KRISTOF PROVOST

AUTOMATED FIREWALL TESTING

WHO AM I?

- Kristof Provost
- kp@FreeBSD.org
- pf (in FreeBSD) maintainer
- Embedded Linux projects
- EuroBSDCon foundation board member
- Not for sale
  - For rent
  - reasonable rates

PF

- Packet Filter
- Imported from OpenBSD
  - Yes, a while ago
- Shiny things in FreeBSD that are not in OpenBSD
  - vnet
  - multi-core capable
    - Faster!
      - (er)!

WHY AUTOMATED TESTING?

- Make sure things actually work
- Convenient test case
- Prevent regressions
- Quick sanity check when making changes

REGRESSIONS

- IPv6 fragment handling
  - IPv6 fast path code broke it
  - Took ~9 months to discover and fix
- IPv6 fragments, again
  - Fix for https://nvd.nist.gov/vuln/detail/CVE-2018-6923 broke things
  - Tests found it immediately
  - two weeks between introduction and fix
  - Heisenbug. Went away during DTracing

```
int
frag6_input(struct mbuf **mp, int *offp, int proto)
{
    /* ... (9 lines) */
    uint32_t hash, hashkey[sizeof(struct in6_addr) * 2 + 1],
*hashkeyp;

    /* ... (78 lines) */

    hashkeyp = hashkey;
    memcpy(hashkeyp, &ip6->ip6_src, sizeof(struct in6_addr));
    hashkeyp += sizeof(struct in6_addr) / sizeof(*hashkeyp);
    memcpy(hashkeyp, &ip6->ip6_dst, sizeof(struct in6_addr));
    hashkeyp += sizeof(struct in6_addr) / sizeof(*hashkeyp);
    *hashkeyp = ip6f->ip6f_ident;
    hash = jenkins_hash32(hashkey, nitems(hashkey), V_ip6q_hashseed);
    hash &= IP6REASS_HMASK;
    head = IP6Q_HEAD(hash);
    IP6Q_LOCK(hash);

    /* ... */
}
```

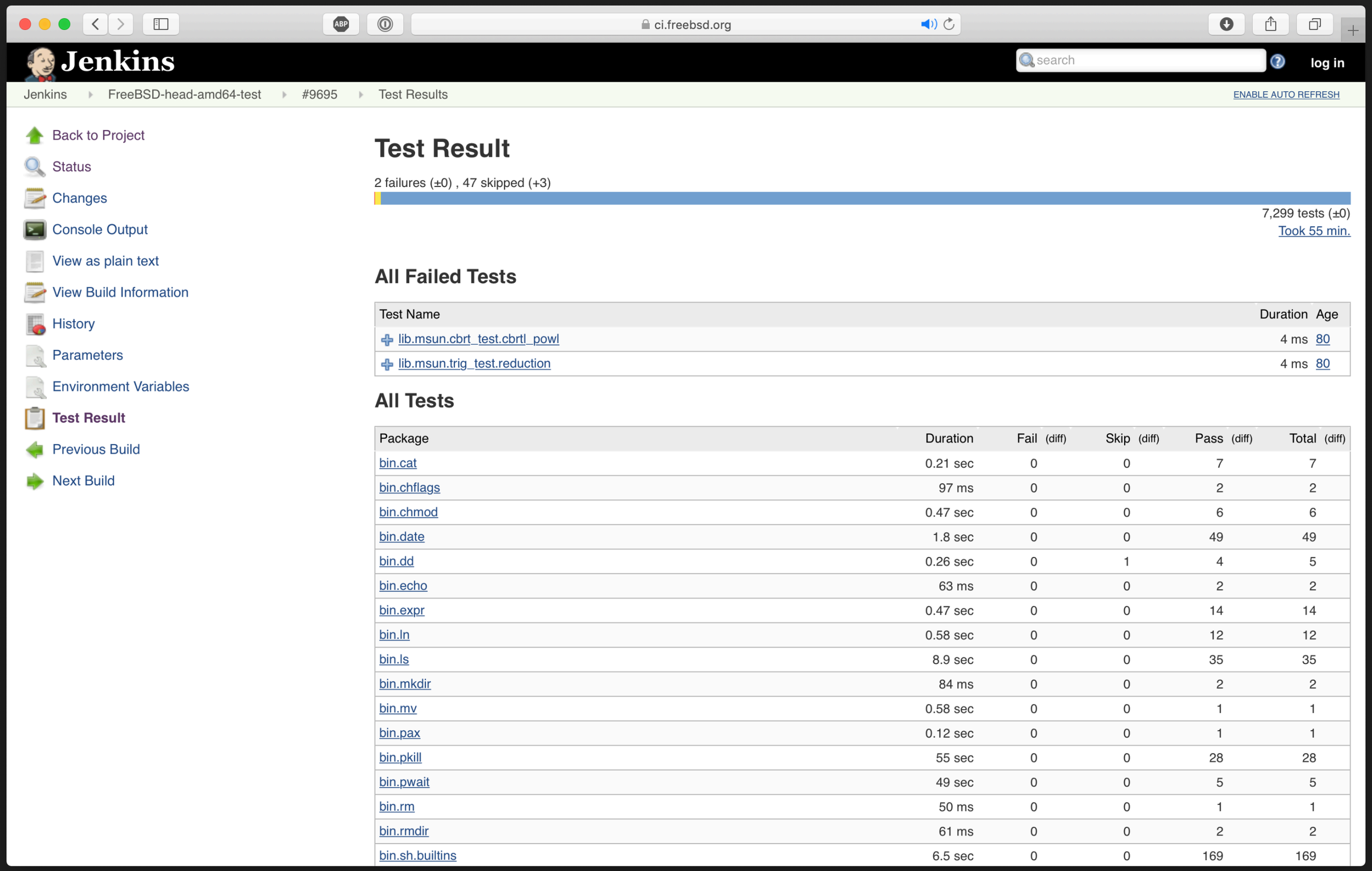
I FIXED A THING!

```
diff --git a/sys/netinet6/frag6.c b/sys/netinet6/frag6.c
index 0f30801540a..bbdbf448f7c 100644
--- a/sys/netinet6/frag6.c
+++ b/sys/netinet6/frag6.c
@@ -218,7 +218,9 @@ frag6_input(struct mbuf **mp, int *offp, int proto)
     int offset = *offp, nxt, i, next;
     int first_frag = 0;
     int fragoff, frgpartlen;    /* must be larger than u_int16_t */
-    uint32_t hash, hashkey[sizeof(struct in6_addr) * 2 + 1], *hashkeyp;
+    uint32_t hashkey[(sizeof(struct in6_addr) * 2 +
+            sizeof(ip6f->ip6f_ident)) / sizeof(uint32_t)];
+    uint32_t hash, *hashkeyp;
     struct ifnet *dstifp;
     u_int8_t ecn, ecn0;
 #ifdef RSS
```

OBJECTIVES

- Easy to write
- Easy for everyone to run
- Fast to run
- Integrate with ATF / ci.freebsd.org

CI.FREEBSD.ORG



TAKE ONE: HARDWARE

- Send packets from A to B, check replies
  - Server / switch / server
- But what if we want to forward?
  - Server / switch / server / switch / server
- What if we want to test pfsync or carp?
  - server / switch / server + server / switch / server

ISSUES WITH TAKE ONE

- What if we block all traffic?
  - Serial lines?
- What pf or FreeBSD version on all systems?
  - Netboot?
- Panics?
- What about even more complex setups?
- Where does all this hardware live?
- How do other people write tests?
  - Standardised hardware?

TAKE TWO: VIRTUAL HARDWARE

- bhyve!
- Approach taken in GSoC 2017

ISSUES WITH TAKE TWO

- What if we block all traffic?
  - Emulated serial port
  - Nested bhyve … (ci.freebsd.org)
  - Really annoying to build VM during test run
  - Panics? Possible, but still annoying
  - Slow to run

TAKE THREE: VNET

- Virtual network stack
  - Associated with jail
  - Enabled by default in 12.0
  - pf supports this (as of 12.0)

OKAY, SO HOW DO I START A JAIL WITH ITS OWN STACK? I BET IT'S HARD. IT'S HARD ISN'T IT?

- sudo jail -c name=alcatraz vnet persist

WHAT? NO NETWORK? I BET THAT'S HARD!

- sudo ifconfig epair create
  - epair0a / epair0b
- sudo ifconfig epair0a 192.0.2.1/24 up
- sudo jail -c name=alcatraz vnet persist vnet.interface=epair0b
- sudo jexec ifconfig epair0b 192.0.2.2/24 up
- ping -c 1 192.0.2.2

SAMPLE TEST: PASS_BLOCK (1/3)

```
# $FreeBSD$

. $(atf_get_srcdir)/utils.subr

atf_test_case "v4" "cleanup"
v4_head()
{
    atf_set descr 'Basic pass/block test for IPv4'
    atf_set require.user root
}
```

SAMPLE TEST: PASS_BLOCK (2/3)

```
v4_body()
{
    pft_init

    epair=$(pft_mkepair)
    ifconfig ${epair}a 192.0.2.1/24 up

    # Set up a simple jail with one interface
    pft_mkjail alcatraz ${epair}b
    jexec alcatraz ifconfig ${epair}b 192.0.2.2/24 up

    # Trivial ping to the jail, without pf
    atf_check -s exit:0 -o ignore ping -c 1 -t 1 192.0.2.2

    # pf without policy will let us ping
    jexec alcatraz pfctl -e
    atf_check -s exit:0 -o ignore ping -c 1 -t 1 192.0.2.2

    # Block everything
    pft_set_rules alcatraz "block in"
    atf_check -s exit:2 -o ignore ping -c 1 -t 1 192.0.2.2
}
```

SAMPLE TEST: PASS_BLOCK (3/3)

```
v4_cleanup()
{
    pft_cleanup
}
```

```
atf_init_test_cases()
{
    atf_add_test_case "v4"
}
```

SAMPLE OUTPUT

```
% sudo kyua test pass_block:v4

pass_block:v4  ->  passed  [1.200s]


Results file id is usr_tests_sys_netpfil_pf.20190106-081724-193657

Results saved to /root/.kyua/store/results.usr_tests_sys_netpfil_pf.
20190106-081724-193657.db


1/1 passed (0 failed)
```

PFSYNC: FOR ADVANCED USERS (1/2)

```
basic_body()
{
    pfsynct_init

    epair_sync=$(pft_mkepair)
    epair_one=$(pft_mkepair)
    epair_two=$(pft_mkepair)

    pft_mkjail one ${epair_one}a ${epair_sync}a
    pft_mkjail two ${epair_two}a ${epair_sync}b

    # pfsync interface
    jexec one ifconfig ${epair_sync}a 192.0.2.1/24 up
    jexec one ifconfig ${epair_one}a 198.51.100.1/24 up
    jexec one ifconfig pfsync0 \
        syncdev ${epair_sync}a \
        maxupd 1 \
        up
    jexec two ifconfig ${epair_two}a 198.51.100.2/24 up
    jexec two ifconfig ${epair_sync}b 192.0.2.2/24 up
    jexec two ifconfig pfsync0 \
        syncdev ${epair_sync}b \
        maxupd 1 \
        up
```

PFSYNC: FOR ADVANCED USERS (2/2)

```
    # Enable pf!
    jexec one pfctl -e
    pft_set_rules one \
        "set skip on ${epair_sync}a" \
        "pass keep state"
    jexec two pfctl -e
    pft_set_rules two \
        "set skip on ${epair_sync}b" \
        "pass keep state"

    ifconfig ${epair_one}b 198.51.100.254/24 up

    ping -c 1 -S 198.51.100.254 198.51.100.1

    # Give pfsync time to do its thing
    sleep 2

    if ! jexec two pfctl -s states | grep icmp | grep 198.51.100.1 | \
        grep 198.51.100.2 ; then
        atf_fail "state not found on synced host"
    fi
}
```

WHERE TO FIND THE TESTS

- Source
  - /usr/src/tests/sys/netpfil/pf
- Installed
  - /usr/tests/sys/netpfil/pf

HOW DO I RUN TESTS?

- pkg install kyua scapy
- kldload pfsync
- cd /usr/tests/sys/netpfil
- kyua test

MORE INFORMATION

- FreeBSD journal
  - March/April 2019
  - https://www.freebsdfoundation.org/journal/

SERIOUSLY, WRITE TESTS. TESTS ARE GOOD.

Me. Just now.

WHAT'S IN IT FOR YOU?

- Prototype setups
- Prevent your use case from breaking
- Make it easy for me to fix your bug
  - Seriously. I'm lazy. Make it easy
  - Often reproducing is more than half of the actual work
    - Assuming I even understand your setup
  - With a good test it's often easier to fix than to review a patch
    - I'd have to write the test anyway. Do it for me
- Money also motivates me

OTHER VNET TESTS

- netipsec
  - Olivier was tired of IPSec being broken
  - Now
    - there are tests
    - IPSec isn't broken
    - If someone does break it, Li-Wen will shout[*] at them

[*] WELL... POLITELY ASK THEM TO FIX IT

QUESTIONS?

ONE MORE?

NAT: OBSCURE NAT PROBLEM (1/2)

```
exhaust_body()
{
    pft_init
    epair_nat=$(vnet_mkepair)
    epair_echo=$(vnet_mkepair)

    vnet_mkjail nat ${epair_nat}b ${epair_echo}a
    vnet_mkjail echo ${epair_echo}b

    ifconfig ${epair_nat}a 192.0.2.2/24 up
    route add -net 198.51.100.0/24 192.0.2.1

    jexec nat ifconfig ${epair_nat}b 192.0.2.1/24 up
    jexec nat ifconfig ${epair_echo}a 198.51.100.1/24 up
    jexec nat sysctl net.inet.ip.forwarding=1

    jexec echo ifconfig ${epair_echo}b 198.51.100.2/24 up
    jexec echo /usr/sbin/inetd $(atf_get_srcdir)/echo_inetd.conf

    # Enable pf!
    jexec nat pfctl -e
    pft_set_rules nat \
        "nat pass on ${epair_echo}a inet from 192.0.2.0/24 to any
-> (${epair_echo}a) port 30000:30001 sticky-address"
```

NAT: OBSCURE NAT PROBLEM (2/2)

```
    # Sanity check
    atf_check -s exit:0 -o ignore ping -c 3 198.51.100.2

    echo "foo" | nc -N 198.51.100.2 7
    echo "foo" | nc -N 198.51.100.2 7

    # This one will fail, but that's expected
    echo "foo" | nc -N 198.51.100.2 7 &

    sleep 1

    # If the kernel is stuck in pf_get_sport() this will not
succeed either.
    timeout 2 jexec nat pfctl -sa
    if [ $? -eq 124 ]; then
        # Timed out
        atf_fail "pfctl timeout"
    fi
}
```