From: "Gilles Chehade" <gilles@openbsd.org>
To: "Ah, Jacques Cousteau" <ajacoutot@openbsd.org>
Date: Sun, 24 Sep 2017 CET
Subject: OpenSMTPD, current state of affairs

The plan

- Made tons of slides, I'll just skip over some if needed...
- Sick AF, may need to run out during the talk…
- Should this happen, stay calm and don’t panic, I’ll be back :-)

$ whoami

- Gilles Chehade <gilles@openbsd.org>
  - I'm also **@poolpOrg** on twitter and github
  - I live in the beautiful city of Nantes, France (west coast riprizent !)

- OpenBSD user since 2.6 (1999), OpenBSD developer since 4.2 (2007)
  - Also used NetBSD and FreeBSD a lot in the past, I enjoyed all BSD systems

- Started working on smtpd in 2007 as personal project for my own needs

- **pyr@**, **reyk@** and **henning@** tricked me into turning it into OpenSMTPD
  - "It will be fun", they said with a grin.

$ whoami

- Currently a Lead-Developer for the Vente-Privée Group 
  - Platinum Sponsor of EuroBSDCon 2017 btw, woohoo !
  - We are hiring. We are hiring. We are hiring. We are hiring. Mail me ;-)
- I no longer do R&D work in the mail industry
  - Still do mail experiments in private though ;-)
  - My daily job has NOTHING to do with mails whatsoever, no conflicts of interest
- Vente-Privée has a few OpenSMTPD instances as main MTA
  - I wasn't sure, I had to ask Miky Mike, the guy who knows this stuff
  - We also have a few OpenBSD installs, not sure I can say where and why, so…

The OpenSMTPD crew

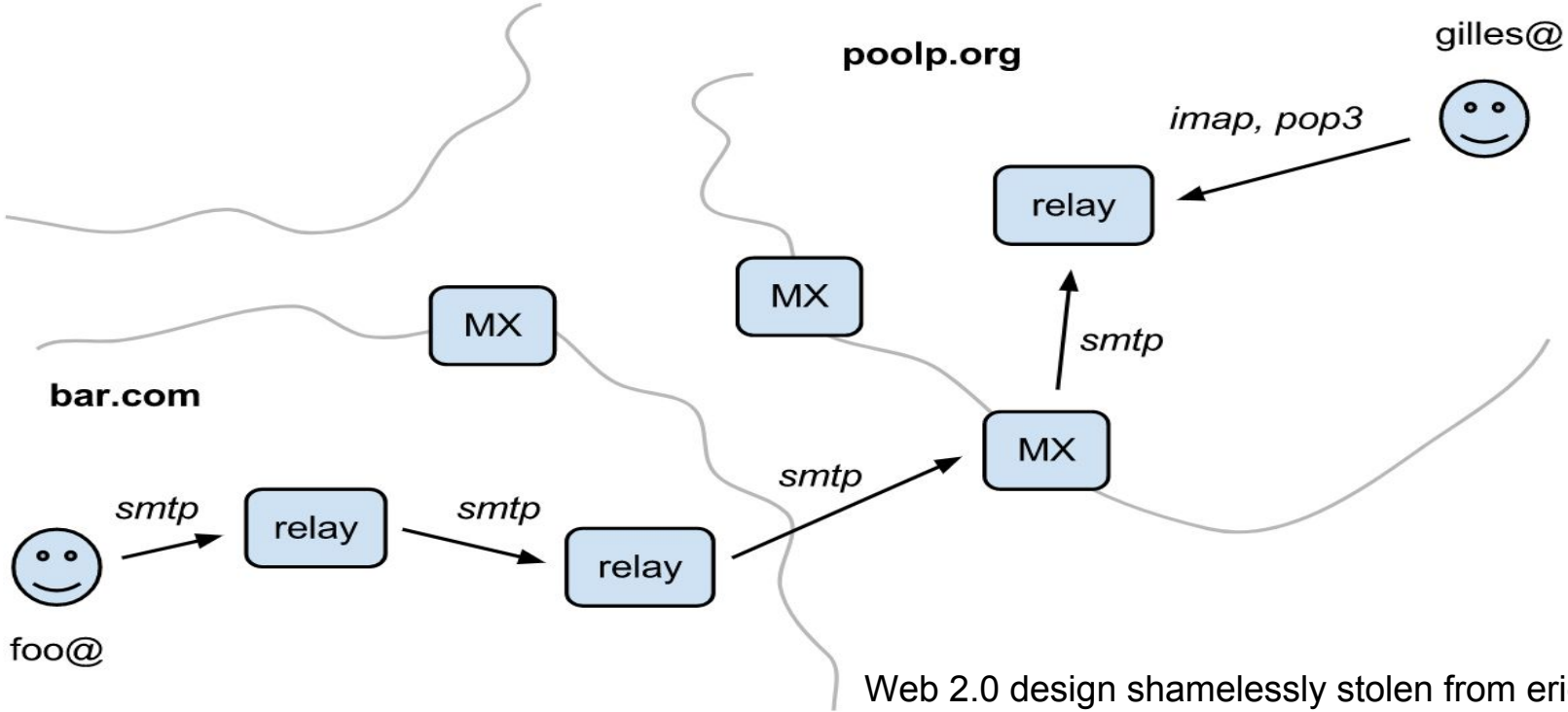
- Eric Faurot <eric@openbsd.org> aka "The Doctor"
- Sunil Nimmagadda <sunil@openbsd.org>
- Jung Joerg <jung@openbsd.org>
- We tend to cc: our diffs to Todd Miller <millert@openbsd.org>
- We receive **a few** contributions from the community
  - Mostly Linux people, just saying...
  - Not many "stable" contributors, people come and go

What is SMTP and OpenSMTPD ?

What is SMTP ? (dramatic vulgarization)

- SMTP is a "simple protocol" to exchange messages between machines
  - It only takes in charge the transfer, retrieval for users is done with POP/IMAP/...
- SMTP relies heavily on the DNS protocol to locate MX (Mail eXchangers)
- Each MX is a node on the SMTP graph operating as destination or relay
- The goal of an MX is to route a message closer to its destination
- Each MX sees the next hop as a destination, it doesn't see what's behind

What is SMTP ? (dramatic vulgarization)



Web 2.0 design shamelessly stolen from eric@

What is SMTP ? (dramatic vulgarization)

- In addition, RFC imposes responsibility over messages
- You're not allowed to lose a message that you accepted
  - No "ooopsie, I tripped over the cable at the very wrong time" is allowed
- If you can't relay for some reason, the sender must be acknowledged
  - And the hop on which the error occurred is responsible for this
- Each MX has an interest in getting message out of its way fast ;-)
  - Who wants to handle angry users who lost their mail due to a disk crash … not me.
- The SMTP protocol is **transactional**, accepted message is a **commit**.

What is SMTP ? (dramatic vulgarization)

```
220 poolp.org ESMTP OpenSMTPD
EHLO mx1.poolp.org                                          # just saying hi
250-poolp.org Hello mx1.poolp.org [127.0.0.1], pleased to meet you
250-8BITMIME
250-ENHANCEDSTATUSCODES
250-SIZE 36700160
250-DSN
250 HELP
MAIL FROM:<gilles@openbsd.org>                              # starting the transaction
250 2.0.0: Ok
RCPT TO:<eric@openbsd.org>                                  # part of the transaction
250 2.1.5 Destination address valid: Recipient ok
RCPT TO:<deraadt@openbsd.org>                               # part of the transaction
250 2.1.5 Destination address valid: Recipient ok
DATA                                                        # beginning DATA exchange
354 Enter mail, end with "." on a line by itself
[insert lame attempt at passing one more commit during OpenBSD freeze here]
.
250 2.0.0: 990b7169 Message accepted for delivery           # server assigned a transaction ID
```

What is SMTP ? (dramatic vulgarization)

- The session spans from connection to disconnection
- The transaction spans from MAIL to '.' and can be interrupted
  - client: MAIL and RSET commands trigger a transaction rollback
  - server: rejected sender or rejected message trigger a transaction rollback
- Recipients in the same transaction share a common message identifier
- They will all receive the same copy of the message
- You can have multiple transactions within the same session

What is OpenSMTPD ?

- OpenSMTPD is a **general purpose** implementation of the SMTP protocol
- It features both server-side and client-side implementations
- Operating as a server, it accepts messages from SMTP clients
- Operating as a client, it relays messages to other SMTP servers
- It has some other duties too as we'll see soon
- Doesn't follow the OpenBSD release cycle but aligns to it

What is OpenSMTPD ?

- OpenSMTPD summarized in a few lines:
  - Accepts messages from a unix socket or the network for delivery or relaying
  - Resolves SMTP addresses into recipients (aliases, virtual, …)
  - Manages a local persistent queue of messages that aren't allowed to be lost
  - Schedules delivery with some retry logic in case of temporary failures
  - Relays messages to other hosts over the network
  - Delivers messages locally by executing a MDA with the user privileges

What is OpenSMTPD ?

- We have a nice community with helpful people on IRC and our mailing-list
  - #opensmtpd @ freenode, misc@opensmtpd.org
- We've been packaged to several systems and distributions
  - We don't really track this
  - Discovered three Linux distros in the last two years due to a maintainer contacting me
- People choosing an MTA today give OpenSMTPD a try
  - **That was not the case about two years ago**
  - People now say "you should try postfix or opensmtpd" on forums
  - We have the benefit of simplicity: we often win when use-case matches
  - Sometimes we lack a feature, it's ok, no royalties on installs ;-p

It's been a while...

It's been a while...

- Last talk was made by eric@ in 2013 at AsiaBSDCon
  - We announced our first production-ready release during his talk
- Back then, we had a sponsorship and worked on OpenSMTPD full-time
  - **THANKS:** Antoine "Rocky" Millet and Julien "Flop" Mangeard :-*
- It was very comfortable, we could spend days improving an area
  - We only had one goal: make OpenSMTPD awesome and rock-solid
- I can't even list how many things were done during that time
  - We could do in a week what we currently do in many months !

A bit of context

- Worked 5 years with eric@ on a scalable MTA for an ESP
  - Dealing with many millions of mails daily on a cluster of Postfix
- Is OpenSMTPD capable of simplifying the architecture ?
  - It simplified it _a lot_, complex multi-machine config became 20 lines.
- Is OpenSMTPD capable of dealing with same volumes ?
  - We proved it could, but we still needed more time to optimize and stabilize.
- Four months later, ops were confident enough to switch infrastructure
  - We then had an OpenSMTPD instance dealing with large volumes of real-world traffic

A bit of context

- 1½ year later, a general purpose MTA is not the right tool for this job
  - Let's cut a steak with a spoon !
- We could continue trying to make OpenSMTPD fit
  - And keep the sponsorship going
- Or write a custom MTA tailored for the need
  - And put an end to the sponsorship
- We went for the second solution
  - We didn't want to influence OpenSMTPD decisions because of the sponsorship
  - Hard decision but no real debate, we both agreed it was better for the project

A bit of context

- During the sponsored OpenSMTPD development
  - We ran in a very high-volume environment, we wrote code and operated it
  - Bugs that triggered every few weeks in the community would trigger in minutes
  - We hit a lot of interoperability issues and fixed them until rock-solid
  - We hit a lot of bottlenecks far beyond the realm of regular users and fixed them
  - We had to optimize everything from memory usage to disk access or space

- No one interfered with our work, it was cool
  - We had to find solutions to real use-cases and issues
  - We decided how we wanted to solve these issues with no pressure
  - SMTP is nice for failing gracefully so we could work on real solutions, not hacks
  - Everything fixed for the ESP came back to the community, no special branch

A bit of context

- After sponsorship ended, we had a long period of calm
  - occasional commits, mostly bug fixes for community reported issues
- Hard to go back from full-time to a few hours here and there
  - starting large refactor is hard because you know it'll be interrupted and take longer
  - dissatisfying because you know in advance something is going to take many weeks
- We also needed a small break from working 100% in mail area
  - OpenSMTPD set aside, it was still our day time job to develop and operate an MTA...
- Work resumed after a few months though because it's still fun ;-)
  - not the same goals, a very nice community, a testbed for experiments

A bit of context

- We wrote another MTA for high-volumes, not general purpose
- Some ideas were brought back to OpenSMTPD
- Some others couldn't because made no sense for a general purpose MTA
- We learnt a lot from this
  - Re-factors in progress result from lessons learnt in the last few years
  - We can make some areas MUCH simpler because we no longer "assume" high volumes
- OpenSMTPD will benefit from all this non-OpenSMTPD work indirectly


OpenSMTPD
MAKE E-MAIL
GREAT AGAIN

Make Email Great Again

- In my opinion, our design is quite good today, it evolved with experience
  - I would have said "design goes in a good direction" a few years ago
  - If started today, I'd change a lot of things but keep the design we have now
- We made many improvements over the years, moved code here and there
- We still have areas to improve but it's mostly code patterns not design
- We survived a harsh audit and the design saved us from a catastrophe
- Since then, we improved further

Make Email Great Again

- OpenSMTPD is a multi-process daemon:

```
root      48840  0.0  0.1  2176  2528 ??  Isp   28Aug17    0:18.97 smtpd
_smtpd    13940  0.0  0.1  1828  4748 ??  Ip    28Aug17    0:16.12 smtpd: control (smtpd)
_smtpd    56434  0.0  0.1  1960  5276 ??  Ip    28Aug17    6:44.55 smtpd: lookup (smtpd)
_smtpd    13243  0.0  0.1  1588  4548 ??  Ip    28Aug17    6:29.29 smtpd: klondike (smtpd)
_smtpq    87444  0.0  0.1  1908  4608 ??  Ip    28Aug17    2:18.37 smtpd: queue (smtpd)
_smtpd    74308  0.0  0.1  1556  4500 ??  Ip    28Aug17    0:11.92 smtpd: scheduler (smtpd)
_smtpd    21903  0.0  0.2  2948  6696 ??  Ip    28Aug17   11:10.60 smtpd: pony express (smtpd)
```

- Privileges separation in action !

- Each process has a well-defined set of tasks to achieve

Make Email Great Again

- Except for parent, there's no privileged process
  - Can't kill privileged process, system auth and privileges dropping of MDA require root
- Except for lookup and parent, all processes are chrooted
  - Can't chroot() lookup, we need an unprivileged process that can access lookup ressources
- Queue and other processes run as different users
  - A bug in control, lookup or pony processes can't just go rogue and delete the queue
- IPC with the imsg framework, no shared memory between processes
- Processes are pledged and go through fork and re-exec as we'll see

Make Email Great Again

- Most OpenSMTPD core features are API-zed (that's a verb right ?)
  - OpenSMTPD only access the feature through a defined API with a few primitives
  - It allows us to test new code, developers can adapt OpenSMTPD to specific needs

```
377  struct table_backend {
378          const unsigned int      services;
379          int     (*config)(struct table *);
380          void   *(*open)(struct table *);
381          int     (*update)(struct table *);
382          void    (*close)(void *);
383          int     (*lookup)(void *, struct dict *, const char *, enum table_service, union lookup *);
384          int     (*fetch)(void *, struct dict *, enum table_service, union lookup *);
385  };
```

Make Email Great Again

- We have a "table" API to perform all kinds of lookups
  - You can implement a new driver to support lookups over whatever backend you want
  - We provide glue code that handles the plumbing
  - You write the primitives for the "table" API and end up with a standalone libexec
  - OpenSMTPD doesn't care what your dependencies are, it talks imsg with that libexec
  - All libexec drivers benefit from ASLR, can have different privileges and chroot dirs

- We have also have a "queue" and a "scheduler" API
  - Same as above applies
  - It is trivial to delocalize queue to any key-value store, not necessarily local
  - Scheduler is a bit trickier, API is not frozen yet, it kind of works though
  - **Insert anecdote about queue_cloud PoC here**

Make Email Great Again

```
514  int
515  main(int argc, char **argv)
516  {
517          int ch;
518
519          log_init(1);
520          log_verbose(~0);
521
522          while ((ch = getopt(argc, argv, "")) != -1) {
523                  switch (ch) {
524                  default:
525                          fatalx("bad option");
526                          /* NOTREACHED */
527                  }
528          }
529          argc -= optind;
530          argv += optind;
531
532          if (argc != 1)
533                  fatalx("bogus argument(s)");
534
535          conffile = argv[0];
536
537          if ((config = config_load(conffile)) == NULL)
538                  fatalx("error parsing config file");
539          if (config_connect(config) == 0)
540                  fatalx("could not connect");
541
542          table_api_on_update(table_redis_update);
543          table_api_on_check(table_redis_check);
544          table_api_on_lookup(table_redis_lookup);
545          table_api_on_fetch(table_redis_fetch);
546          table_api_dispatch();
547
548          return 0;
549  }
```

```
498  int
499  main(int argc, char **argv)
500  {
501          int ch;
502
503          log_init(1);
504          log_verbose(~0);
505
506          while ((ch = getopt(argc, argv, "")) != -1) {
507                  switch (ch) {
508                  default:
509                          fatalx("bad option");
510                          /* NOTREACHED */
511                  }
512          }
513          argc -= optind;
514          argv += optind;
515
516          if (argc != 1)
517                  fatalx("bogus argument(s)");
518
519          conffile = argv[0];
520
521          if ((config = config_load(conffile)) == NULL)
522                  fatalx("error parsing config file");
523          if (config_connect(config) == 0)
524                  fatalx("could not connect");
525
526          table_api_on_update(table_postgres_update);
527          table_api_on_check(table_postgres_check);
528          table_api_on_lookup(table_postgres_lookup);
529          table_api_on_fetch(table_postgres_fetch);
530          table_api_dispatch();
531
532          return 0;
533  }
```

```
449  int
450  main(int argc, char **argv)
451  {
452          int ch;
453
454          log_init(1);
455          log_verbose(~0);
456
457          while ((ch = getopt(argc, argv, "")) != -1) {
458                  switch (ch) {
459                  default:
460                          fatalx("bad option");
461                          /* NOTREACHED */
462                  }
463          }
464          argc -= optind;
465          argv += optind;
466
467          if (argc != 1)
468                  fatalx("bogus argument(s)");
469
470          config = argv[0];
471
472          dict_init(&sources);
473
474          if (table_sqlite_update() == 0)
475                  fatalx("error parsing config file");
476
477          table_api_on_update(table_sqlite_update);
478          table_api_on_check(table_sqlite_check);
479          table_api_on_lookup(table_sqlite_lookup);
480          table_api_on_fetch(table_sqlite_fetch);
481          table_api_dispatch();
482
483          return 0;
484  }
```

Make Email Great Again

```
349  static int
350  queue_python_init(int server)
351  {
352          queue_api_on_message_create(queue_python_message_create);
353          queue_api_on_message_commit(queue_python_message_commit);
354          queue_api_on_message_delete(queue_python_message_delete);
355          queue_api_on_message_fd_r(queue_python_message_fd_r);
356          queue_api_on_message_corrupt(queue_python_message_corrupt);
357          queue_api_on_message_uncorrupt(queue_python_message_uncorrupt);
358          queue_api_on_envelope_create(queue_python_envelope_create);
359          queue_api_on_envelope_delete(queue_python_envelope_delete);
360          queue_api_on_envelope_update(queue_python_envelope_update);
361          queue_api_on_envelope_load(queue_python_envelope_load);
362          queue_api_on_envelope_walk(queue_python_envelope_walk);
363          queue_api_on_message_walk(queue_python_message_walk);
364
365          return 1;
366  }
```

Here's a transition duckling

Make Email Great Again

- We don't know where we'll implement our next bug
- We don't know how attackers will hit us
- Design takes this into account
  - Within each process, assume remote code execution and turn this into a non-problem
  - Mitigate the impact of a bug or a catastrophe
- We prefer a DoS to RCE and privileges escalation
  - Make it hard to corrupt code path or memory without bringing the daemon down
  - Do not restart automatically a process after a crash to give attacker a free new attempt
  - RAISE THE BAR.

Make Email Great Again

- imsg framework allows message passing between OpenSMTPD processes
- We built a new interface on top of it: **mproc**
- Basically adds types and checks to messages exchanged over imsg
  - MUST deserialize correctly (no remaining data, missing data, invalid data for type)
  - **Anything fishy hints a memory corruption or a logic bug and triggers a fatal**

```
2455  static void
2456  smtp_queue_create_message(struct smtp_session *s)
2457  {
2458          m_create(p_queue, IMSG_SMTP_MESSAGE_CREATE, 0, 0, -1);
2459          m_add_id(p_queue, s->id);
2460          m_close(p_queue);
2461          tree_xset(&wait_queue_msg, s->id, s);
2462  }
```

```
79          case IMSG_SMTP_MESSAGE_CREATE:
80                  m_msg(&m, imsg);
81                  m_get_id(&m, &reqid);
82                  m_end(&m);
```

Make Email Great Again

- deraadt@ introduced the pledge() system call
  - Classify system calls into categories of related calls (stdio, inet, …)
  - Allow a process to restrict which categories are usable from a given point in execution
  - Abort processes that violate the restriction !

- A process only uses a restricted set of system calls at runtime
  - Event-driven daemons tend to use a larger set during setup than in event loop …
  - … so let them do their setup and restrict before entering event loop.

- A developer should know what system calls will be done from a given point
  - pledge() can confirm that knowledge (libraries may do stuff you didn't know)
  - It can also expose API layer violations (design errors in application)

Make Email Great Again

- We adapted to pledge() before it was cool (tame…)
  - Most of OpenSMTPD was pledged overnight
  - Pledges were a bit permissive at first & exposed a cases of "shouldn't be done here"
- Many people see pledge() as a security feature
  - If no pledge("exec"), how is a shellcode going to execute /bin/sh ?
- I see it as a code quality feature
  - Nope, this process SHOULD NOT have this pledge, so this code should be elsewhere
- Nowadays all processes are pledged with tight pledges
  - In some cases, restrictions increase several times as code unrolls

Make Email Great Again

```
292         /* check if working in offline mode */
293         /* If the server is not running, enqueue the message offline */
294
295         if (!srv_connected()) {
296                 if (pledge("stdio", NULL) == -1)
297                         err(1, "pledge");
298
299                 return (enqueue_offline(save_argc, save_argv, fp, ofp));
300         }
301
302         if ((msg.fd = open_connection()) == -1)
303                 errx(EX_UNAVAILABLE, "server too busy");
304
305         if (pledge("stdio wpath cpath", NULL) == -1)
306                 err(1, "pledge");
307
308         fout = fdopen(msg.fd, "a+");
309         if (fout == NULL)
310                 err(EX_UNAVAILABLE, "fdopen");
```

Make Email Great Again

- OpenBSD provides out-of-the-box ASLR and randomized malloc()
  - Every time you run OpenSMTPD, it has a different memory layout from previous run
  - Two children processes performing a malloc() will get a chunk at a different address

- For privileges separation, OpenSMTPD forks at beginning
  - Due to fork() semantics, memory layout is inherited from parent instance
  - Child process memory space starts diverging with malloc() calls
  - Global structures inherited from parent remain at same address in child
  - It's a feature, not a bug, we relied on it to have configuration inherited in children

- At Cambridge, deraadt@ caught both eric@ and I in a hallway...

Make Email Great Again

- It would be nice for children to re-exec /usr/sbin/smtpd after fork()
  - Each child process would get a different memory layout thanks to ASLR
  - This would randomize the position of a similar structures within different children
  - It would avoid potential issues like inheriting something we shouldn't
  - And avoid attacks based on knowledge of the address of a structure in another process

- OH GOSH. This is going to need a tricky refactor.
  - I was already under the water :sadface:

- eric@ came up with such a diff just two or three days later
  - "I have a small diff to show you" © Eric Faurot, 2016

Make Email Great Again

- Basically, at startup the parent process does the bootstrap:
  - Create a set of socketpair() needed for IPC
  - Fork a process and drop privileges for each child
  - Re-exec /usr/sbin/smtpd with the -x option so the child can resume configuration
- When done, all processes have completely different memory layouts
- Nothing but the socketpair() descriptors have been inherited
- In the future, some processes may possibly re-exec at runtime

Make Email Great Again

ASLR

+

random malloc

```
$ ./a.out
0x7f7ffffd4514  address of a stack variable
0x1061902ffe40  address of a malloc() allocation
0x105eeb000a64  address of a function within the program
0x105eeb000a6a  address of the main function
0x1060fcf67a90  address of a libc function

$ ./a.out
0x7f7ffffbb714  address of a stack variable
0xf7cbac277c0   address of a malloc() allocation
0xf7a05d00a64   address of a function within the program
0xf7a05d00a6a   address of the main function
0xf7c34ba9a90   address of a libc function

$ ./a.out
0x7f7ffffeab74  address of a stack variable
0x15b5c11cc5c0  address of a malloc() allocation
0x15b354800a64  address of a function within the program
0x15b354800a6a  address of the main function
0x15b5ab116a90  address of a libc function
$
```

Make Email Great Again

ASLR

+

random malloc

+

fork()

```
$ ./a.out
=== in parent
0x7f7ffffd3c44  address of a stack variable
0x8e8a457c040   address of a malloc() allocation BEFORE fork
0x8e8181631c0   address of a malloc() allocation AFTER fork
0x8e5ce100b34   address of a function within the program
0x8e5ce100c20   address of the main function
0x8e88231aa90   address of a libc function
=== in child
0x7f7ffffd3c44  address of a stack variable
0x8e8a457c040   address of a malloc() allocation BEFORE fork
0x8e8ad0311c0   address of a malloc() allocation AFTER fork
0x8e5ce100b34   address of a function within the program
0x8e5ce100c20   address of the main function
0x8e88231aa90   address of a libc function

$ ./a.out
=== in parent
0x7f7ffffc0684  address of a stack variable
0x1fc99fc64100  address of a malloc() allocation BEFORE fork
0x1fc9443a6280  address of a malloc() allocation AFTER fork
0x1fc704400b34  address of a function within the program
0x1fc704400c20  address of the main function
0x1fc97c85aa90  address of a libc function
=== in child
0x7f7ffffc0684  address of a stack variable
0x1fc99fc64100  address of a malloc() allocation BEFORE fork
0x1fc90bb1e280  address of a malloc() allocation AFTER fork
0x1fc704400b34  address of a function within the program
0x1fc704400c20  address of the main function
0x1fc97c85aa90  address of a libc function
$
```

Make Email Great Again

ASLR

+

random malloc

+

fork()

+

exec*()

```
$ ./a.out
=== in parent
0x7f7fffff6984  address of a stack variable
0x10a78e9cdc0   address of a malloc() allocation
0x10873500b84   address of a function within the program
0x10873500c57   address of the main function
0x10b6b748a90   address of a libc function
=== in child
0x7f7fffff8bc4  address of a stack variable
0x142a9fe228c0  address of a malloc() allocation
0x14289e900b84  address of a function within the program
0x14289e900c57  address of the main function
0x142ad94e0a90  address of a libc function

$ ./a.out
=== in parent
0x7f7ffffbdee4  address of a stack variable
0x1f673b59d700  address of a malloc() allocation
0x1f6482500b84  address of a function within the program
0x1f6482500c57  address of the main function
0x1f66b1b71a90  address of a libc function
=== in child
0x7f7ffffdcc74  address of a stack variable
0xe10ddc66e40   address of a malloc() allocation
0xe0ec9f00b84   address of a function within the program
0xe0ec9f00c57   address of the main function
0xe10d3a5fa90   address of a libc function
$
```

Make Email Great Again

- I originally wanted to go through all components of OpenSMTPD
  - Describe current state, cool features and stuff we want to improve
- I reached 120 slides and counting, so…
- … I deleted half of it
- Let's not bore you to death **yet**, there's a BOF after this talk ;-)

Code quality

Code quality

- Some error code paths are hard to test
  - FD exhaustion is easy, memory exhaustion is easy, disk space exhaustion is easy
  - Writing to a socket for a session that was suspended for an hour, no so much
- During development, we test by adding code we remove before commit
  - Turn a == into a != and watch the daemon turn into a neutron star
  - Return -1 right away when entering a function, watch the caller collapse
- Sometimes it is hard and bug in error code path is not obvious
  - Error may rely on moon phases to actually trigger
  - Or it may be a combination of many unlikely situations that you can easily overlook
  - Lost weeks on a libevent bug years ago...

Code quality

- smtpscript
  - An SMTP session scripting language written by eric@
  - Allows the writing of regression tests for an SMTP Server implementation

```
# this is a function init-helo that we want to call in all our regress tests
proc init-helo {
    expect smtp ok
    writeln "HELO regress"
    expect smtp helo
}

# each of the test-case will be called sequentially
test-case name "mailfrom.empty" {
    call init-helo
    writeln "MAIL FROM:<>"
    expect smtp ok
}

test-case name "mailfrom.broken" {
    call init-helo
    writeln "MAIL FROM:< @bleh>"
    expect smtp permfail
}
```

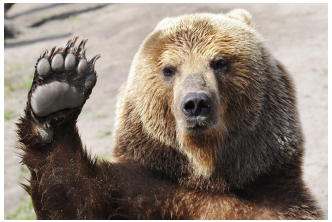
```
$ smtpscript foo
===> running test-case "mailfrom.empty" ok
===> running test-case "mailfrom.broken" ok
===> all run
passed: 2/2 (skipped: 0, failed: 0, error: 0)
$
```

Code quality

- We relied a lot on code review
  - Doing reading passes ourselves assuming everything went wrong on the system
  - What if disk went full between this call and this call within same function ?

- We relied a lot on static analysis tools
  - Scan-build helped uncover some issues, many false positives though
  - We ran coverity build which uncovered more issues, better results than scan-build

- We also relied on a particular branch I came up with
  - Bear with me a minute...



Coverity & scan-build

avoid possible NULL deref
poolpOrg committed on 29 Jan 2013

- missing free ...
poolpOrg committed on 29 Jan 2013

fix wrong type returned by internal call
poolpOrg committed on 29 Jan 2013

missing free
poolpOrg committed on 29 Jan 2013

fix use after free Browse files

portable · opensmtpd-6.0.2 ... opensmtpd-201301311831

ericfaurot committed on 29 Jan 2013 — 1 parent 48a80ee — commit 7c6c81dc574a9fd5f10417a3040282a325843bbf

Showing 1 changed file with 1 addition and 2 deletions. Unified Split

3 smtpd/mfa_session.c View

```
@@ -394,15 +394,14 @@ mfa_drain_query(struct mfa_query *q)
394 394                 free(q->smtp.response);
395 395         }
396 396
    397 +       TAILQ_REMOVE(&q->session->queries, q, entry);
397 398         /* If the query was a disconnect event, the session can be freed */
398 399         if (q->type == HOOK_DISCONNECT) {
399 400                 /* XXX assert prev == NULL */
400 401                 free(q->session);
401 402         }
402 403
403 404         log_trace(TRACE_MFA, "mfa: freeing query %016" PRIx64, q->qid);
404     -
405     -       TAILQ_REMOVE(&q->session->queries, q, entry);
406 405         free(q);
407 406  }
408 407
```

Masturbating Monkeys

"I think the OpenBSD crowd is a bunch of **masturbating monkeys** [...]"
-- Linus Torvalds

source: https://lkml.org/lkml/2008/7/15/296

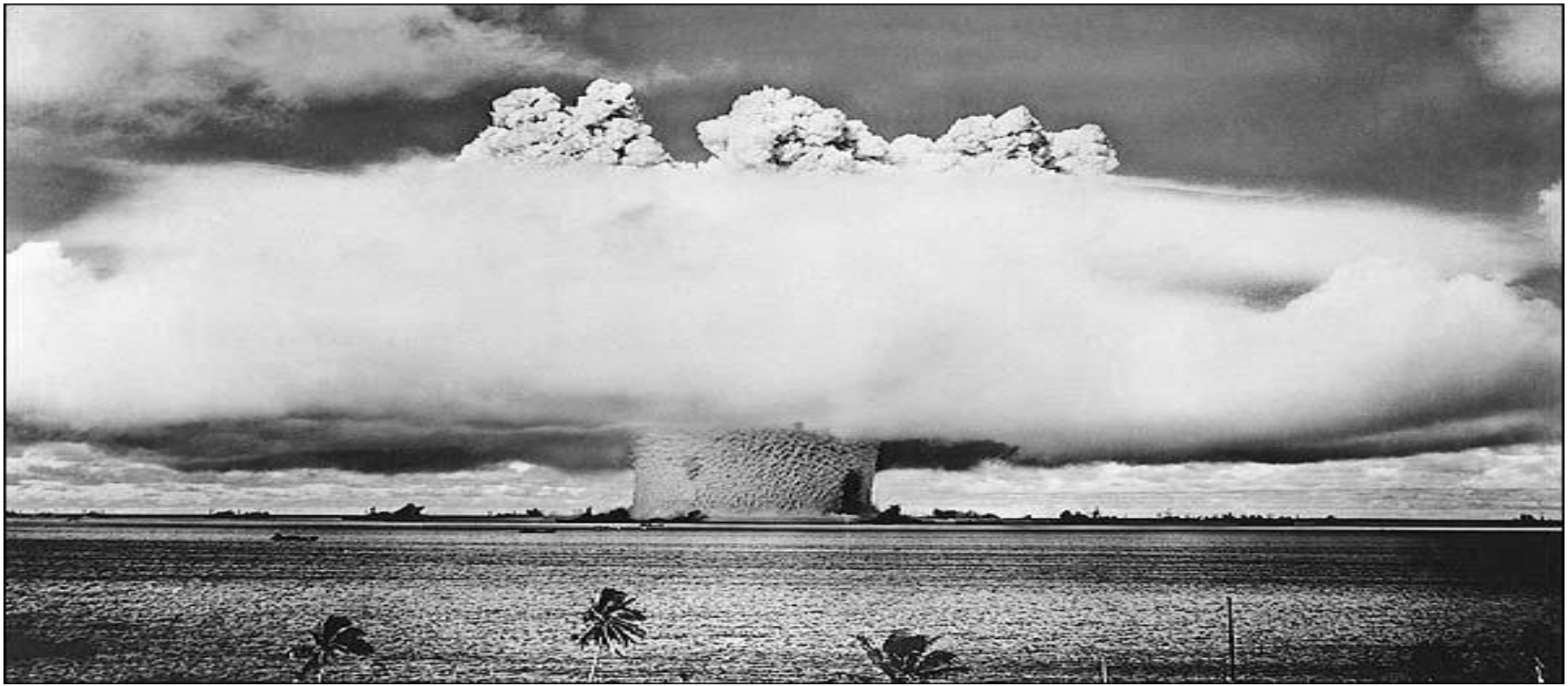
Masturbating Monkeys

- Inspired by Netflix and its Chaos Monkey
  - failure of a server should be ok due to infrastructure coping
  - ok, let's shoot some servers down at random since it shouldn't be an issue

- OpenSMTPD introduced the Masturbating Monkeys
  - failure of most system calls should be ok and result in a 421 Temporary Failure
  - ok, let's have some of them fail at random since this shouldn't be an issue

- A special branch where we introduced random failures at strategic places
  - queue read/writes, all kinds of lookups, memory allocations, …
  - random latency here and there to trigger timeouts
  - Then flood in a loop of random mail submissions

Masturbating Monkeys



Masturbating Monkeys

- A lot of error code paths eventually led to fatal()
  - Proper way to handle an issue in some startup cases, obviously not at runtime
- Initiated an audit of all fatal() calls to determine if too harsh
  - Is this caused by internal inconsistency OR by temporary lack of resources ?
- When too harsh, fix and give the **masturbating monkey** a run
- A fatal() call is eventually hit after a few seconds or minutes
  - Investigate which fatal() was hit and make the failure graceful
- Done when monkey can run for hours with a continuous flow of mails

Masturbating Monkeys

```
71  int
72  table_lookup(struct table *table, const char *key, enum table_service kind,
73      void **retp)
74  {
75          /* XXX - simulate temporary error */
76          MONKEY_RETURN(-1);
77
78          return table->t_backend->lookup(table->t_handle, key, kind, retp);
79  }
```

```
1268  1267          if (event & EV_WRITE) {
      1268  +               MONKEY_PAUSE(arc4random() % 3);
1269  1269                  if (msgbuf_write(&iev->ibuf.w) == -1)
1270  1270                          err(1, "%s: msgbuf_write", proc_to_str(smtpd_process));
1271  1271          }
1272  1272
```

Masturbating Monkeys

- fix a bug spotted by our monkeys

Browse files

portable opensmtpd-6.0.2 ... master-last-working

poolpOrg committed on 29 Nov 2012 1 parent 4beaac5 commit 574e4208071f41b5b15c163ca3622a7bdd75d14e

Showing **1 changed file** with **1 addition** and **0 deletions.** Unified | Split

1 smtpd/ruleset.c View

```
@@ -109,6 +109,7 @@ ruleset_check_source(struct table *table, const struct sockaddr_storage *ss)
109 109         case -1:
110 110                 log_warnx("warn: failure to perform a table lookup on table %s",
111 111                     table->t_name);
    112 +               return -1;
112 113         default:
113 114                 break;
114 115         }
```

Twitter shitstorms

- We use twitter as a community fuzzing tool and to test scalability

Twitter shitstorms

- The twitter shitstorms let us put pressure on an OpenSMTPD instance
  - Thousands of concurrent sessions continuously flooding sessions.

- People run their shitstorm on all kinds of MTA
  - Sendmail / Postfix / Qmail / Exim / Exchange / java corporate weird stuff
  - Self-written scripts using send-mail interface or language specific implementations
  - TLS / IPv4 / IPv6 / fast connections / laggy connections / small & large mails / …

- Pure chaos. Good. Usually triggers regressions quite fast.

- We summon shitstorms when we have scary changes in SMTP or MTA.

Our plans for the future

Our plans for the future

- Too many to be listed.
- Let's just go through a handful of the highest priority ones.
- World domination.

Our current _default_ configuration file

```
#	$OpenBSD: smtpd.conf,v 1.9 2016/05/03 18:43:45 jung Exp $

# This is the smtpd server system-wide configuration file.
# See smtpd.conf(5) for more information.

table aliases file:/etc/mail/aliases

# To accept external mail, replace with: listen on all
#
listen on lo0

# Uncomment the following to accept external mail for domain "example.org"
#
# accept from any for domain "example.org" alias <aliases> deliver to mbox
accept for local alias <aliases> deliver to mbox
accept from local for any relay
```

Our current configuration file

- The current configuration file format is about ten years old more or less
- Very simple, most people come to OpenSMTPD because of it
- Reads almost as plain english, understanding is straightforward
- TLS, Auth, DKIM, IPv4/IPv6, virtual domains and aliases is about 15 lines
- You can actually write it from scratch without editing an example config

Our current configuration file

```
pki mx1.poolp.org certificate "/etc/ssl/acme/poolp.org/fullchain.pem"
pki mx1.poolp.org key "/etc/ssl/acme/private/poolp.org/privkey.pem"
pki mail.poolp.org certificate "/etc/ssl/acme/poolp.org/fullchain.pem"
pki mail.poolp.org key "/etc/ssl/acme/private/poolp.org/privkey.pem"

listen on lo0
listen on lo0 port 10028 tag DKIM
listen on egress tls pki mx1.poolp.org hostnames { 212.83.181.7 = mail.poolp.org, 212.83.181.8 = mx1.poolp.org }
listen on egress port submission tls pki mail.poolp.org auth hostname mail.poolp.org

table sources   { 212.83.181.8 }                      # outgoing IP addresses
table helonames { 212.83.181.8 = mx1.poolp.org }      # outgoing HELO name per-IP address
table aliases   file:/etc/mail/aliases                # local aliases for my primary domains
table pdomains  file:/etc/mail/pdomains               # my primary domains (poolp.org, opensmtpd.org, ...)
table bdomains  file:/etc/mail/bdomains               # domains I'm backup MX for
table shithole  file:/etc/mail/shithole               # list of annoying senders

reject from any sender <shithole> for any                                       # reject annoying senders right away
accept for local alias <aliases> deliver to maildir
accept from any for domain <pdomains> alias <aliases> deliver to maildir        # mails for my primary domains hit maildirs
accept from any for domain <bdomains> relay backup mx1.poolp.org                # relay to higher priority MX for backup domains
accept tagged DKIM for any relay source <sources> hostnames <helonames>         # mails reinjected by DKIM-proxy can leave
accept for any relay via smtp://127.0.0.1:10027                                 # relay to DKIM-proxy
```

Our current configuration file

- There is a fundamental design flaw that needs to be addressed though
  - Rule = Conditions + Action (ie: accept from any for domain poolp.org deliver to maildir)
- Looks very cool because every rule is a single line
- However this causes TONS of issues
  - Due to transactional nature of SMTP, rule is evaluated during transaction
  - Action becomes assigned to envelope at session time
  - What if we change action later on in config ? woops, the rule we matched no longer exists
- Long story short:
  - Prevents some key features from being implemented, makes code a lot more complex

Our future configuration file

- Let's accept that actions and matching don't belong together
  - We'll need two lines rather than one to express a rule (we can have shortcuts though)
  - This is just an example, words may change

```
dispatcher foobar maildir
dispatcher barbaz relay
match from any for domain poolp.org dispatch to foobar
match from local for any dispatch to barbaz
```

- This simple indirection unlocks many problems:
  - Reloading becomes possible
  - On reload/restart, envelopes already in queue can catch up new action
  - LOTS of code gets simplified (removed 900 lines of code and counting…)

Transfer layer change

- OpenSMTPD has been used in very high-volume environments
- The transfer layer is quite good at its job
- However, it is the result of small evolutions without a full picture
- Writing a mail router and operating it for an ESP taught us a lot
- Our abstractions were wrong, they work but complexify the code
  - Code almost eric@-only readable, the code is clean but the logic is very complex
  - I don't think we can be blamed, it made sense while it evolved in micro steps.

Lookup layer change

- BTW, DANE.
- Been in our TODO for long.
- I had a working PoC two years ago that was not committable.
- It needs to be updated and made production-ready.
- Same goes for many nice features related to the lookup process.

Crypto layer change

- reyk@ made RSA privsep when OpenSSL offered the heartbleed feature
- We have an increasing demand for ECDSA support from users
- RSA privsep is implemented as an OpenSSL engine
- We need to do the same for ECDSA, tricky code…
  - … doable with a bit of time
  - … the proper mindset
  - … enough beers to cheer you up

```
316 static RSA_METHOD rsae_method = {
317         "RSA privsep engine",
318         rsae_pub_enc,
319         rsae_pub_dec,
320         rsae_priv_enc,
321         rsae_priv_dec,
322         rsae_mod_exp,
323         rsae_bn_mod_exp,
324         rsae_init,
325         rsae_finish,
326         0,
327         NULL,
328         NULL,
329         NULL,
330         rsae_keygen
331 };
```

Crypto layer change

- "I have a dream"© to kill direct OpenSSL support in OpenSMTPD
  - LibreSSL comes with a libtls interface which is simpler and less error-prone
  - We could depend on libtls interface and not the OpenSSL interface

```
264         SSL_CTX_set_session_cache_mode(ctx, SSL_SESS_CACHE_OFF);
265         SSL_CTX_set_timeout(ctx, SSL_SESSION_TIMEOUT);
266         SSL_CTX_set_options(ctx,
267             SSL_OP_ALL | SSL_OP_NO_SSLv2 | SSL_OP_NO_SSLv3 | SSL_OP_NO_TICKET);
268         SSL_CTX_set_options(ctx,
269             SSL_OP_NO_SESSION_RESUMPTION_ON_RENEGOTIATION);
270         SSL_CTX_set_options(ctx, SSL_OP_NO_CLIENT_RENEGOTIATION);
271         SSL_CTX_set_options(ctx, SSL_OP_CIPHER_SERVER_PREFERENCE);
```

```
if (tls_configure(ctx, NULL) == -1)
        err(1, "failed to configure: %s", tls_error(ctx));
```

- Does that mean OpenSMTPD won't run with OpenSSL anymore ?
  - Nope, but someone (please not me) gets to write a libtls wrapper on top of OpenSSL

- This is not doable today without hurting our community
  - Should we make this a two years plan ? that's an open question...

Crypto layer change

crash with OpenSSL 1.0.2f portable regression

#650 by gahr was closed on 2 Feb 2016

schuemann commented on 1 Feb 2016

The effect has been reported by other FreeBSD users:
http://comments.gmane.org/gmane.os.freebsd.devel.ports/128616

And it is not FreeBSD specific:
https://bugs.debian.org/cgi-bin/bugreport.cgi?bug=813398
https://bugs.archlinux.org/task/47967

Downgrading OpenSSL to 1.0.2e is a workaround.

ericfaurot commented on 1 Feb 2016

This is due to an API change in a "patchlevel" release of OpenSSL. The API we relied on is now broken.
We are working on a solution.

```
354  smtp_sni_callback(SSL *ssl, int *ad, void *arg)
355  {
356  #if defined(HAVE_TLSEXT_SERVERNAME)
357          const char              *sn;
358          void                    *ssl_ctx;
```

```
678  #ifdef HAVE_GCM_CRYPTO
679          if (env->sc_queue_key) {
680                  if (!crypto_setup(env->sc_queue_key, strlen(env->sc_queue_key)))
681                          fatalx("crypto_setup: invalid key for queue encryption");
682                  log_info("queue: queue encryption enabled");
683          }
684  #endif
```

Filters

- Filter API has been WIP for very very very long.

- People want them. Badly.
  - "You should do milter, why are you taking so much time just to go all NIH ?"
  - "I enabled this code, which you disabled and told us not to use, how do I do XXX ?"
  - "I'm facing a bug, please help. Yes, I'm running with a filter, forgot to tell you."

- The problem is not the interface, the problem is the plumbing.
  - OK let's go milter. HOW do we plug them. The interface doesn't magically plug itself.

- We have a very precise idea of what we want for our implementation.

Filters change

- We want filters to:
  - Run in different memory spaces from smtpd and one from another
  - Be able to run with different privileges, chroot()-ed in different directories
  - Be able to be written in Perl, python, lua or even shell: filters are admin tools
  - Not require our cooperation: built independently with whatever dependency
  - Be able to interact with any user input and server output in an SMTP session

- What we have:
  - All of the above, except for the last item which is more or less 90% functional
  - Sadly, the 10% is unfixable with the current model…
  - There are also minor shortcomings which prove the model isn't right

Filters change

- The filter engine is entangled in the SMTP state machine :sadface:
  - Most reliability issues in last releases were caused by subtle state fuc.. issues
- Making improvements in filters is risky
  - Regressions may be introduced in smtpd, including for setups without filters
- Each change requires considerable testing in code paths uninvolved
  - Did we break TLS because we added a filter indirection for RCPT ?
  - Are we all available to troubleshoot a regression ?
- This causes development in that area to be wayyyy too slow :-(
  - You don't want to do it too close to release locks, holidays, daytime job milestones, ...

Filters change

- Our plan is to go from THIS:

Filters change

- to THIS:

Filters change

- Welcome smtpfd, the smtp filtering daemon
- Basically kind of an SMTP proxy
- OpenSMTPD no longer knows about filters, only deals w/ regular sessions
- smtpfd no longer needs access to sessions states, can't mess w/ them
- Reuses 90% of the existing filter code AND makes OpenSMTPD simpler

Filters change

- smtpfd receives raw lines from OpenSMTPD using small & simple protocol
- it establishes a regular SMTP session back to OpenSMTPD
- it basically does a MITM between OpenSMTPD and SMTP client:
  - possibly strip / alter / inject commands and responses
- once smtpf protocol implemented in OpenSMTPD, piece of cake
- all filtering development done _outside_ the OpenSMTPD codebase

Filters change

- FYI, we already have a working smtpfd, this is not just an idea ;-)
- We'll publish it soon with a call for testing
- We're aiming a first release for OpenBSD 6.3, May 2018
  - Hopefully, eric@ has a nice talk about smtpfd next year
- smtpfd may be used with another MTA implementing the protocol
- smtpfd does not need to run on the same machine as the MTA

Other changes

- Finally, we also have small related side projects
  - deraadt@ wants us to have an spf_fetcher in base
  - I'd like a tool to autogen certs so TLS can work out of the box

- We have some ideas for nice experimental features
  - ESMTP extensions (EHLO) lets us play with new features while not breaking SMTP

How to help us ?

- Test our code, spot bugs and report them.
- Contribute, write new features, help close problem reports.
- **Donate to OpenBSD Foundation so we can have nice hackathons**
  - I want another hackathon in Nantes :-)
- Sponsor development of features (rent a dev, basically)
- Help find sponsors for development of features

On a note completely unrelated to OpenSMTPD...

Did I tell you that Vente-Privée is hiring ?

~~Call~~ Mail me maybe ?


That's all Folks!

Questions ?

IRC: #OpenSMTPD @ Freenode

Twitter: @OpenSMTPD

Mailing-lists: misc@openbsd.org / misc@opensmtpd.org

shout outs to the **dream team** :-*