Gilles Chehade / جيل شحادة

foobarbaz-ing since 1981

Jan 30, 2020 23 min read

Share

OpenSMTPD advisory dissected

go to comments

```
TL;DR:
- Qualys released an advisory for a bad, bad vulnerability
- an MTA is a very bad software to have a vulnerability in
- hole was plugged but that's not enough, similar bugs should be mitigated in the future
- article discusses what could have prevented escalation despite the bug
```

What happened ?

Qualys contacted by e-mail to tell me they found a vulnerability in OpenSMTPD and would send me the encrypted draft for advisory.

Receiving this kind of e-mail when working on a daemon that **can't revoke completely privileges** is not a thing you want to read, particularly when you know how efficient they are at spotting a small bug and leveraging into a full-fledged clusterfuck.

While I was waiting for them to mail me back the draft, I immediately jumped to the privileged process which is what worries me the most, and started reading all the entry points to that process at runtime.

This isn't a big task, the privileged process doesn't do many operations at runtime:

- it opens `~/.forward` files located in home directories
- it authenticates username/password pair against `bsd_auth(3)`
- it creates user-owned mda processes for mail delivery agents

In the case of `~/.forward` files, it checks if the user has a `~/.forward` file with correct permissions and mode, opens it then passes the file descriptor back to an unprivileged process without doing anything with it. I didn't see a way this could be exploited.

In the case of authentication, which was my main worry given the recent vulnerabilities they found in the `bsd_auth(3)` layer, it received a username and password and passed them directly to `auth_userokay(3)`, returning the result to unprivileged process. I was highly suspicious of a bug here but given how simple the code is, I didn't see a way this could be exploited either.

Finally, the user-owned mda processes creation became my main worry. It doesn't do much as it basically calls `fork()` and executes the mda command **using recipient user privileges**, but it also has a special case for `mbox` which **can't work without privileges**.

I had hope that the bug they found would be elsewhere and not so bad as the privileged process is fairly isolated. It can only be reached by going through another unprivileged process and its input validation. The **worst-case scenario** would be finding an exploitable bug either in the `mbox` delivery special case, or in the small portion of code before privileges dropping for other mail delivery agents. I focused efforts on reading the code path again and again but didn't find a bug. I gave up after a few hours and decided to wait for the draft.

I woke up to an advisory draft **exploiting the worst-case scenario**... through a **logic bug in the input validation code of an unprivileged process**.

Perfect, just perfect.

The problem

OpenSMTPD uses an input validation function, `smtp_mailaddr()`, to ensure that e-mail addresses that enter the queue are valid.

What the function does is that it checks that the e-mail address parses into a `struct mailaddr`, then validates the local part with `valid_localpart()` and the domain part with `valid_domainpart()`. The `valid_localpart()` function checks that the local part only contains characters from an allowed set. The `valid_domainpart()` either checks that the domain is a valid IP address using `inet_pton()`, or that it is a valid hostname using `res_hnok()`.

The `smtp_mailaddr()` function also has to handle **two cases** where an invalid address is still acceptable and has to go through:

The first one is that of bounces (MAILER-DAEMON) with empty local and domain parts, `MAIL FROM: <>`, which is only acceptable during the MAIL FROM phase of SMTP.

The second one is that of addresses with empty domains, `MAIL FROM:<gilles>`, which is generated by some MUA when used locally and which will use the local domain attached to the listener, `MAIL FROM:<gilles@laptop.poolp.org>` on my machine. This was initially not accepted by the daemon but had to be implemented for interoperability, which is when the bug was introduced.

Unfortunately for us, Qualys identified a **logic error** in `smtp_mailaddr()`. If one of the checks on local or domain parts fails, `smtp_mailaddr()` checks if it is due to one of the two cases above. In the case of failed domain, it mistakenly does an erroneous check that cancels out the previous call `valid_localpart()` and, as a result, an address that was rejected due to an invalid local part gets accepted again if there is no domain.

```
MAIL FROM:<gilles;test@laptop.poolp.org>
553 5.1.0 Sender address syntax error

MAIL FROM:<gilles;test>
250 2.0.0 Ok
```

This mistake was introduced five years ago with the following diff:

```
===================================================================
RCS file: /cvs/src/usr.sbin/smtpd/smtp_session.c,v
retrieving revision 1.215
retrieving revision 1.216
diff -u -r1.215 -r1.216
--- src/usr.sbin/smtpd/smtp_session.c   2014/07/09 12:44:54     1.215
+++ src/usr.sbin/smtpd/smtp_session.c   2014/10/02 21:27:54     1.216
@@ -1,4 +1,4 @@
-/*     $OpenBSD: smtp_session.c,v 1.215 2014/07/09 12:44:54 eric Exp $ */
+/*     $OpenBSD: smtp_session.c,v 1.216 2014/10/02 21:27:54 gilles Exp $       */

 /*
  * Copyright (c) 2008 Gilles Chehade <gilles@poolp.org>
@@ -1758,21 +1758,20 @@

        if (!valid_localpart(maddr->user) ||
            !valid_domainpart(maddr->domain)) {
-               /* We accept empty sender for MAIL FROM */
-               if (mailfrom &&
-                   maddr->user[0] == '\0' &&
-                   maddr->domain[0] == '\0')
+               /* accept empty return-path in MAIL FROM, required for bounces */
+               if (mailfrom && maddr->user[0] == '\0' && maddr->domain[0] == '\0')
                        return (1);

-               /* We accept empty domain for RCPT TO if user is postmaster */
-               if (!mailfrom &&
-                   strcasecmp(maddr->user, "postmaster") == 0 &&
-                   maddr->domain[0] == '\0') {
+               /* no user-part, reject */
+               if (maddr->user[0] == '\0')
+                       return (0);
+
+               /* no domain, local user */
+               if (maddr->domain[0] == '\0') {
                        (void)strlcpy(maddr->domain, domain,
                            sizeof(maddr->domain));
                        return (1);
                }
-
                return (0);
        }
```

Back then, the mistake had no consequences as OpenSMTPD had a very different implementation for delivery. It didn't handle MDA through external utilities, like is done in other MTA, but used builtin implementations for all methods (`maildir`, `lmtp`, `file`, `mda`, ...). The `mbox` delivery was handled as a special case through a simple `execle()`.

With this way of dealing with MDA, there was no shell involved and the incorrectly validated e-mail address wasn't in the path of delivery, the bug had no consequences despite being there. **This way of dealing with MDA wasn't right though**. It came with its share of complexity and short-comings, most

notably a very complex mda layer handling all methods differently with many special cases, and a more complex `forkmda()` function which is code running in the privileged process and that we want as simple as possible.

The bug became exploitable with another unrelated commit **three years and a half later**, in a different area, when OpenSMTPD switched to the new grammar and brought support for MDA through external utilities, ditching its builtin delivery methods.

Basically, instead of each delivery method coming with its own structure and custom checks in the entire code path, they were reimplemented as standalone executables that could be used by any MTA. From this point, OpenSMTPD didn't know the difference between `maildir` and `lmtp`, they were just regular mail delivery agents. The MDA execution code path got simplified by removing all delivery-specific cases and unified so it would craft a command line, **using what we assumed to be correctly validated input**, that could be executed after the `forkmda()`function dropped its privileges.

Because the delivery agents are standalone executables that expect their options on the command line with quoting and escaping support, we decided to use the shell to execute the command rather than rewriting our own command-line parser that might contain parsing bugs. **This is something done in other MTA, we didn't invent a clever way to be lazy**. The shell is executed with the recipient privileges and passed the mda command, similarly to if the recipient had executed the command themselves in their own shell. Since the command is either crafted from `smtpd.conf` which is controlled by postmaster, or `.forward` which is controlled by the recipient, there is supposedly no problem **as long as the input is properly validated**.

We could jump to the conclusion that using `system()` instead of `execle()` is what allowed the escalation but this is inaccurate. Incorrectly validated input and `system()` made the **exploit** possible, **not the escalation**. All delivery methods use the **exact same code path** and they are not all subject to the escalation. For instance, `lmtp` allowed **unprivileged** execution of commands whereas `maildir` is **unimpacted**.

What made the escalation possible is a special case that OpenSMTPD has to handle **specifically** for `mbox`, and which opens the door to escalation if a bug manages to be exploited in that (relatively small) code path.

The fix

When I received the advisory draft, I hurried to produce a fix as soon as possible then ended with two diffs that I showed Qualys: a quick fix that just plugged the hole, and a better fix that plugged the hole by rewriting the `smtp_mailaddr()` function differently.

The quick fix is straightforward and can be implemented as a one-liner, it just tested `valid_localpart()` again when handling the missing domain case:

```
--- src/usr.sbin/smtpd/smtp_session.c   2019/10/04 08:34:29     1.415
+++ smtp_session.c      2020-01-30 08:51:31.000000000 +0100
@@ -2247,7 +2247,7 @@ smtp_mailaddr(struct mailaddr *maddr, ch
                        return (0);

                /* no domain, local user */
-               if (maddr->domain[0] == '\0') {
+               if (valid_localpart(maddr->user) && maddr->domain[0] == '\0') {
                        (void)strlcpy(maddr->domain, domain,
                            sizeof(maddr->domain));
                        return (1);
```

The better fix however rearranged the function in a different way that is less error-prone, and less likely to reintroduce similar mistakes. It did so by not grouping validators into a common block, but handling each case separately:

```
===================================================================
RCS file: /cvs/src/usr.sbin/smtpd/smtp_session.c,v
retrieving revision 1.415
retrieving revision 1.415.2.1
diff -u -r1.415 -r1.415.2.1
--- src/usr.sbin/smtpd/smtp_session.c   2019/10/04 08:34:29     1.415
+++ src/usr.sbin/smtpd/smtp_session.c   2020/01/28 21:39:20     1.415.2.1
@@ -1,4 +1,4 @@
-/*     $OpenBSD: smtp_session.c,v 1.415 2019/10/04 08:34:29 gilles Exp $       */
+/*     $OpenBSD: smtp_session.c,v 1.415.2.1 2020/01/28 21:39:20 gilles Exp $   */

 /*
  * Copyright (c) 2008 Gilles Chehade <gilles@poolp.org>
@@ -2215,24 +2215,22 @@
                memmove(maddr->user, p, strlen(p) + 1);
        }

-       if (!valid_localpart(maddr->user) ||
-           !valid_domainpart(maddr->domain)) {
-               /* accept empty return-path in MAIL FROM, required for bounces */
-               if (mailfrom && maddr->user[0] == '\0' && maddr->domain[0] == '\0')
-                       return (1);
+       /* accept empty return-path in MAIL FROM, required for bounces */
+       if (mailfrom && maddr->user[0] == '\0' && maddr->domain[0] == '\0')
+               return (1);

-               /* no user-part, reject */
-               if (maddr->user[0] == '\0')
-                       return (0);
-
-               /* no domain, local user */
-               if (maddr->domain[0] == '\0') {
-                       (void)strlcpy(maddr->domain, domain,
-                           sizeof(maddr->domain));
-                       return (1);
-               }
+       /* no or invalid user-part, reject */
+       if (maddr->user[0] == '\0' || !valid_localpart(maddr->user))
+               return (0);
+
+       /* no domain part, local user */
+       if (maddr->domain[0] == '\0') {
+               (void)strlcpy(maddr->domain, domain,
+                       sizeof(maddr->domain));
        }
+
+       if (!valid_domainpart(maddr->domain))
+               return (0);

        return (1);
 }
```

There is room for improvement and I think this function should be rewritten in a different style to make it harder to introduce bugs. But this version works, it was tested, proof-read by multiple people (including the auditors), and since it is unlikely to change before a long time (last change was in 2014), I will tackle it with a clear mind, not right now while still stressed.

Fixing the bug like I did plugs the hole, but that's not enough for me to be relaxed, it closes the door to the escalation but doesn't **build a wall of concrete in front of it**. Today, there's no known bug that affects this code path and we may be fine, but unless we actually kill the **potential** for escalation, we'll never be sure that researches won't find another bug and another creative way to exploit it.

In this article, I'll discuss mitigation mechanisms that I think should be put in place to avoid this from happening again, regardless of what bugs are found in processes exposed to attackers. Their goal is to deal with the **potential** for escalation, not with specific bugs.

My post-mortem thoughts

Even though fixing `smtp_mailaddr()` plugs the hole, **a bug in the smtp layer should not have been able to result in such a catastrophic outcome**, **regardless of the bug**. Today, with the combination of `pledge()`, privileges separation (smtp engine runs as unprivileged user `_smtpd`), filesystem isolation (smtp engine is `chroot()`-ed to `/var/empty`), and several internal consistency checks that will `fatal()` if something fishy is detected, misbehaving code is constrained and we expect limited consequences. My take has always been that I can't code bug-free so I **expect that bugs will be found**, and I want OpenSMTPD to **crash** in such cases so that I can fix the bug and roll a release. A crash in SMTP land is really not a big deal, protocol is fault-tolerant and other mail exchangers have retry logic, I can live with that.

The incident here was **not caused by misbehaving code**, it was caused by a **logic error** and **the code did what it was written to do**: it didn't corrupt memory and dodged sanitisation to craft an IPC message all the way to an exploit in another process. This is not a problem caused by the language, likes some people imply, and not a problem caused by a memory corruption of some sort. The sanitisation function **incorrectly validated the input** and from that point, **the input followed the exact same code path as any other valid input** until it reached the mda layer. **This is not a failure of the mechanisms put in place to mitigate misbehaving code**, and I think it is important to consider it for what it is: a correct implementation of an incorrect logic.

Seeing it as such isn't a way to shrug it off, quite the opposite. If we consider the incident without the human factor and try to address this with misbehaving code detection, it is bound to fail because... **the code did exactly what it was written to do**. We may slap a few sanity checks here and there but if they're supposed to check the same thing, they will have the same mistake. We can't fix my stupid with checks.

We can't prevent human mistakes, they will happen because tools won't help spot that a human-described logic is flawed. What we need is to make changes so that **OpenSMTPD becomes more resistant to human errors**. In other words, we need safe-guards that are not dependant on sanity checks and input, we need safe-guards that will guarantee that **even** if OpenSMTPD lets completely untrusted input pass through, this will have the most limited consequences... **then** we ensure that it doesn't let untrusted input pass through.

What made the exploit possible is the logic mistake in validation code and the use of `system()` with that invalid input, but what made the **escalation** possible is the `mbox` delivery method.

The problem(s) with `mbox` delivery

When a delivery takes places, OpenSMTPD does the delivery on behalf of a recipient user. Regardless of the delivery method, it will create a new mda process that will handle the delivery and drop its privileges to that of the recipient user. This is meant to ensure that a user adding a dangerous command in their `.forward` files can only impact themselves.

Furthermore, OpenSMTPD strictly **forbids** execution of an mda command with a privileged account: **root is not allowed to receive mail with `maildir` or `lmtp`**, the delivery will be aborted with an error message that **delivery to root is forbidden without an alias to an unprivileged user**. The mda command is not processed. This is what saves `maildir` and `lmtp` from being subjected to the escalation: **they cannot be ran from a privileged process**, bugs will **at most** impact the recipient user which **can never be root**. Not that it's good, but it's an order of magnitude less bad.

In the case of `mbox` however we can't do that.

First of all, if we take a fresh install, we can't assume that an unprivileged user exists since a user might not have created one at install. In this situation, if `mbox` refuses delivery to root then mails regarding `security` or `daily/weekly/monthly` will not be deliverable. Forbidding delivery to root implies requiring the creation of a user that root can be aliased to. For this reason, we can't easily switch from `mbox` to another delivery method either since they would enforce the restriction, breaking mail out of the box.

Then, unlike `maildir` which writes to a subdirectory of the recipient user and can therefore do it with the recipient user privileges, `mbox` writes to a common directory, usually `/var/mail` or `/var/spool/mail`. The mailbox is user-owned, so that `/var/mail/gilles` can be written or read by `gilles`, but since `gilles` cannot be allowed to create his own mailbox in the common directory, the creation has to be done by a privileged process that can both create the mailbox AND change its ownership.

Since the `mbox` format doesn't support concurrent deliveries, it also requires some locking to avoid concurrent writes. This locking is historically done by creating a `gilles.lock` file in the common directory. Mail user agents expect this. This whole delivery logic is handled outside of the daemon by `mail.local`, a delivery agent that preexists OpenSMTPD and which we inherited from the previous era. The downside is that `mail.local` **requires being called with privileges** to do the work and handle privileges dropping itself.

This is where the potential for escalation resides.

When `mbox` delivery is set, OpenSMTPD **HAS** to create a privileged process to execute `mail.local` no matter what user will receive the mail. It therefore has an explicit special case that forking a privileged mda is forbidden... **except** for `mbox`. Even if root had a `~/.forward` explicitely calling `mail.local`, it would be be forbidden.

The code path is relatively small: the mda process is created with privileges, `mail.local` is executed and will **internally lower its privileges**. Unfortunately for us, the exploit basically managed to replace the execution of `mail.local` with the execution of an arbitrary command, sitting in between the privileged process being `fork()`-ed and `mail.local` dropping privileges.

The **source of the problem** is that unlike other delivery methods, when using the `mbox` delivery method OpenSMTPD is **required** to create this privileged process. In this case the exploit **targeted** the way we executed the mail delivery agent but, thinking further, if an exploit was found that managed to abuse anything between the `fork()` and the execution of `mail.local`, we **could** be in a similar situation using a different attack vector: during a small period of time, we have a **privileged process** with a **potential** for being exploited somehow.

This is why I believe plugging the hole is **not enough**. It solves the **immediate problem** but not the **underlying issue**, it doesn't kill the **potential** for escalation: **if root is allowed to receive mail in a root-owned mailbox** then at some point a privileged process had to do the delivery.

The **only** way to kill this potential for escalation is to get rid of the privileged mda process. No special case, no nothing. If we don't have a privileged mda process, then a bug may exist and an exploit may find its way into the process, but it won't allow escalation.

This is where we must be heading towards and it'll require some creative solutions to work around `mbox` limitations.

Several ideas

Since this blew up to my face, I had several ideas to tackle this. Some were already discussed and not retained because they had potential for other issues. The current ideas are these:

- switching back mail delivery agents to `execle()`
- disallowing delivery to root
- managing to somehow run `mail.local` without privileges
- mbox creation and locking in OpenSMTPD to allow user execution of `mail.local`

Not each of them is straight-forward but each of them would have avoided the escalation, regardless of the bug being present, so I think these are critical to implement in order to avoid future bugs from having the kind of impact we experienced.

Switching back mail delivery agents to `execle()`

This doesn't solve the underlying issue of having a privileged process for `mbox`, but it would have prevented this specific attack.

Moving **all** mail delivery agents to `execle()` is tricky but we're going to got that way. I have already written a diff to switch `mbox` and it will be committed shortly. The `maildir` and `lmtp` will be similarly converted.

The only issue with that move is that some mail delivery agents expect a command-line interface that's more complex than ours:

```
action foobar mda "/usr/local/bin/procmail -a -ton -of -options \"with quotes and escaping\""
```

or it's `~/.forward` variant:

```
|/usr/local/bin/procmail -a -ton -of -options "with quotes and \"escaping\""
```

or even it's `/etc/mail/aliases` variant:

```
lists:          |/usr/local/bin/mailing-list.py -a -ton -of -options
```

I doubt anyone wants to see these go away, and switching them to `execle()` implies reimplementing a command-line parser with shell-like semantics, then using the parsed command line to call the `execle()` function. This is the literal definition of rewriting a shell and I'd rather rely on a standard shell than rewriting my dumbed-down one.

Since these commands can **never run with privileges**, what I suggest is that we use a wrapping mail delivery agent with a simple interface that we can easily plug through `execle()`, then let THAT mail delivery agent run the mda command through a shell. This allows us to have a single code path through `execle()` for all mail delivery agents in the daemon, and restrict the `system()` call to specific use-cases handled by a specific mail delivery agent outside the daemon.

Disallowing delivery to root

No need for much explanations here: this would have effectively prevented the privileges escalation despite the logic bug.

I think we should **disallow deliveries to privileged users unconditionally**. I never thought it was a good idea to accept doing it and this is why I disallowed it for everything but `mbox`. This is going to be inconvenient to many because it means that root **HAS** to be an alias, but everyone using `maildir` or `lmtp` has been in this mode for years now and I never had a complaint about it. We can't tell people not to log in as root and use `doas` or `sudo`, then ensure that everything is convenient for them to log in as root and have their mails delivered knowing the risks. It doesn't make sense.

This will put us at odds with other MTA implementations that accept delivering mail to root, and every time I suggested doing something differently from other MTA people got angry at me for not being able to keep their habits. But now that this blew to my face I don't think there's an excuse for

keeping a time-bomb that we know can blow hard. I hope there will be no resistance to this change, but at some point, if we refuse to diverge from dangerous practices it means we are knowingly leaving potential for escalation. Someone else will have to defend why this is a good idea.

While at it...

On the somewhat related side-note of doing something I dislike because other MTA do it and we shouldn't diverge, supporting commands in aliases is also a bad idea from a security point of view. It doesn't come with the same level of risks, but it also opens the potential for abuse and I've been willing to get rid of that for years. I don't know of ANY use-case of executing a command from aliases that can't be solved from a `~/.forward` file in a safer way. It's less convenient and it requires a dedicated user, but it's safer.

```
/* this battle needs to be fought ... */
if (xn->type == EXPAND_FILTER &&
    strcmp(ep->mda_user, SMTPD_USER) == 0)
        log_warnx("commands executed from aliases "
            "run with %s privileges", SMTPD_USER);
```

managing to somehow run `mail.local` without privileges

Disabling deliveries to privileged users is the right thing to do, however this is not sufficient if `mail.local` still requires privileges to deliver to `gilles`. The problem is the requirement for privileges during delivery, not the fact that a user itself is privileged.

I've already explained the problem with `mail.local` and why it requires privileges. We're discussing with other OpenBSD hackers the option to change permissions of `/var/mail`, the option to use a *setgid* `mail.local` and we'll continue exploring similar options.

The idea for a group-writeable mail spool was apparently rejected in the past but it may be time to reassess. I don't really have a good intuition about doing this, I don't know why but I've been wrong before so it's worth investigating this option with the help of other developers.

The bottom line is that **unless `mail.local` can be started without privileges**, no effort to reduce the potential for escalation will succeed. At the same time, we need to keep in mind that `mail.local` is not an OpenSMTPD specific component, and changing it must be done with care to avoid breaking other software. If managing to use `mail.local` without privileges requires making changes to it, we may have to implement an OpenSMTPD specific implementation.

mbox creation and locking in OpenSMTPD to allow user execution of `mail.local`

Finally, a proposal I discussed with other hackers was to have OpenSMTPD handle the mailbox creation and locking.

We already consider `mbox` as a special case and we could convert that special case from "needs privileges" to "needs locking". In this scenario, when a delivery uses `mbox`, the daemon can do the locking and mailbox creation **itself** before dropping privileges and executing `mail.local` unprivileged, like any other mail delivery agent. The lock would be removed upon *SIGCHLD* of the mail delivery agent. This would effectively allow removing the reasons for which `mail.local` requires privileges.

As for `mail.local`, all it would need to do is allow execution from unprivileged user by removing the **explicit check for root** it has today. This would still allow to use it with other MTA that expect it to be privileged, while allowing it to run unprivileged with OpenSMTPD. As for locking, no changes would be required since already supports skipping locking through the `-L` option.

Some people see this as doing part of the delivery in OpenSMTPD and part in `mail.local`. I disagree and see this as executing a mail delivery agent in locking barriers: the daemon sets a locked environement around mail delivery agent execution, the mail delivery agent doesn't know or care about the locks. This is word plays but seeing it this way makes more sense to me and doesn't sound hackish at all.

This coupled with preventing delivery to root would effectively remove the potential for escalation in `mbox`.

Closing words

I don't know where to start, this section will not be as structured as previous ones.

First of all, I'd like to **thank the people at Qualys**. They found a critical bug that could have remained out of sight until exploited by less friendly people. I won't say it is a pleasure to be on my side of the advisory but as unpleasant as it is, **it makes the software safer on the long run**. I hope they don't find something else, and if they do I hope it's not something as big, but it is a **good thing for users** to have such competent people studying the code, as painful as it is for the developers when they find something.

Then, people should use this as a reminder not to put **too much** trust in it. An MTA is a piece of software that can't drop all privileges, otherwise it would already be done, and at the same time a software that is highly exposed to user-supplied data from untrusted sources. Having an MTA, whichever it is, exposed to the wild and not running on its very own machine will always be a risk. As careful as I can be, I'll make errors and some of these errors will slip in. The only thing I can promise is that I always handle them the way I handled this one, not just fixing the issue but trying to kill the potential.

That being said, I will continue discussing the ideas from this article and possibly others to start implementing them as soon as possible. My goal is to kill the potential for escalation in `forkmda()` **before the next major release**. I have a few weeks in front of me and there may be some very quick wins to do here that would remove the existing risks.

Comments: https://github.com/poolpOrg/poolp.org/discussions/117

« Anxiety, OpenBSD break, COVID-19 and resuming work

January 2020: OpenSMTPD work - libasr and libtls »

1 reaction

19 comments – *powered by giscus*

Oldest Newest

Write Preview Aa

Mastodon © Copyright poolp.org - All rights reserved

Mediumish Theme by WowThemes.net