MWR InfoSecurity

Biting the Apple that feeds you

macOS Kernel Fuzzing

Alex Plaskett (@alexjplaskett) / James Loureiro (@NerdKernel)

LABS

Agenda

- System call fuzzing (OSXFuzz)
- Scaling up
- Code coverage
- IOKit and Mach fuzzing (OPALROBOT)
- Fuzzer comparisons
- Conclusion

OSXFuzz

MWR InfoSecurity

LABS

Kernel fuzzer – basic principles

MWR LABS

- Based on MWR Windows Kernel fuzzer
- Presented at DEFCON 2016
  - (https://github.com/mwrlabs/KernelFuzzer)
- Want to identify vulns for privilege escalation
  - Sandbox escapes
  - r00t
- Effective, scalable fuzzer
- Can we use same principle on macOS?

Kernel fuzzer – basic principles

MWR LABS

- Functions return ‘fuzzed’ values

```
bool_t get_fuzzed_bool (void);
char8_t get_fuzzed_char8 (void);
char16_t get_fuzzed_char16 (void);
```

- Random but not ‘too random’
- Calls fail when arguments don’t make sense
- Predefined list of ‘good’ values per data type
- Increases likelihood of succeeding

Kernel fuzzer – object database

- Allows us to store specific objectives (such as file descriptors)
- We create valid objects at launch of fuzzer
- Use and add new objects created by fuzzer (if valid) to be reused
- We can ask for an object of a specific type
  - Allows us to go through checks to make sure the types are correct

```
h_bh_iosurfacegetid = get_random_object_by_name("iosurfaceref");

h_BH_IOServiceOpen_service_connect = get_random_object_by_name("io_connect_t");
```

Kernel fuzzer – object database

MWR LABS

- Generate objects at fuzzer launch

```
for (fd_idx = 0; fd_idx < 64; fd_idx += 1) {
      while(HANDLES[fd_idx] == 0x0000000000000000) {
         switch (rand() % 6){
         case 0:
            temp_fd = open("/dev/random", O_RDWR);
         ...
         HANDLES[fd_idx] = temp_fd;
               temp_fd = -1;
```

```
 tempobject = (OBJECT)get_io_connect_t();
                  logger("//[Handler_Function]:
make_OBJECTS : n = %u, object = 0x%08X,
OBJECT_CREATOR[n] = %s", object_idx, tempobject,
"io_connect_t");
                 OBJECTS[object_idx] = tempobject;
                 OBJECT_CREATOR[object_idx] =
"io_connect_t";
                  tempobject = (OBJECT)-1;
                  break;
```

Kernel fuzzer – object database

- Objects held within an object struct
- Value = real value
- Index = Offset into array (so we can recall during a repro run)
- Tag = so we can get the correct object type

```
typedef struct {
    object value;
    int index;
    char *tag;
} bh_object;
```

Kernel fuzzer – syscall fuzzing

MWR LABS

- BSD Syscalls pulled from ‘syscalls.master’
- Mach traps also added as separate syscall table
- Auto created array from modifying dyjakan’s script
- Convert to known types
  - Int
  - Char
  - Bool
- Else Void* and hope…

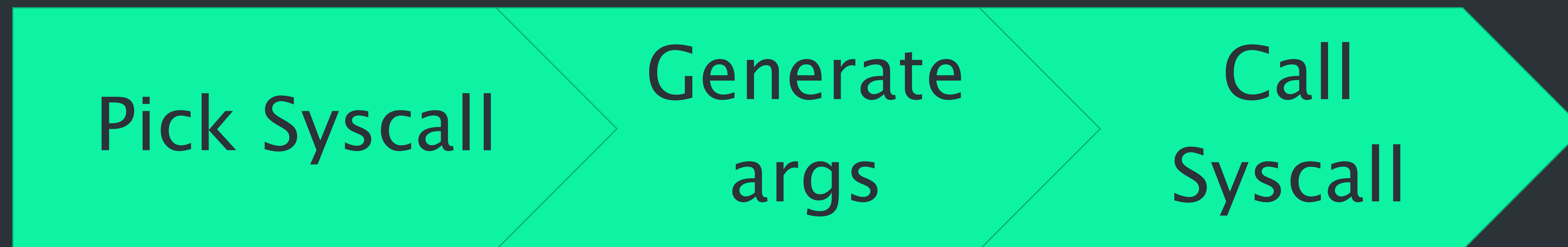
Kernel fuzzer – syscall fuzzing

MWR LABS

- Use syscall() function for calling
- Simple and easy

```
int ret = syscall(SYS_syscall, arg1, arg2, ..);
```

Kernel fuzzer – syscall fuzzing



Kernel fuzzer – syscall fuzzing

- Worked OK for some basic syscalls
- Most failed (not executing)
- Arguments didn't make sense
- Mach traps cannot be called like this

Kernel fuzzer – syscall fuzzing take 2

MWR LABS

- Time to stop being lazy…
- Write each syscall individually

Kernel fuzzer – syscall fuzzing take 2

MWR LABS

- Each syscall is a function
- Same principle as before, but we ensure arguments are correct
  - We create structs populated with fuzzed data
- This ensure the arguments are roughly correct
- More likely the syscall will execute

Kernel fuzzer – BSD and Mach syscalls

MWR LABS

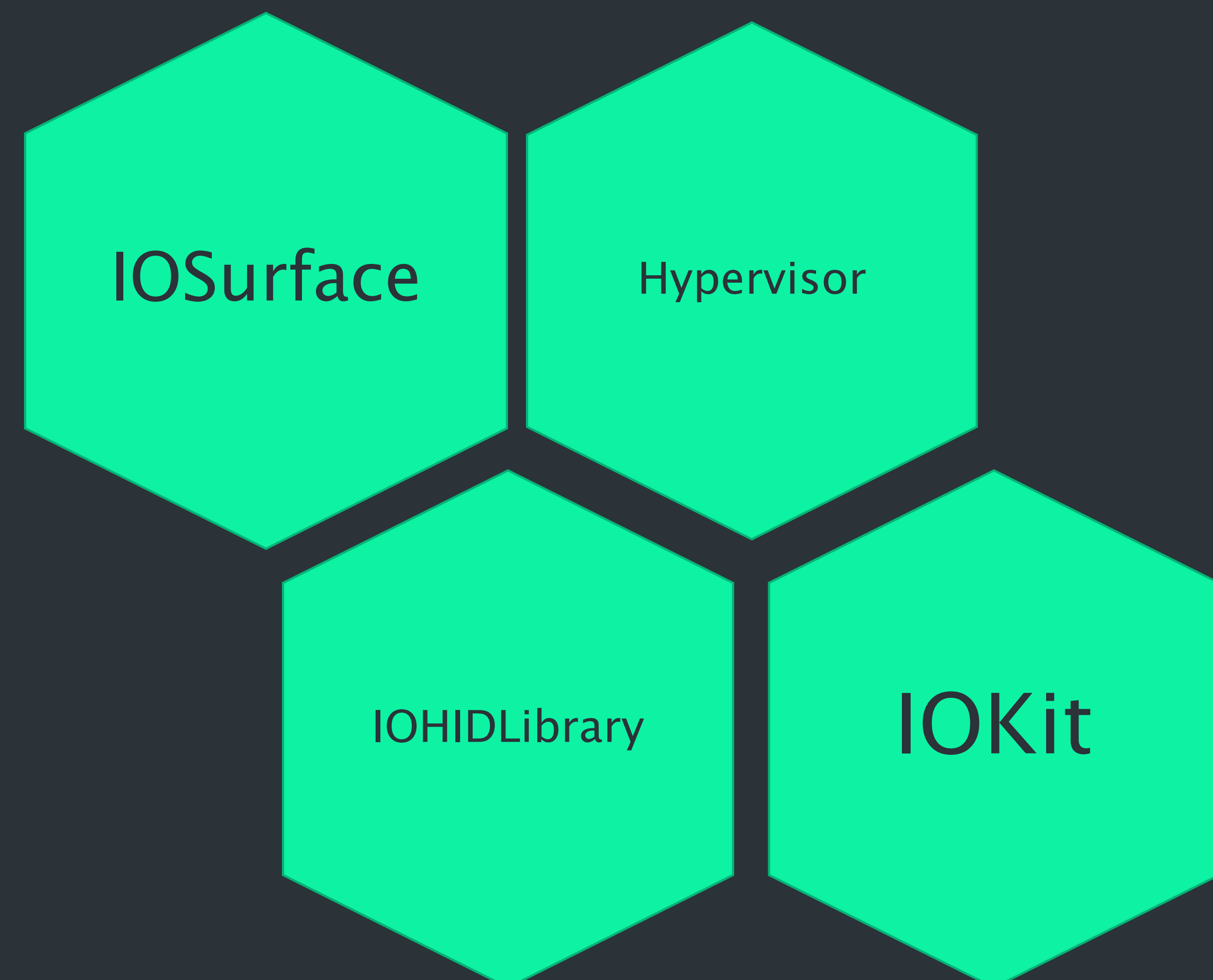
- Generate ‘variable id’ for logs
- Get integer
- Log integer
- Log syscall
- Execute syscall
- Log return value

```
void bh_aue_exit()
{
    char vid[16];
    sprintf(vid,"%u",get_time_in_ms()+rand());

    int rand_int = get_fuzzed_int32();
    logger("int rand_int%s = %d;", vid, rand_int);

    logger("syscall(SYS_exit, rand_int%d);", vid);
    int ret = syscall(SYS_exit, rand_int);

    return_logger("SYS_exit", ret);
}
```

Kernel fuzzer – Library Calls

MWR LABS

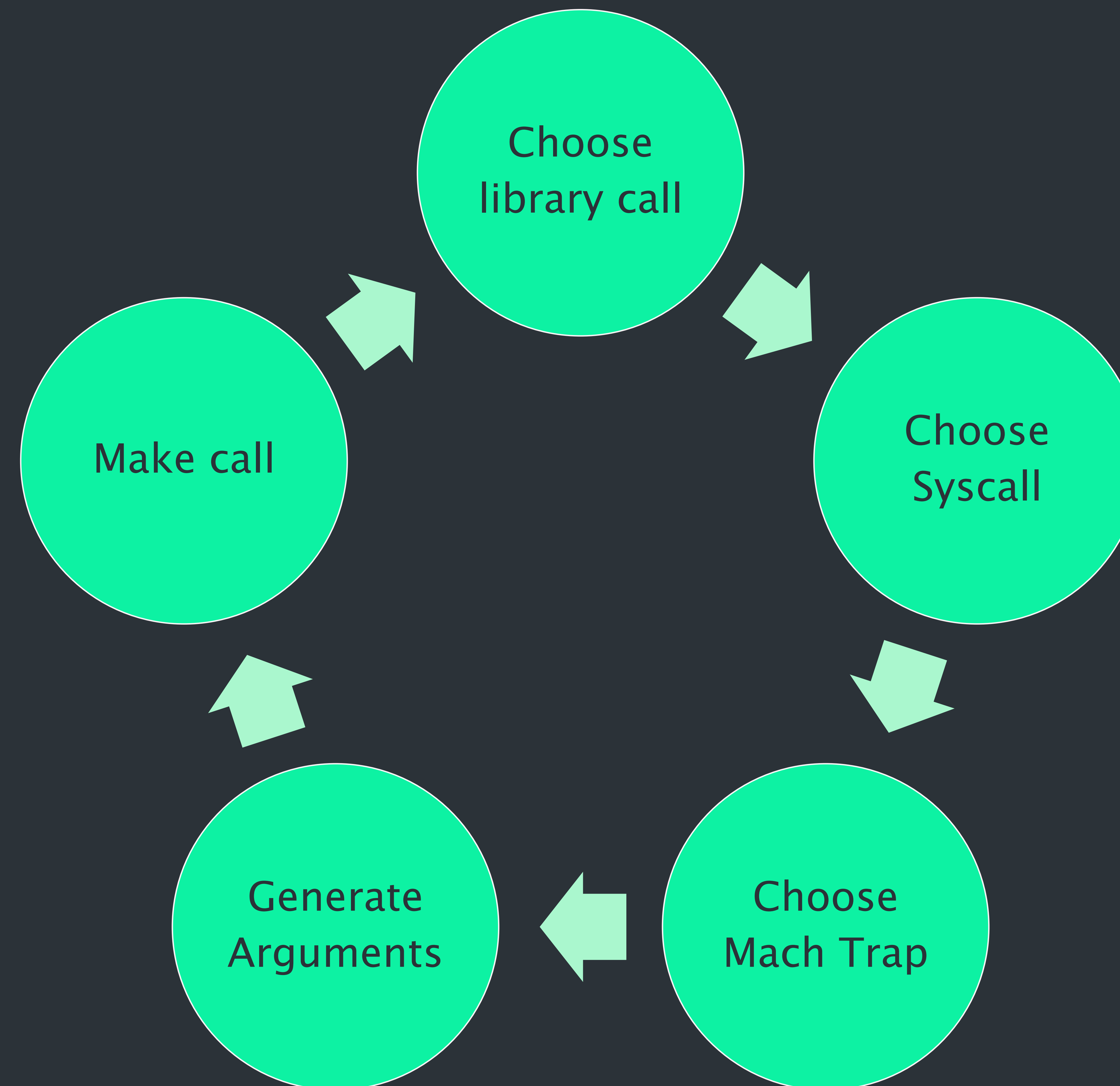
- Similar approach to syscall’s
- Catalog of common API calls




```
void BH_IOConnectAddRef()
{
    BH_Object h_BH_IOConnectAddRef = {0};
    int ret = -1;

    char vid[16];
    sprintf(vid,"%u",get_time_in_ms()+rand());
    logger("io_service_t h_BH_IOConnectAddRef%s = 0;",vid);

    h_BH_IOConnectAddRef =
get_random_object_by_name("io_service_t");
    logger("h_BH_IOConnectAddRef%s =
get_specific_object(%d);",vid,h_BH_IOConnectAddRef.index);

    logger("//[Library Call]: BH_IOConnectAddRef");
    logger("IOConnectAddRef(h_BH_IOConnectAddRef%s);",vid);

    ret =
IOConnectAddRef((io_service_t)h_BH_IOConnectAddRef.value);
    return_logger("IOConnectAddRef", ret);
}
```

Kernel fuzzer – Fuzz loop

MWR LABS



Kernel fuzzer – Logging

MWR LABS

- We log valid C
- We can then quickly build a crashing test case
- Does make creating logging statements slightly more difficult…
- Sent over network port to fuzzer control
- We wait until we receive response so we know log has been sent
  - Avoids us having non reproducible test cases

Kernel fuzzer – Logging

- We also log a ‘seed’ value
  - We use rand() for all decisions
  - We can replay fuzzer by seeding rand with same number

```
mach_port_t h_BH_IOCatalogueSendData1363174862 = 0;
uint32_t flag1363174862 = 0;
const char *buf1363174862 =
"<array><dict><key>IOProviderClass</key><string>ZZZZ</string><key>ZZZZ</key><array><stri
ng>AAAAAAAAAAAAAAAAAAAA</string></array></dict></array>";
uint32_t size1363174862 = 8;
h_BH_IOCatalogueSendData1363174862 = get_specific_object(41);
//[Library Call]: IOCatalogueSendData
IOCatalogueSendData(get_specific_object(41),flag1363174862,buf1363174862,size1363174862);
//Func:IOCatalogueSendData returned: -536870211
```

Scaling

LABS

MWR InfoSecurity

VM Automation (Fusion)

MWR LABS

- We want to run at scale – more likely to get bugs
- We want to auto capture bugs, get kernel dumps, revert the VM
- Python wrapper scripts control everything…
- vmrun for Fusion automation:

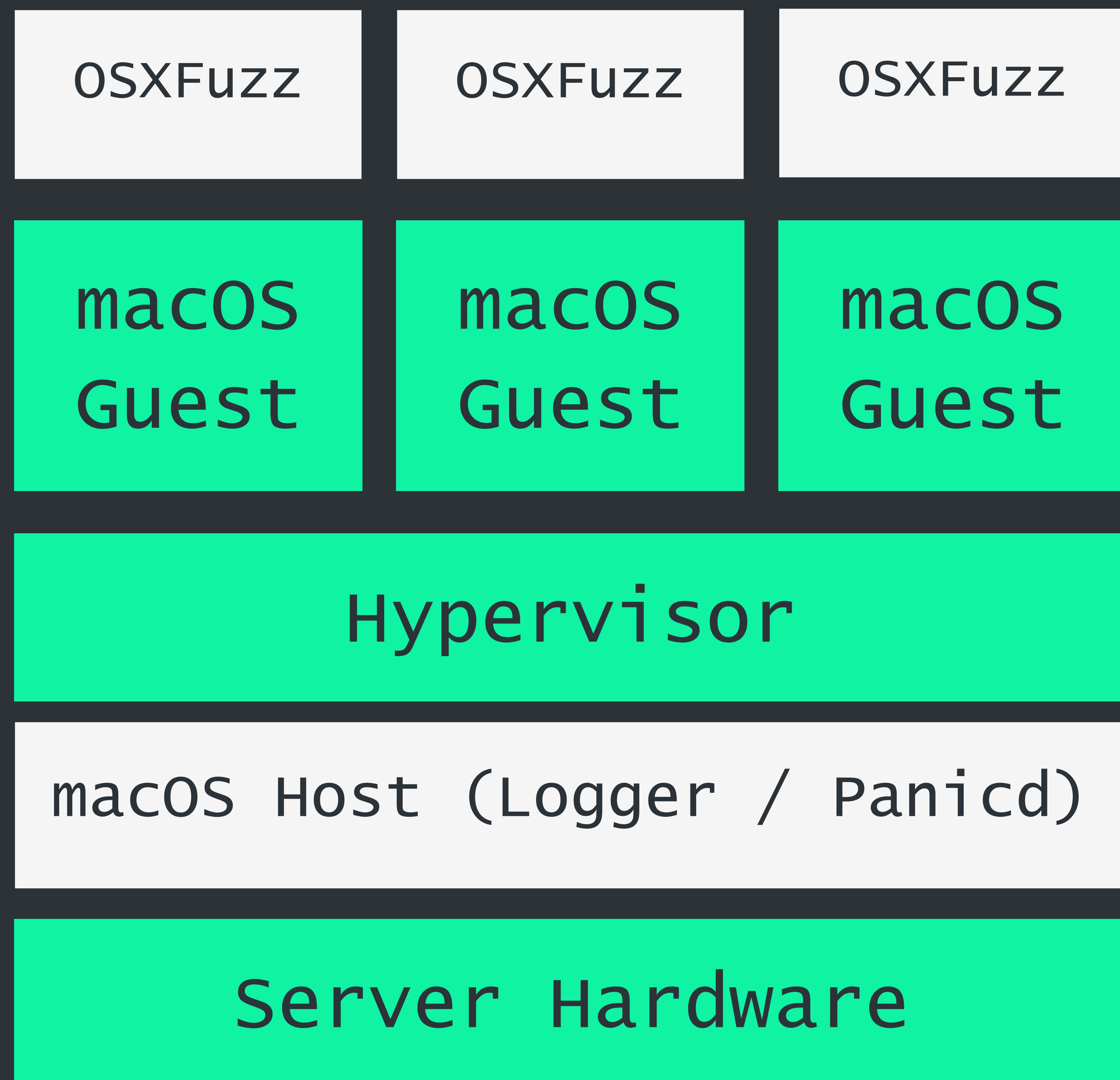
```
vmrun -T fusion revertToSnapshot vmx_path prepd
vmrun -T fusion start vmx_path
```

VM Automation (QEMU)

MWR LABS

- Must run on Mac Hardware, which we obviously do ☺
- Allowed us to investigate code coverage support
- Some challenges:
  - OVMF/Clover (nvram support)
  - virtio-net (IOKernelDebug interface)
  - Memory snapshot support

VM Automation (Fusion)



nvram boot-args=-v debug=0x2444 keepsyms=1 -zp -zc _panicd_ip=192.168.0.2

Code coverage

MWR InfoSecurity

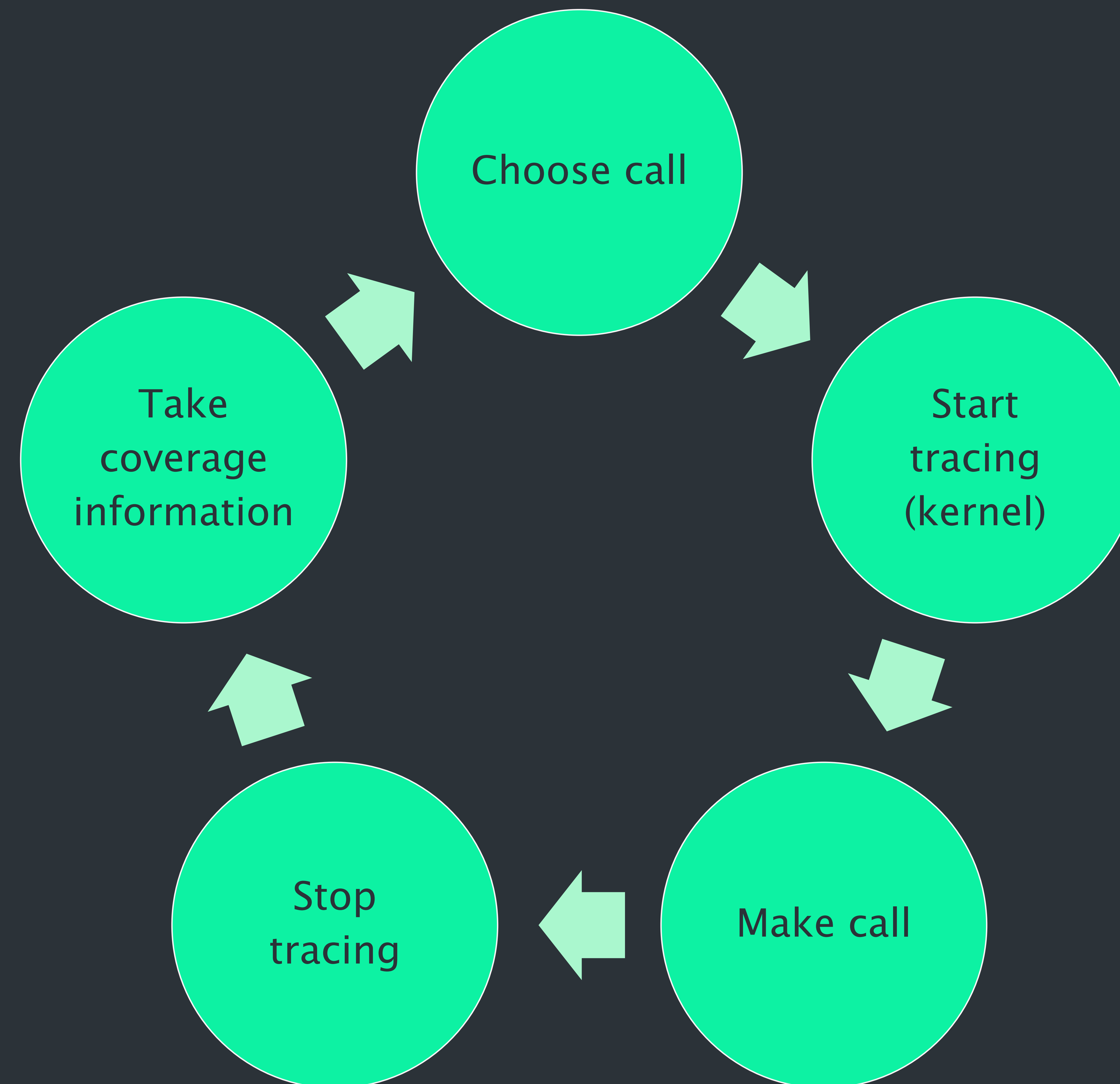
LABS

Code coverage

- We utilise NCC's Triforce in order to gain code coverage
- Only a basic setup at the moment
- Backport Qemu patches to support latest MacOS Sierra
- Will push changes to Triforce

Code coverage

MWR LABS



Code coverage

- We take the coverage and call to understand if the call hit new paths
- If yes we keep the call and arguments for future runs
- This allows us to ensure that we are not wasting future cycles with something that doesn't add coverage…
- Needs a fair bit of work (our original design didn't take into account code coverage use case)

In-memory fuzzing

MWR InfoSecurity

LABS

Common IOKit/Mach Vulnerability Classes

MWR LABS

- Idea was to focus on commonly found issues
  - IOConnectCallMethod issues
  - IORegistry properties
  - Shared memory mapping problems
  - Mach message handling
  - TOCTOU
- Combine static binary analysis with dynamic analysis
- Do as much as we can in-memory without having to touch disk.

Python In-Memory Fuzzing

- Multiple components
  - CORALSUN – Cython IOKIT/Mach utility library
  - KEXTLib – IDA Python static binary analysis scripts
  - OPALROBOT – Fuzzing/dynamic sniffing harness
- Codenames since it seems to be the in-thing with security these days! ☺
- There are similar approaches but limited code actually released.

CORALSUN (Cython Library)

MWR LABS

- Wrapper common functionality used for fuzzing (Python => C)
- Make it easy to test ideas out.

Python	C function
open_service	IOServiceOpen
connect_call_method	IOConnectCallMethod
map_sharedmemory	IOConnectMapMemory
mach_msg_send	send_mach_msg
ioconnect_setproperty	IOConnectSetCFProperty
ioregistry_setproperty	IORegistrySetCFProperty

Cython implementation

```
def connect_call_method(self,conn,selector,scalar_input,structInput,scalar_output,struct_output):
        print("Connect call method called")
        cdef uint64_t *input_scalar
        cdef uint32_t input_scalar_size
        cdef uint32_t output_scalar_size
        …

        # Handle the input scalar first.
        if type(scalar_input) is list:
            print("++ An input list was passed ++")

            input_scalar_len = len(scalar_input)
            print("input scalar len = %d" % input_scalar_len)

            input_scalar_size = input_scalar_len
            i = 0
            input_scalar = <uint64_t *>malloc(input_scalar_size * sizeof(uint64_t))
            if not input_scalar:
                raise MemoryError()

            # Convert the python array to native.
            for elem in scalar_input:
                if isinstance(elem, (int, long)):
                    input_scalar[i] = elem
                    i += 1

        if isinstance(structInput, basestring):
            inputStructCnt = len(structInput)
            # First convert the python string to char * string to get at raw data
            inputStruct = structInput

        kr =
IOConnectCallMethod(conn,selector,input_scalar,input_scalar_size,inputStruct,inputStructCnt,output_scalar,&output_scalar_size,outputSt
ruct,&outputStructCnt)
```

Python wrapper

MWR LABS

IOConnectCallMethod

```
iokit = iokitlib.iokit()
h = iokit.open_service("IntelAccelerator",5)

input_scalar = None
input_struct =
binascii.unhexlify("3f0000000100000000000000000000000000000000000000")

output_scalar = None
output_struct =
binascii.unhexlify("3f0000000100000000000000000000000000000000000000")

selector = 11

kr =
iokit.connect_call_method(h,selector,input_scalar,input_struct,output_scalar,o
utput_struct)
```

IOConnectMapMemory

```
iokit = iokitlib.iokit()
h = iokit.open_service("IOFramebuffer",1)
kr = iokit.map_sharedmemory(h,100,4096)
iokit.set_sharedmemory(kr,"BBBBBBBBBBBB");
memory = iokit.dump_sharedmemory(kr,4096);
```

IOConnectSetCFProperty

```
iokit = iokitlib.iokit()
h = iokit.open_service("IOFramebuffer",1)
kr = iokit.ioconnect_setproperty(h,“BBBBB","TEST")
```

IORegistrySetCFProperty

```
iokit = iokitlib.iokit()
h = iokit.open_service("IOFramebuffer",1)
kr = iokit.ioregistry_setproperty(h,“BBBBB","TEST")
```

send_mach_msg

```
iokit = iokitlib.iokit()
data =
binascii.unhexlify(“13151300010000000b4b000007070000030c00002327000000000000000”
);
service = “com.apple.CoreServices.coreservicesd”
ret = iokit.send_mach_msg(service,data);
```

Static Binary Analysis Flow Idea

MWR LABS

- Focuses on macOS kernel extensions (KEXTs)
- General idea is to automate the extraction of details from KEXT
- Using IDA Python / Sark for this task currently.
- Build JSON output of attack surface which can be consumed by fuzzer
- Batch run against all the kernel extensions to generate JSON.

KEXTLib Algorithms (IDA Python)

MWR LABS

- What do we want to pull out?
  - All IOServices and IOUserClients
  - Dispatch Tables (IOExternalMethod and IOExternalMethodDispatch)
  - Shared memory mapping setup calls (createMappingInTask)
  - IORegistry property getters (getProperty, copyProperty)

```
struct IOExternalMethodDispatch
{
    IOExternalMethodAction function;
    uint32_t checkScalarInputCount;
    uint32_t
checkStructureInputSize;
    uint32_t checkScalarOutputCount;
    uint32_t
checkStructureOutputSize;
};
```

```
struct IOExternalMethod {
    IOService *         object;
    IOMethod            func;
    IOOptionBits        flags;
    IOByteCount         count0;
    IOByteCount         count1;
};
```

KEXTLib Algorithms (Example)

MWR LABS

- How do we do this? (Pass one: using xrefs and code flow)

IOService Matching	IOExternalMethod Matching	IOExternalMethodDispatch Matching
For each function in code segment: if “newUserClient” in name: Find xrefs from function Find type constants and xrefs Add potential userclient	For each _const segment: For each line in disassembly: If line xref to data: if “getTargetAndMethodForIndex” or “getExternalMethodForIndex” in line: Process the table using pattern matching	For each function in code: if “externalMethod” in name: for each xref from: Determine if potential IOExternalmethod

KEXTLib Algorithms (Example)

MWR LABS

- How do we do this? (Pass two: dumb pattern matching on structs)

IOExternalMethodDispatch	IOExternalMethod
For each line within _const segment: If the line has an xref_to code segment and is followed by 4 32bit integers: Then potentially an IOExternalMethodDispatch struct	For each line within _const segment: If the line has an xref_to code segment and is followed by 4 64bit integers or a pointer followed by 3 64bit integers: Then potentially a IOExternalMethod

- Works for a large number of KEXTs reasonably well
- Some KEXT's do their own thing..
- Could be improved (e.g. vtable matching) but I was in a hurry ☺

KEXTLib Output (IntelAccelerator.json)

MWR LABS

IOConnectCallMethod

```
{
    "IOServices": [
    {
            "id":"IntelAccelerator",
            "IOUserClients": [
                    {
                    "id":"IGAccelSurface",
                    "type": 0,
                    "IOExternalMethodDispatch" : [

                            {"selector": 0, "checkScalarInputCount": 1, "checkStructureInputSize": 232, "checkScalarOutputCount": 0,
"checkStructureOutputSize": 968},
                            {"selector": 1, "checkScalarInputCount": 1, "checkStructureInputSize": 0, "checkScalarOutputCount": 0,
"checkStructureOutputSize": 0},
                            {"selector": 2, "checkScalarInputCount": 1, "checkStructureInputSize": 12, "checkScalarOutputCount": 0,
"checkStructureOutputSize": 968},
                            {"selector": 3, "checkScalarInputCount": 1, "checkStructureInputSize": 12, "checkScalarOutputCount": 0,
"checkStructureOutputSize": 4},
                            ….
                    ]
                    },
            ]
    } ]
}
```

IOConnectMapMemory

```
{
        "id":"IGAccel2DContext",
        "type": 2,
        "IOConnectMapMemoryTypes" : [0,2],
},
```

MWR

Dynamic Analysis Flow Idea

MWR LABS

- Supplement the static binary analysis with actual valid data.
- Make it easily extensible (JavaScript)
- Kernel capture does work but reproduction / logging harder in my opinion.
- Know we can definitely trigger issue from userspace.

OPALROBOT

- **Sniffing Module**
  - Capture and replay of IOKit calls, Mach messages etc.
    - Hook key functions and dump the data
    - Pickle up the data and store it..
    - Provide a way to build JSON from it or replay.
- **Fuzzing Module:**
  - Loads either pickled data or JSON for mutation.
  - Mutate and test data

FRIDA

OPALROBOT

FRIDA MWR LABS

- Function hooking example (IOConnectCallMethod)

```
Interceptor.attach(Module.findExportByName("IOKit", "IOConnectCallMethod"), {
            onEnter: function (args) {
                         console.log("IOConnectCallMethod called");
                         var connection = args[0].toInt32();
                         var selector = args[1].toInt32();

                         // Scalar input arguments
                         var input_scalar = args[2];                                   // const uint64_t *input
                         var input_scalar_count = args[3].toInt32();                   // uint32_t inputCnt

                         // Struct input arguments
                         var input_struct = args[4];                                   // const void *inputStruct
                         var input_struct_count = args[5].toInt32();                   // size_t inputStructCnt
                         // Scalar output arguments
                         var output_scalar = args[6];                                  // uint64_t *output
                         var output_scalar_count = 0;                                  // uint32_t outputCnt
…

 payload = {
               "service_name" : user_client,
               "service_type" : user_client_type,
               "selector" : selector,
               "input_scalar" : input_scalar_arr,
               "input_scalar_count" : input_scalar_count,
               "input_struct_count" : input_struct_count,
               "output_scalar_count" : output_scalar_count,
               "output_scalar" : output_scalar_arr,
               "output_struct_count" : output_struct_count,
};

send(payload,data);
```

OPALROBOT

FRIDA MWR LABS

- Challenge is resolution of io_service_t and mach_port_t's
- Io_service_t's and mach_port_t are unique to a process
- We need to track io_service_t creation (IOServiceOpen) and mach_port_t lookups.
- Solution is to hook IOServiceOpen on return and save output io_connect_t * in a lookup table with the service matcher name.
- With mach ports IPC we can hook task_get_special_port, bootstrap_look_up2, bootstrap_look_up3, bootstrap_register to put a name to a port.
- Some pitfalls with this approach as hooking is after process creation time.

OPALROBOT

FRIDA MWR LABS

- Code example:

```
Interceptor.attach(Module.findExportByName("IOKit", "IOServiceOpen"), {

        onEnter: function (args) {
            console.log("IOServiceOpen called");
            connect_ptr = args[3]; // io_connect_t *connect
            classname = Memory.alloc(256); // Temp buffer to hold class name on the heap.
            // Determine the class name of the IOKit object
            var IOObjectGetClass = Module.findExportByName(null, "IOObjectGetClass");
            var IOObjectGetClassFunc = new NativeFunction(ptr(IOObjectGetClass), 'int', ['pointer','pointer']);
            IOObjectGetClassFunc(args[0],classname);
            //console.log("IOObjectGetClass = " + Memory.readUtf8String(classname));
            type = args[2];
            console.log("IOServiceOpen(" + args[0] + "," + args[1] + "," + args[2] + "," + args[3] + ");");
        },
```

```
 onLeave: function (retval) {
            // If we have a valid connection
            if (retval == 0) {
                var handle = Memory.readU32(connect_ptr);
                var userclient = Memory.readUtf8String(classname);
                console.log("IOServiceOpen ret = " + handle);
                console.log("IOServiceOpen userclient = " + userclient);
                console.log("IOServiceOpen type = " + type);
                // Store the details in the map
                service_ids[handle] = [userclient,type];
            }
        }
 });
```

OPALROBOT (Pickles)

```
fuzzer01:pickles mwr$ ls
AppleKeyStore_0x0_0.p               IOSurfaceRoot_0x5_10.p          IntelAccelerator_0x100_256.p    IntelAccelerator_0x5_8.p
AppleKeyStore_0x0_17.p              IOSurfaceRoot_0x5_11.p          IntelAccelerator_0x100_257.p    IntelAccelerator_0x5_9.p
ApplePlatformEnabler_0x1_0.p        IOSurfaceRoot_0x5_7.p           IntelAccelerator_0x100_7.p      IntelAccelerator_0x6_0.p
IOBluetoothHCIController_0x0_0.p    IOSurfaceRoot_0x6_0.p           IntelAccelerator_0x101_0.p      IntelAccelerator_0x6_1.p
IOSurfaceRoot_0x0_0.p               IOSurfaceRoot_0x6_1.p           IntelAccelerator_0x101_1.p      IntelAccelerator_0x6_10.p
IOSurfaceRoot_0x0_1.p               IOSurfaceRoot_0x6_10.p          IntelAccelerator_0x101_10.p     IntelAccelerator_0x6_11.p
IOSurfaceRoot_0x0_10.p              IOSurfaceRoot_0x6_20.p          IntelAccelerator_0x101_15.p     IntelAccelerator_0x6_14.p
IOSurfaceRoot_0x0_11.p              IOSurfaceRoot_0x6_7.p           IntelAccelerator_0x101_2.p      IntelAccelerator_0x6_15.p
IOSurfaceRoot_0x0_13.p              IOSurfaceRoot_0x9_1.p           IntelAccelerator_0x101_20.p     IntelAccelerator_0x6_2.p
IOSurfaceRoot_0x0_14.p              IOUSBDevice_0x0_4.p             IntelAccelerator_0x101_22.p     IntelAccelerator_0x6_20.p
IOSurfaceRoot_0x0_15.p              IOUSBDevice_0x0_7.p             IntelAccelerator_0x101_9.p      IntelAccelerator_0x6_22.p
IOSurfaceRoot_0x0_16.p              IOUSBRootHubDevice_0x0_4.p      IntelAccelerator_0x102_0.p      IntelAccelerator_0x6_3.p
IOSurfaceRoot_0x0_2.p               IOUSBRootHubDevice_0x0_7.p      IntelAccelerator_0x102_2.p      IntelAccelerator_0x6_4.p
IOSurfaceRoot_0x0_20.p              IntelAccelerator_0x0_0.p        IntelAccelerator_0x102_9.p      IntelAccelerator_0x6_7.p
IOSurfaceRoot_0x0_22.p              IntelAccelerator_0x0_1.p        IntelAccelerator_0x1_2.p        IntelAccelerator_0x6_8.p
IOSurfaceRoot_0x0_3.p               IntelAccelerator_0x0_15.p       IntelAccelerator_0x1_256.p      IntelAccelerator_0x6_9.p
IOSurfaceRoot_0x0_7.p               IntelAccelerator_0x0_2.p        IntelAccelerator_0x1_257.p      IntelAccelerator_0x9_1.p
```

Key:

userclientname_userclienttype_selectornumber.p

OPALROBOT (Example)


```
(dp0
S'output_scalar'
p1
(lp2
sS'service_name'
p3
VIntelAccelerator
p4
sS'output_struct_count'
p5
I968
sS'service_selector'
p6
I0
sS'input_struct'
p7
S'd300000006000081100000008494f5375726661636548656967687400200000041800000000000000f000008494f5375726661636557696474680000200000412'
p8
sS'input_scalar'
p9
(lp10
sS'input_scalar_count'
p11
I0
sS'output_scalar_count'
p12
I0
sS'input_struct_count'
p13
I232
sS'output_struct'
p14
S'00b0589bff7f0000200000000000000009051f297ff7f000007000000000000002000000000000000a07df297ff7f000050e132f5ec7f00000000000000000000f06'
p15
sS'service_type'
p16
V0x0
p17
s.
```

OPALROBOT (Sniffing)

MWR LABS

- Increase coverage from captured data
- AXElements for UI Automation to generate captures
- DSL (Example) – Recursive UI tree iteration and perform action at random
  - Click
  - Press
- Good programs to sniff for captures:

- Blacklist UI calls like shutdown which stops VMs running ☺

OPALROBOT (Mutation)

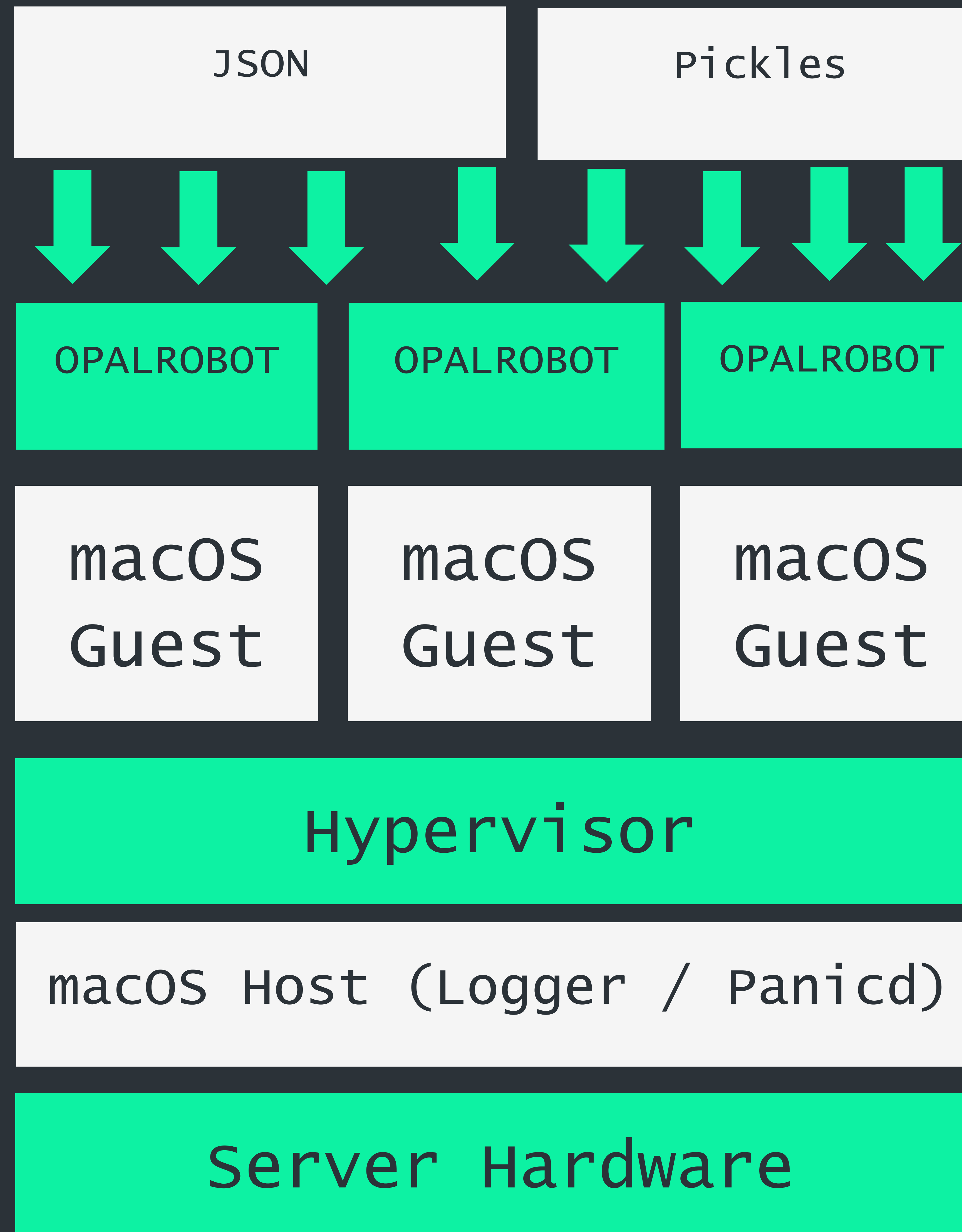
MWR LABS

- Perform operations commonly used to trigger vulnerabilities.
- General fuzzer algorithm:
  - Choose pickled data
  - Choose JSON data
  - Mutate input data
  - Make call (IOConnectCallMethod / IOConnectMapMemory).

- Leak detection for kernel pointer leaks (from pickled data and output)
- Crash detection via panicd server

OPALROBOT – Virtualized Architecture

MWR LABS



nvram boot-args=-v debug=0x2444 keepsyms=1 -zp -zc _panicd_ip=192.168.0.2

Fuzzer Comparison (so far..)

MWR LABS

OSXFuzz	OPALROBOT
Use-after-free (CVE-TBC)	Use-after-free (CVE-TBC)
Heap overflow (CVE-TBC)	Uninitialized memory call (CVE-2017-7054)
	IORegistry Issues (CVE-2017-7051)
	Out of bounds write (CVE-TBC)

*really slow reporting stuff and bug collisions ☹
** Issues to be released soon when fixed

Conclusion

MWR LABS

- Syscall fuzzer is good for UAFs in core XNU code
- IOKIT is still a good source of bugs
- Different approaches == different bug classes
- Need to focus on where we are finding bugs
  - New syscalls
  - IOKit
- Code review helped for focusing on new features
- Scaling up macOS is more challenging

Future Work

MWR LABS

- More scaling of the fuzzer
- ANGR / Manticore for improved binary analysis?
- iOS integration / automation
- Improved XPC/Mach messaging support
- Further work on code coverage / feedback

To be released

MWR LABS

- To be released:
  - Coralsun
  - OSXFuzz
- https://github.com/mwrlabs

Credits

- Lots of great previous work in this area (too many to list)
- Ian Beer https://bugs.chromium.org/p/project-zero/issues/list?can=1&redir=1
- Moony Li – https://pacsec.jp/psj16/PSJ2016_MoonyLi_pacsec-1.8.pdf
- Optimized Fuzzing IOKIT (https://www.blackhat.com/docs/us-15/materials/us-15-Lei-Optimized-Fuzzing-IOKit-In-iOS.pdf)
- The Python bytes your Apple (https://speakerdeck.com/flankerhqd/the-python-bites-your-apple-fuzzing-and-exploiting-osx-kernel-bugs)

MWR InfoSecurity

LABS