Fuzzing iOS Kernel Networking in Usermode

Ned Williamson @ POC 2020

Whoami

- Employed at Google Project Zero
- Generally interested in creative fuzzing of difficult or frustrating targets
- Designed and implemented this fuzzer as a 20% project

Agenda

- Project Overview
- Attack Surface Research and Planning
- Building with CMake
- Wrapping Syscalls
- Writing a Fuzz Target
- Faking Functionality
- Improving Coverage
- Reproducing Known Crashes

SockFuzzer

- Novel iOS kernel fuzzer based on XNU sources
- Fuzzes subset of kernel functionality in userland
- Goal is to fuzz the whole stack
  - Assume local attacker app + a cooperating remote server
- Anecdotally, attack surfaces where state is influenced on “two ends” is a good target for bug hunting
- Local only, remote only, combination in scope
- Inspired by Chrome IPCs + cooperating remote server

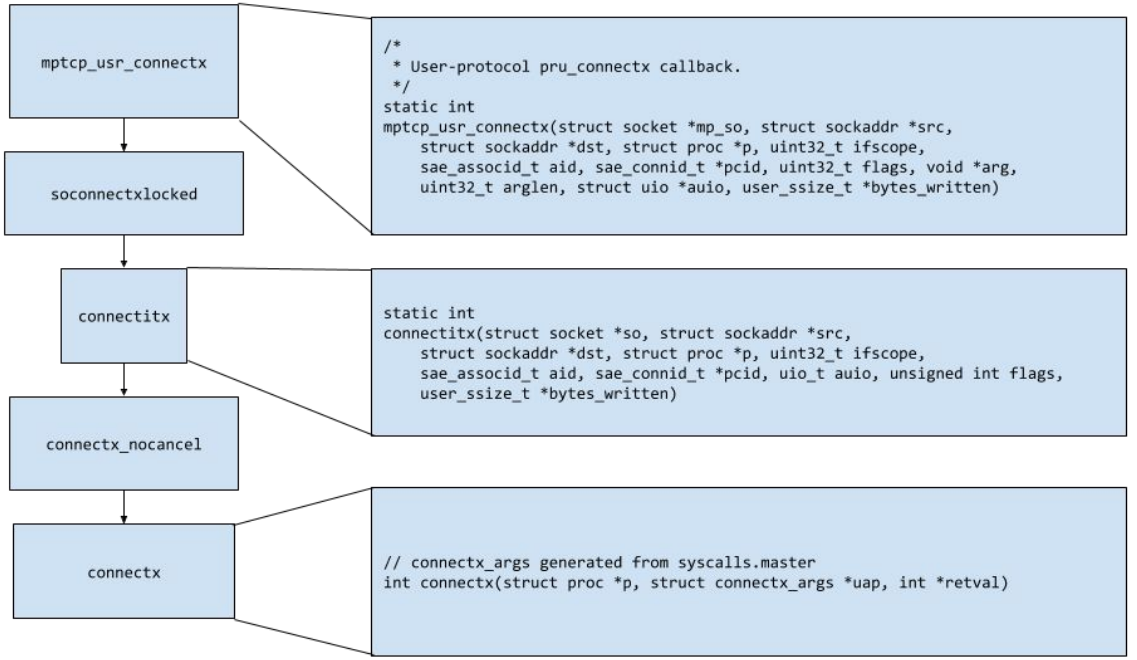
Background Research

- Why network stack?
  - Lots of byte level manipulation.
  - Lots of state.
  - Greatly exposed to app sandbox and remote attackers.
- Inspiration
  - CVE-2018-4241 (multipath): a crafted connectx syscall
  - CVE-2018-4407 (ICMP OOB write): a crafted remote packet

A Hybrid Fuzzing Approach

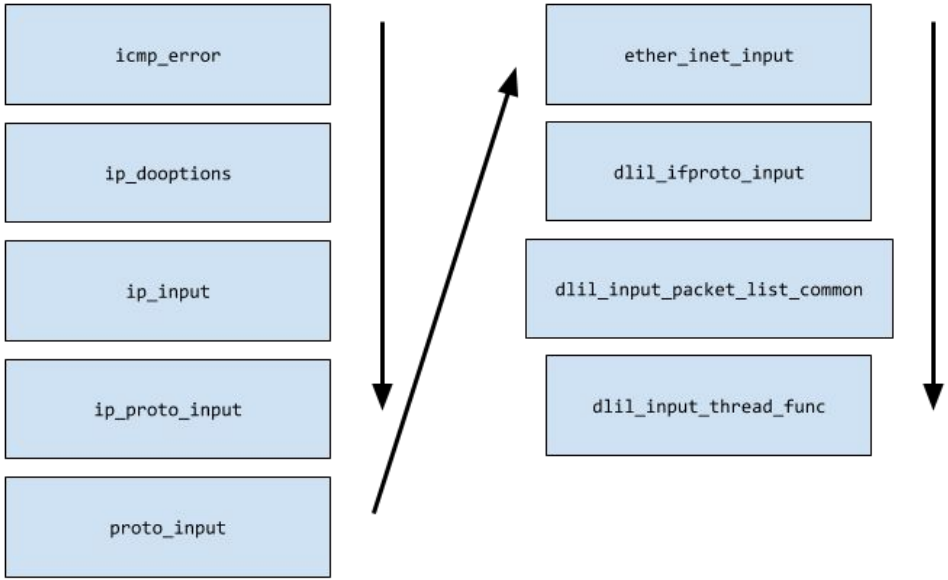
- Cover everything from the top down in one fuzz target
- Initial drafts of fuzzer called intermediate functions
  - Quickly settled on interleaving {network syscalls, ip_input, ip6_input}
- Userland
  - ASAN
  - LibFuzzer (and other fuzzing engines)
  - Faster iteration with ninja-based build and deploy
  - Fast coverage visualization with clang-coverage

Attack Surface Review: CVE-2018-4241 (Multipath)



Going down:
More code covered, more setup work, more representative of attack surface.

Attack Surface Review: CVE-2018-4407 (ICMP)

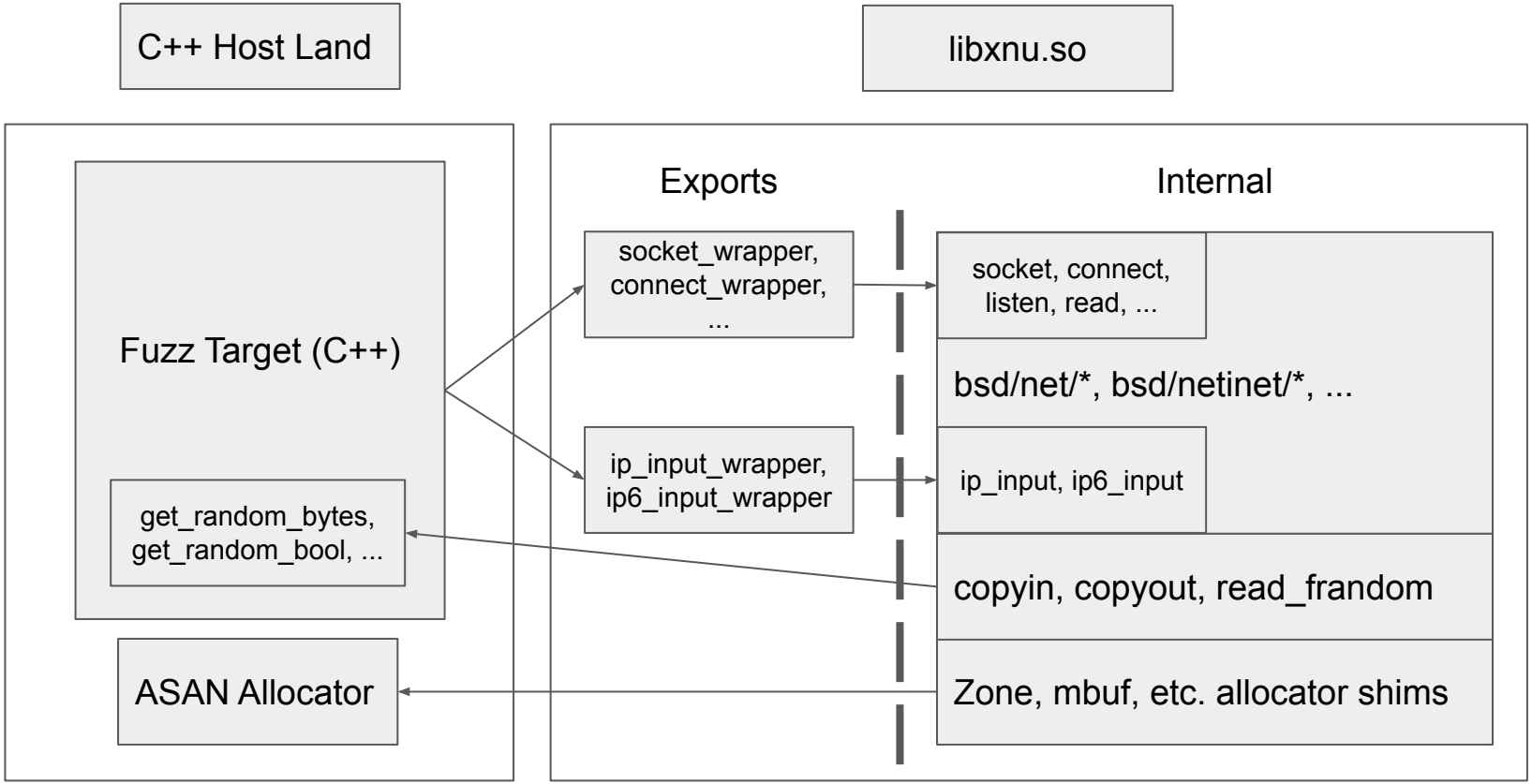


Building the Kernel

- Most kernel code is actually quite portable
  - It doesn’t have external dependencies by design
- Kernel is a black box that has syscalls (and other interfaces) and talks to hardware
- Lots of kernel code is not hardware-specific but lives there for historical or performance reasons
- Let’s build a subset that includes the functionality we want to test
- Fake and stub the rest
- Microkernels exist because most functionality doesn’t really need to live in kernel anyways. We have userland network stacks for this reason. (e.g. usrsctp)

Building the Kernel: Using CMake

- I replaced the original build system with CMake
  - Captured compiler invocations from a normal kmk build
- Benefits
  - Build multiple targets (ASAN+libfuzzer, clang coverage) with ninja!
  - Surprisingly easy to do; most files in a given subsystem are built with the same flags
  - Provides lots of insight and a high degree of control over the build
    - Psychological: it’s discouraging to work with a large project-specific build system, but if you “own” a small build system it’s easy to play with
  - Porting a working build to Linux is easy
- Drawbacks
  - Manual translation effort involved
  - Need to account for generated headers and files


C++ Host Land
libxnu.so
Fuzz Target (C++)
get_random_bytes,
get_random_bool, ...
ASAN Allocator
Exports
Internal
socket_wrapper,
connect_wrapper,
...
ip_input_wrapper,
ip6_input_wrapper
socket, connect,
listen, read, ...
bsd/net/*, bsd/netinet/*, ...
ip_input, ip6_input
copyin, copyout, read_frandom
Zone, mbuf, etc. allocator shims

Building libxnu.so

```
set(XNU_DEFINES
    -DAPPLE
    -DKERNEL
    # ...
)

set(XNU_SOURCES
    bsd/conf/param.c
    bsd/kern/kern_asl.c
    # ...
    fuzz/syscall_wrappers.c
    fuzz/ioctl.c
    fuzz/backend.c
)

add_library(xnu SHARED ${XNU_SOURCES} ${FUZZER_FILES} ${XNU_HEADERS})
add_executable(net_fuzzer fuzz/net_fuzzer.cc ${NET_PROTO_SRCS} ...)
target_include_directories(net_fuzzer PRIVATE libprotobuf-mutator)
```

Adding Stubs

```
$ ninja
...
  "_zdestroy", referenced from:
      _if_clone_detach in libxnu.a(if.c.o)
  "_zfree", referenced from:
      _kqueue_destroy in libxnu.a(kern_event.c.o)
      _knote_free in libxnu.a(kern_event.c.o)
      _kqworkloop_get_or_create in libxnu.a(kern_event.c.o)
      _kev_delete in libxnu.a(kern_event.c.o)
      _pipepair_alloc in libxnu.a(sys_pipe.c.o)
      _pipepair_destroy_pipe in libxnu.a(sys_pipe.c.o)
      _so_cache_timer in libxnu.a(uipc_socket.c.o)
      ...
  "_zinit", referenced from:
      _knote_init in libxnu.a(kern_event.c.o)
      _kern_event_init in libxnu.a(kern_event.c.o)
      ...
ld: symbol(s) not found for architecture x86_64
clang: error: linker command failed with exit code 1 (use -v to see invocation)
ninja: build stopped: subcommand failed.
```

Building libxnu.so

```
set(XNU_DEFINES
    -DAPPLE
    -DKERNEL
    # ...
)

set(XNU_SOURCES
    bsd/conf/param.c
    bsd/kern/kern_asl.c
    # ...
    fuzz/syscall_wrappers.c
    fuzz/ioctl.c
    fuzz/backend.c
    fuzz/stubs.c
    fuzz/fake_impls.c
)

add_library(xnu SHARED ${XNU_SOURCES} ${FUZZER_FILES} ${XNU_HEADERS})
add_executable(net_fuzzer fuzz/net_fuzzer.cc ${NET_PROTO_SRCS} ...)
target_include_directories(net_fuzzer PRIVATE libprotobuf-mutator)
```

Adding Stubs

```
// Unimplemented stub functions
// These should be replaced with real or mock impls.

#include <kern/assert.h>
#include <stdbool.h>

int printf(const char* format, ...);

void Assert(const char* file, int line, const char* expression) {
  printf("%s: assert failed on line %d: %s\n", file, line, expression);
  __builtin_trap();
}

void IOBSDGetPlatformUUID() { assert(false); }
```

Wrapping Syscalls

```
# CMakeLists.txt
set_target_properties(xnu PROPERTIES C_VISIBILITY_PRESET hidden)

# fuzz/syscall_wrappers.c (generated from syscalls.master)
__attribute__((visibility("default"))) int accept_wrapper(int s, caddr_t name, socklen_t* anamelen, int* retval) {
  struct accept_args uap = {
      .s = s,
      .name = name,
      .anamelen = anamelen,
  };
  return accept(kernproc, &uap, retval);
}

__attribute__((visibility("default"))) void ip_input_wrapper(void* m) {
  ip_input((mbuf_t)m);
}
```

Writing a Target: Calling Syscalls

```
# net_fuzzer.proto
message Session {
  repeated Command commands = 1;
  required bytes data_provider = 2;
}

message Command {
  oneof command {
    Packet ip_input = 1;
    SetSocketOpt set_sock_opt = 2;
    ...
}

message SetSocketOpt {
  optional Protocol level = 1;
  optional SocketOptName name = 2;
  // TODO(nedwill): structure for val
  optional bytes val = 3;
  optional FileDescriptor fd = 4;
}
```

```
// net_fuzzer.cc
DEFINE_TEXT_PROTO_FUZZER(const Session &session) {
  if (!ready) {
    initialize_network();
    ready = true;
  }

  for (const Command &command : session.commands()) {
      case Command::kSetSockOpt: {
        int s = command.set_sock_opt().fd();
        int level = command.set_sock_opt().level();
        int name = command.set_sock_opt().name();
        size_t size = command.set_sock_opt().val().size();
        std::unique_ptr<char[]> val(new char[size]);
        memcpy(val.get(), command.set_sock_opt().val().data(), size);
        setsockopt_wrapper(s, level, name, val.get(), size, nullptr);
        break;
      }
// ...
```

Writing a Target: Packet Structure

```
message Packet {
  oneof packet {
    TcpPacket tcp_packet = 1;
  }
}

message TcpPacket {
  required IpHdr ip_hdr = 1;
  required TcpHdr tcp_hdr = 2;
  optional bytes data = 3;
}

message IpHdr {
  required uint32 ip_hl = 1;
  required IpVersion ip_v = 2;
  required uint32 ip_tos = 3;
  required uint32 ip_len = 4;
  required uint32 ip_id = 5;
  required uint32 ip_off = 6;
  required uint32 ip_ttl = 7;
  required Protocol ip_p = 8;
  required InAddr ip_src = 9;
  required InAddr ip_dst = 10;
}
```

```
std::string get_ip_hdr(const IpHdr &hdr, size_t expected_size) {
  struct in_addr ip_src = {.s_addr = (unsigned int)hdr.ip_src()};
  struct in_addr ip_dst = {.s_addr = (unsigned int)hdr.ip_dst()};
  struct ip ip_hdr = {
      .ip_hl = hdr.ip_hl(),
      .ip_v = hdr.ip_v(),
      .ip_tos = (u_char)hdr.ip_tos(),
      .ip_len = (u_short)__builtin_bswap16(expected_size),
      .ip_id = (u_short)hdr.ip_id(),
      .ip_off = (u_short)hdr.ip_off(),
      .ip_ttl = (u_char)hdr.ip_ttl(),
      .ip_p = (u_char)hdr.ip_p(),
      .ip_sum = 0,
      .ip_src = ip_src,
      .ip_dst = ip_dst,
  };
  std::string dat((char *)&ip_hdr, (char *)&ip_hdr + sizeof(ip_hdr));
  return dat;
}
```

Supporting Usermode XNU

- We need some supporting functionality to port the stack to userland.
  - One time “boot-up” initialization
  - Allocators (classic, zone, mbuf)
  - Threads
  - Randomness
  - Authentication

Booting Up: Initializing Full BSD (684 lines)

```
/*
 * This function is called very early on in the Mach startup, from the
 * function start_kernel_threads() in osfmk/kern/startup.c.  It's called
 * in the context of the current (startup) task using a call to the
 * function kernel_thread_create() to jump into start_kernel_threads().
 * Internally, kernel_thread_create() calls thread_create_internal(),
 * which calls uthread_alloc().  The function of uthread_alloc() is
 * normally to allocate a uthread structure, and fill out the uu_sigmask,
 * uu_context fields.  It skips filling these out in the case of the "task"
 * being "kernel_task", because the order of operation is inverted.  To
 * account for that, we need to manually fill in at least the contents
 * of the uu_context.vc_ucred field so that the uthread structure can be
 * used like any other.
 */

void
bsd_init(void)
{
    struct uthread *ut;
    unsigned int i;
    struct vfs_context context;
// ...
```

Booting Up: Initializing Humble mini-BSD

```
void initialize_network() {
  mcache_init();
  mbinit();
  eventhandler_init();
  pipeinit();
  dlil_init();
  socketinit();
  domaininit();
  loopattach();
  ether_family_init();
  tcp_cc_init();
  net_init_run();
  necp_init();
}
```

Allocators: Classic Malloc+Free

- Easy: just send these to the host libc

```
void* __MALLOC(size_t size, int type, int flags, vm_allocation_site_t* site) {
  if (size == 0) {
    return NULL;
  }

  void* addr = malloc(size);
  if (!addr) {
    return NULL;
  }

  if (flags & M_ZERO) {
    bzero(addr, size);
  }

  return addr;
}
```

Allocators: Zone Allocator

```
struct zone {
  uintptr_t size;
};

struct zone* zinit(uintptr_t size, uintptr_t max, uintptr_t alloc,
                   const char* name) {
  struct zone* zone = (struct zone*)calloc(1, sizeof(struct zone));
  zone->size = size;
  return zone;
}

void* zalloc(struct zone* zone) {
  assert(zone != NULL);
  return calloc(1, zone->size);
}

void zfree(void* zone, void* dat) {
  (void)zone;
  free(dat);
}
```

Allocators: Mbufs

- Much more complicated, requires supporting the mbuf structure format
- Essentially a linked list of packets with inline or out of line data
- Still manageable and worth the investment to get ASAN-aware allocations

```
struct mbuf {
    struct m_hdr m_hdr;
    union {
        struct {
            struct pkthdr MH_pkthdr;        /* M_PKTHDR set */
            union {
                struct m_ext MH_ext;    /* M_EXT set */
                char    MH_databuf[_MHLEN];
            } MH_dat;
        } MH;
        char    M_databuf[_MLEN];               /* !M_PKTHDR, !M_EXT */
    } M_dat;
};
```

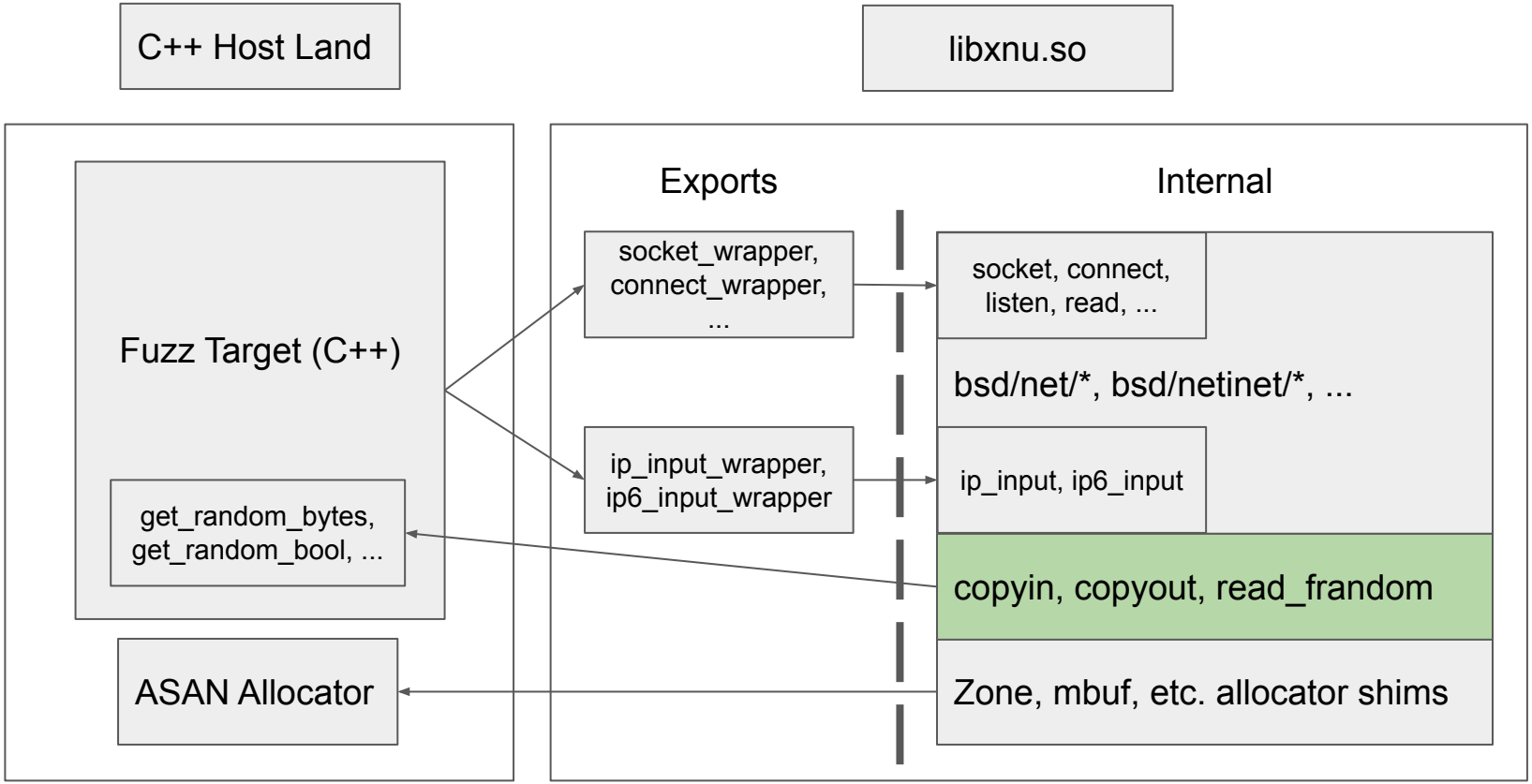
Allocators: Mbufs

- To simplify linking, added code to libxnu to allocate an opaque mbuf for me

```
struct mbuf* mbuf_create(const uint8_t* data, size_t size, bool is_header,
                         bool force_ext, int mtype, int pktflags) {
  struct mbuf* m = NULL;
  if (posix_memalign((void**)&m, MSIZE, sizeof(struct mbuf))) {
    return NULL;
  }
  m->m_type = mtype;

  if (size > sizeof(m->M_dat.MH.MH_dat.MH_databuf)) {
    // Allocate external buffer and initialize RFA
  }

  if (data) memcpy(m->m_data, data, size);
  m->m_len = size;
  // Initialize header if needed (elided)
  return m;
}
```


C++ Host Land
libxnu.so
Fuzz Target (C++)
get_random_bytes,
get_random_bool, ...
ASAN Allocator
Exports
Internal
socket_wrapper,
connect_wrapper,
...
ip_input_wrapper,
ip6_input_wrapper
socket, connect,
listen, read, ...
bsd/net/*, bsd/netinet/*, ...
ip_input, ip6_input
copyin, copyout, read_frandom
Zone, mbuf, etc. allocator shims

Accessing User Memory

```c
int copyin(void* user_addr, void* kernel_addr, size_t nbytes) {
  // Address 1 means use fuzzed bytes, otherwise use real bytes.
  // NOTE: this does not support nested useraddr.
  if (user_addr != (void*)1) {
    memcpy(kernel_addr, user_addr, nbytes);
    return 0;
  }

  if (get_fuzzed_bool()) {
    return -1;
  }

  get_fuzzed_bytes(kernel_addr, nbytes);
  return 0;
}
```

```c
int copyout(const void* kaddr, user_addr_t udaddr, size_t len) {
  // randomly fail
  if (get_fuzzed_bool()) {
    return 1;
  }

  if (!udaddr || udaddr == 1 || !real_copyout) {
    void* buf = malloc(len);
    memcpy(buf, kaddr, len);
    free(buf);
    return 0;
  }

  memcpy((void*)udaddr, kaddr, len);
  return 0;
}
```

Synchronization and Threads

- Only use 1 thread, disable all sync primitives
- Cooperative scheduling by doing scheduled work from main fuzzer loop
- This works well for finding stateful bugs and determinism
- Hides race conditions (definitely a drawback)
- Biased by work on Chrome where async work uses a “task pool” abstraction

Synchronization and Threads

```
void* thread_call_allocate_with_options() { return (void*)1; }

void* lck_grp_attr_alloc_init() { return (void*)1; }

void* lck_grp_alloc_init() { return (void*)1; }

void* lck_attr_alloc_init() { return (void*)1; }

void* lck_rw_alloc_init() { return (void*)1; }

void lck_mtx_assert() {}

void lck_mtx_init() {}

void lck_mtx_lock() {}

void lck_spin_init() {}

int kernel_thread_start() { return 0; }

// TODO: handle timer scheduling
void timeout() { assert(false); }
```

Synchronization and Threads

```
__attribute__((visibility("default"))) void clear_all() {
  inpcb_timeout(NULL, NULL);
  key_timehandler();
  frag_timeout();
  nd6_slowtimo();
  nd6_timeout();
  in6_tmpaddrtimer();
  mp_timeout();
  igmp_timeout();
  tcp_fuzzer_reset();
  frag6_timeout();
  mld_timeout();

  // this adds work to the work queue
  in6_rtqtimo(NULL);
  in_rtqtimo(NULL);

  // TODO: nd6_dad_timer
  nstat_idle_check(NULL, NULL);
  domain_timeout(NULL);
}
```

Randomness

```
unsigned long RandomULong() {
  // returning 0 here would be a failure
  return 1;
}

void read_frandom(void* buffer, unsigned int numBytes) {
  get_fuzzed_bytes(buffer, numBytes);
}

// These are callbacks to let the C-based backend access the fuzzed input
// stream.
void get_fuzzed_bytes(void *addr, size_t bytes) {
  // If we didn't initialize the fdp just clear the bytes.
  if (!fdp) {
    memset(addr, 0, bytes);
    return;
  }
  memset(addr, 0, bytes);
  std::vector<uint8_t> dat = fdp->ConsumeBytes<uint8_t>(bytes);
  memcpy(addr, dat.data(), dat.size());
}
```

Authentication

```
void* proc_ucred() {
  return (void*)1;
}

int suser(void* arg1, void* arg2) {
  (void)arg1;
  (void)arg2;
  return 0;
}


int mac_socket_check_create() { return 0; }

int mac_socket_check_accept() { return 0; }
```

Authentication

```
// KAuth check
if (!kauth_cred_issuser(kauth_cred_get())) {
    return EPERM;
}
```

```
// Our fake implementations
void* kauth_cred_get() {
  return (void*)1;
}

bool kauth_cred_issuser() { return true; }
```

First Draft Complete

- At this point we have the pieces we need:
  - Network stack ported to a userland library
  - Wiring of memory allocators to “host” ASAN allocator
  - Implementations/fakes for essential kernel facilities
  - A fuzz target

Coverage Guided Development

```
3332  57.5k        if (optlen < IPOPT_OLEN + sizeof(*cp) ||
3333  57.5k            optlen > cnt) {
3334  51.1k          code = &cp[IPOPT_OLEN] - (u_char *)ip;
3335  51.1k          goto bad;
3336  51.1k        }
3337  8.61k      }
3338  8.61k      switch (opt) {
3339  8.61k      default:
3340  8.61k        break;
3341     0
3342     0      /*
3343     0       * Source routing with record.
3344     0       * Find interface with current destination address.
3345     0       * If none on this machine then drop if strictly routed,
3346     0       * or do nothing if loosely routed.
3347     0       * Record interface address and bring up next address
3348     0       * component.  If strictly routed make sure next
3349     0       * address is on directly accessible net.
3350     0       */
3351     0      case IPOPT_LSRR:
3352     0      case IPOPT_SSRR:
3353     0        if (optlen < IPOPT_OFFSET + sizeof(*cp)) {
3354     0          code = &cp[IPOPT_OLEN] - (u_char *)ip;
3355     0          goto bad;
3356     0        }
3357     0        if ((off = cp[IPOPT_OFFSET]) < IPOPT_MINOFF) {
3358     0          code = &cp[IPOPT_OFFSET] - (u_char *)ip;
3359     0          goto bad;
3360     0        }
```

Coverage Guided Development

- Improve completeness
  - Top-down: add to grammar
  - Bottom-up: modify target code
- Improve soundness
  - Revert bottom-up changes
    - If useful changes, support them from target level using grammar changes
    - If not useful, just revert and move on
  - Fix mis-modeled faked functionality
- Goals
  - (optional) Ensure known bugs can be reproduced
  - Cover all attacker-reachable code in the subsystem
  - Only have sound changes

Bottom Up: Checksums

```
diff --git a/bsd/netinet/tcp_input.c b/bsd/netinet/tcp_input.c
index d7d04516..078c2127 100644
--- a/bsd/netinet/tcp_input.c
+++ b/bsd/netinet/tcp_input.c
@@ -7126,7 +7126,8 @@ tcp_input_checksum(int af, struct mbuf *m, struct tcphdr *th, int off, int tlen)
        if (th->th_sum != 0) {
                tcpstat.tcps_rcvbadsum++;
                IF_TCP_STATINC(ifp, badformat);
-               return -1;
+               // nedwill: always accept tcp hash
+               return 0;
        }
```

Bottom Up: Packet Delivery

```
void
tcp_input(struct mbuf *m, int off0)
{
// ...
        if (isipv6) {
            inp = in6_pcblookup_hash(&tcbinfo, &ip6->ip6_src, th->th_sport,
                &ip6->ip6_dst, th->th_dport, 1,
                m->m_pkthdr.rcvif);
        } else
#endif /* INET6 */
        inp = in_pcblookup_hash(&tcbinfo, ip->ip_src, th->th_sport,
            ip->ip_dst, th->th_dport, 1, m->m_pkthdr.rcvif);
```

Bottom Up: Packet Delivery

```
/*
 * Lookup PCB in hash list.
 */
struct inpcb *
in_pcblookup_hash(struct inpcbinfo *pcbinfo, struct in_addr faddr,
    u_int fport_arg, struct in_addr laddr, u_int lport_arg, int wildcard,
    struct ifnet *ifp)
{
// ...
    head = &pcbinfo->ipi_hashbase[INP_PCBHASH(faddr.s_addr, lport, fport,
        pcbinfo->ipi_hashmask)];
    LIST_FOREACH(inp, head, inp_hash) {
-               if (inp->inp_faddr.s_addr == faddr.s_addr &&
-                   inp->inp_laddr.s_addr == laddr.s_addr &&
-                   inp->inp_fport == fport &&
-                   inp->inp_lport == lport) {
+               if (!get_fuzzed_bool()) {
                    if (in_pcb_checkstate(inp, WNT_ACQUIRE, 0) !=
                        WNT_STOPUSING) {
                        lck_rw_done(pcbinfo->ipi_lock);
                        return inp;
```

Bottom Up: Packet Delivery

```
diff --git a/bsd/netinet/in_pcb.h b/bsd/netinet/in_pcb.h
index a5ec42ab..37f6ee50 100644
--- a/bsd/netinet/in_pcb.h
+++ b/bsd/netinet/in_pcb.h
@@ -611,10 +611,9 @@ struct inpcbinfo {
        u_int32_t               ipi_flags;
 };

-#define INP_PCBHASH(faddr, lport, fport, mask) \
-       (((faddr) ^ ((faddr) >> 16) ^ ntohs((lport) ^ (fport))) & (mask))
-#define INP_PCBPORTHASH(lport, mask) \
-       (ntohs((lport)) & (mask))
+// nedwill: let all pcbs share the same hash
+#define        INP_PCBHASH(faddr, lport, fport, mask) (0)
+#define        INP_PCBPORTHASH(lport, mask) (0)

 #define INP_IS_FLOW_CONTROLLED(_inp_) \
        ((_inp_)->inp_flags & INP_FLOW_CONTROLLED)
```

Reproducing the Old Findings: ICMP

```
==3613660==ERROR: AddressSanitizer: heap-buffer-overflow on address 0x61d0001ff474 at pc 0x0000004a1015 bp 0x7fff6a689c10 sp
0x7fff6a6893d8
WRITE of size 20 at 0x61d0001ff474 thread T0
    #0 0x4a1014 in __asan_memmove /out/llvm-project/compiler-rt/lib/asan/asan_interceptors_memintrinsics.cpp:30:3
    #1 0x7f721b4cded9 in __asan_bcopy fuzz/san.c:37:3
    #2 0x7f721b1e8dcf in icmp_error bsd/netinet/ip_icmp.c:362:2
    #3 0x7f721b1fa25b in ip_dooptions bsd/netinet/ip_input.c:3577:2
    #4 0x7f721b1f34db in ip_input bsd/netinet/ip_input.c:2230:34
    #5 0x7f721b4c7740 in ip_input_wrapper fuzz/backend.c:132:3
    #6 0x4e05b9 in DoIpInput fuzz/net_fuzzer.cc:610:7
    #7 0x4e1ca3 in TestOneProtoInput(Session const&) fuzz/net_fuzzer.cc:720:9

0x61d0001ff474 is located 12 bytes to the left of 2048-byte region [0x61d0001ff480,0x61d0001ffc80)
allocated by thread T0 here:
    #0 0x4a1822 in calloc /out/llvm-project/compiler-rt/lib/asan/asan_malloc_linux.cpp:154:3
    #1 0x7f721b4cce20 in mbuf_create fuzz/zalloc.c:157:45
    #2 0x7f721b4cd49e in mcache_alloc fuzz/zalloc.c:187:12
    #3 0x7f721ade5e84 in m_getcl bsd/kern/uipc_mbuf.c:3962:6
    #4 0x7f721b1e88fc in icmp_error bsd/netinet/ip_icmp.c:296:7
    #5 0x7f721b1fa25b in ip_dooptions bsd/netinet/ip_input.c:3577:2
    #6 0x7f721b1f34db in ip_input bsd/netinet/ip_input.c:2230:34
    #7 0x7f721b4c7740 in ip_input_wrapper fuzz/backend.c:132:3
    #8 0x4e05b9 in DoIpInput fuzz/net_fuzzer.cc:610:7
    #9 0x4e1ca3 in TestOneProtoInput(Session const&) fuzz/net_fuzzer.cc:720:9
```

Reproducing the Old Findings: ICMP

```
commands {
  ip_input {
    raw_ip4:
"M\001\000I\001\000\000\000\000\000\000\000III\333\333\333\333\333\333\333\333\333\333IIIIIIIIIIIIIII\000\000\000\000\000III\333\
333\333\333\333\333\333\333\333\333\333\333IIIIIIIIIIIIIII"
  }
}
data_provider: ""
```

Reproducing the Old Findings: MPTCP

```
==875==ERROR: AddressSanitizer: attempting free on address which was not malloc()-ed: 0x619001b86fdc in thread T0
    #0 0x4a6d72 in free /b/s/w/ir/cache/builder/src/third_party/llvm/compiler-rt/lib/asan/asan_malloc_linux.cpp:127:3
    #1 0x7fc03b46f21d in _FREE /source/build3/../fuzz/zalloc.c:288:36
    #2 0x7fc03af5eaab in mptcp_session_destroy /source/build3/../bsd/netinet/mptcp_subr.c:742:3
    #3 0x7fc03af165bc in mptcp_gc /source/build3/../bsd/netinet/mptcp_subr.c:4615:3
    #4 0x7fc03aed2944 in mp_timeout /source/build3/../bsd/netinet/mp_pcb.c:118:16
    #5 0x7fc03b4644ea in clear_all /source/build3/../fuzz/backend.c:83:3
    #6 0x4ee9e1 in TestOneProtoInput(Session const&) /source/build3/../fuzz/net_fuzzer.cc:1010:3

0x619001b86fdc is located 348 bytes inside of 920-byte region [0x619001b86e80,0x619001b87218)
allocated by thread T0 here:
    #0 0x4a7152 in calloc /b/s/w/ir/cache/builder/src/third_party/llvm/compiler-rt/lib/asan/asan_malloc_linux.cpp:154:3
    #1 0x7fc03b46cac6 in zalloc /source/build3/../fuzz/zalloc.c:36:10
    #2 0x7fc03aed3637 in mp_pcballoc /source/build3/../bsd/netinet/mp_pcb.c:222:8
    #3 0x7fc03af84426 in mptcp_attach /source/build3/../bsd/netinet/mptcp_usrreq.c:211:15
    #4 0x7fc03af75880 in mptcp_usr_attach /source/build3/../bsd/netinet/mptcp_usrreq.c:128:10
    #5 0x7fc03a6b33ac in socreate_internal /source/build3/../bsd/kern/uipc_socket.c:784:10
    #6 0x7fc03a6b4b8f in socreate /source/build3/../bsd/kern/uipc_socket.c:871:9
    #7 0x7fc03a722be6 in socket_common /source/build3/../bsd/kern/uipc_syscalls.c:266:11
    #8 0x7fc03a72242a in socket /source/build3/../bsd/kern/uipc_syscalls.c:214:9
    #9 0x7fc03b463a13 in socket_wrapper /source/build3/../fuzz/syscall_wrappers.c:371:10
    #10 0x4e95c8 in TestOneProtoInput(Session const&) /source/build3/../fuzz/net_fuzzer.cc:655:19
```

MPTCP Raw Testcase

```
commands {
  socket {
    domain: AF_MULTIPATH
    so_type: SOCK_STREAM
    protocol: IPPROTO_IP
  }
}
commands {
  connectx {
    socket: FD_0
    endpoints {
      sae_srcif: IFIDX_CASE_0
      sae_srcaddr {
        sockaddr_generic {
          sa_family: AF_MULTIPATH
          sa_data:
"\377\377\377\377\377\377\377\377\377\377\377\377\377\377\377\377\377\377\377\377\377\377\377\377\377\377\377\377\377\377\377\377\377\377\377\377\377\377\377\377\377\377\377\377\377\377\377\377\377\377\377\377\377\377\377\377\377\377\377\377\377\377\377\377\377\377\377\377\
377\377\377\377\377\377\377\377\377\377\377\377\377\377\377\377\377\377\377\377\377\377\377\377\377\377\377\377\377\377\377\377\377\377\377\377\377\377\377\377\377\377\377\377\377\377\377\377\377\377\377\377\377\377\377\377\377\377\377\377\377\377\377\377\377\377\377\37
7\304"
        }
      }
      sae_dstaddr {
        sockaddr_generic {
          sa_family: AF_MULTIPATH
          sa_data: ""
        }
      }
    }
    associd: ASSOCID_CASE_0
    flags: CONNECT_DATA_IDEMPOTENT
    flags: CONNECT_DATA_IDEMPOTENT
    flags: CONNECT_DATA_IDEMPOTENT
  }
}
commands {
  connectx {
    socket: FD_0
    endpoints {
      sae_srcif: IFIDX_CASE_0
      sae_dstaddr {
        sockaddr_generic {
          sa_family: AF_MULTIPATH
          sa_data: ""
        }
      }
    }
    associd: ASSOCID_CASE_0
    flags: CONNECT_DATA_IDEMPOTENT
  }
}
commands {
  connectx {
    socket: FD_0
    endpoints {
      sae_srcif: IFIDX_CASE_0
      sae_srcaddr {
        sockaddr_generic {
          sa_family: AF_MULTIPATH
          sa_data: ""
```

MPTCP After Minimization

```
commands {
  socket {
    domain: AF_MULTIPATH
    so_type: SOCK_STREAM
    protocol: IPPROTO_IP
  }
}
commands {
  connectx {
    socket: FD_0
    endpoints {
      sae_srcif: IFIDX_CASE_1
      sae_dstaddr {
        sockaddr_generic {
          sa_family: AF_MULTIPATH
          sa_data:
"bugmbuf_debutoeloListen_dedeloListen_dedebuloListete_debugmbuf_debutoeloListen_dedeloListen_dedebuloListeListen_dedebuloListe_d
trte" # string length 131
        }
      }
    }
    associd: ASSOCID_CASE_0
  }
}
data_provider: ""
```

Conclusions

- Will open source this ASAP, not quite ready as of presentation date.
- Usermode unit testing isn’t as hard as it looks, and provides real value.
- Mocking and faking behavior works can work for the kernel by linking in test-only support libraries.
- Kernel fuzzing recommendations
  - Write user mode or similarly lightweight unit tests that are run per-commit (see KUnit + UML)
  - Maintain public XNU branch, upstream this fuzzer in stages
    - Proprietary code is okay: see AOSP w/ internal private branches