there already was a bug.

When I started looking into the possibilities it became obvious that the task would be easy on one hand, and somewhat difficult on the other. The easy part came about because several subsystems in SpamAssassin make use of storage APIs, so it would be easy enough to code up a SQL based module and simply plug it in. The hard part was that bayes had a storage API that was highly geared towards DBM based databases.

With auto-whitelist having the cleanest storage API, it was easy to get going quickly, so we'll start with that. Plus, setup and use is really simplistic, it really takes care of itself once it is up and running.

The auto-whitelist API is dead simple, get an entry, add score to an entry and remove an entry. I had an implementation in a day or so, although it took a week or so for me to get up enough nerve to actually put it into production. Once I did, I never looked back.

[Auto-Whitelist Database Setup]

Setting up auto-whitelist for SQL is fairly straight forward. The first thing you need to do is create your table space. Setup is very similar to the userpref table, assuming you've already created the database and user login, you can either cut and paste the create table command directly or run the create sql file located in the sql directory.

```
CREATE TABLE awl (  
  username varchar(100) NOT NULL default "",  
  email varchar(200) NOT NULL default "",  
  ip varchar(10) NOT NULL default "",  
  count int(11) default '0',  
  totscore float default '0',  
  PRIMARY KEY (username,email,ip)  
) TYPE=MyISAM;
```

```
mysql -u<adminusername> -p spamassassin < awl_mysql.sql  
Enter password: <adminpassword>
```

[Auto-Whitelist Configuration slide]

Unlike user preferences, auto whitelist does not require spamd to use the SQL based storage. Once turned on and configured you can use it via spamd, spamassassin or the SpamAssassin API. By default, auto whitelist is turned on in 3.0. You can control the behavior with the use_auto_whitelist config option. You then have to setup the auto whitelist specific DSN, username and password:

```
user_awl_dsn          dbi:mysql:spamassassin
user_awl_sql_username sauser
user_awl_sql_password spamkiller
```

You specify the specific storage back end using the auto_whitelist_factory config option. This option takes the name of a module that implements the Address List API. In this case, Mail::SpamAssassin::SQLBasedAddrList or:

```
auto_whitelist_factory Mail::SpamAssassin::SQLBasedAddrList
```

[Your Mother Doesn't Live Here. Clean up Your Own Mess slide]

With auto-whitelist, there is very little maintenance, it just works. One issue with auto whitelist in general is that there is no expiration. This is something that developers are aware of and plan to fix in a future release. With SQL you have some options to solve this problem on your own. There is certainly more than one solution.

I've worked around this by adding a lastupdate column to the database with a TIMESTAMP type in MySQL. This causes the column to get updated with the latest time whenever the row is updated. You can then perform all sorts of queries to expire your auto whitelist data.

```
CREATE TABLE awl (
  username varchar(100) NOT NULL default "",
  email varchar(200) NOT NULL default "",
  ip varchar(10) NOT NULL default "",
  count int(11) default '0',
  totscore float default '0',
  lastupdate timestamp(14) NOT NULL,
  PRIMARY KEY (username,email,ip)
) TYPE=MyISAM;
```

[Roll Your Own Expiration slide]

Here are a couple of examples, obviously, MySQL specific:

For instance, if you want to delete all auto-whitelist data that has not been updated in the last six months:

```
DELETE FROM awl
WHERE lastupdate <= DATE_SUB(SYSDATE(), INTERVAL 6 MONTH);
```

Or just get rid of all the single count (ie only one message received) entries that haven't been updated in the last 30 days:

```
DELETE FROM awl
WHERE count = 1
AND lastupdate <= DATE_SUB(SYSDATE(), INTERVAL 30 DAY);
```

Delete possibly statistically insignificant tokens:

```
DELETE FROM awl WHERE totscore/count < .1 AND totscore/count > -.1;
```

[Bayes Intro Slide]

To me, especially in a virtual user environment, moving bayes data into SQL just makes sense, but I might be a little biased. And if you're doing bayes sitewide, I think it is totally worth it. You'll no longer have to worry about odd permission or ownership problems of you bayes files.

After getting the auto-whitelist SQL back end done, I turned my attention to bayes. It turned out to be a much larger chore than I first envisioned. The bayes storage API was VERY, and I mean VERY, DBM specific, in some aspects it still is. The first task was to do my best to extract out the best possible storage API, either pushing functions down into a specific DBM module or pulling them out into a high level BayesStore module. Once I had that done and a decent SQL implementation we were all set.

[Bayes Database Setup Slide]

The bayes tables are slightly more complicated than user preferences and auto-whitelist. The setup consists of 6 different tables required for operation.

```
mysql -u <adminusername> -p spamassassin < bayes_mysql.sql
Enter password: <adminpassword>
```

[Bayes Config Slide]

The bayes config options are very similar to the other subsystems

```
bayes_store_module          Mail::SpamAssassin::BayesStore::SQL
```

```
bayes_sql_dsn               dbi:mysql:spamassassin
```

bayes_sql_username	sauser
bayes_sql_password	spamkiller
bayes_sql_override_username	sitewideuser

[Converting from DBM to SQL]

If you are looking to convert your Bayes database from DBM to SQL then you will want to use the --backup and --restore sa-learn options.

First off, if you haven't yet upgraded to 3.0 then do that and follow the procedure in UPGRADE to get your bayes database upgraded to the new format.

Once you are up and running under 3.0 the procedure is very simple:

1) run sa-learn --backup > backup.txt

1a) You can optionally run sa-learn --clear here to remove the old DBM based files.

2) Switch your local.cf config to use SQL based bayes

3) run sa-learn --restore backup.txt

[Bayes using Postgres]

...

[Future Directions Slide]

Obviously, SpamAssassin is always evolving and you have got to keep an eye on the future. Here are a few future related items you might be interested in. I will say, as with any open source project, if you would like to dive in and implement one of these items, patches are more than welcome.

o Learning Support for spamd

In a virtual user environment it can be somewhat of a burden to have to run sa-learn from the command line to populate the bayes database. By adding learning support to the spamd protocol you'll be able to learn from anywhere and any client. That way you can build learning directly into your mail client rather than jumping through hoops with shared IMAP mailboxes or forwarding aliases.

o Persistent Database Connections

Part by necessity, you can after all choose to use just 1 or all of these subsystems together, and part because they all came about at different times the subsystems do not share database connections. That means that if you use SQL for user preferences, auto-whitelist and Bayes storage then you will open up 3 (possibly more) separate database connections per message. With MySQL, where it is very cheap to create a new connection this works out ok, but for other systems it can cause a problem. The goal is to basically create a pool of database connections per child that can be reused for each connection. You might be able to simulate this behavior using DBD::Proxy.

o Better PostgreSQL Support

Perhaps I should say, better non-MySQL support, but lets not split hairs. As we saw, PostgreSQL just does not perform well. I really don't know how well other database engines perform, mostly because I haven't been able to test and benchmark them. If you have a favorite database (other than MySQL and PostgreSQL) feel free to contact me, I'll be happy to help you get setup and running and we'll see about doing some benchmarks.

o Journal Like Support

One advantage that DBM based storage has over SQL is its use of a journal file for token updates. SQL has to update the token information directly in the database. Several options exist and I started exploring several of them but the code got to the point that scanning was actually faster than DBM (even with the journal active) so I stopped. However, it could possibly help speed up learning so it is something I would like to address in the future.