

iOS and OS X ABI

(Hacking in context)

Mikhail Sosonkin



Security Researcher at SYNACK

Working on low level emulation with QEMU and iPhone automation.

Graduate of Polytechnic University

a.k.a Polytechnic Institute of New York University

a.k.a New York University Polytechnic School of Engineering

a.k.a New York University Tandon School of Engineering



СССР 1986

Intel 8080 Clone

1.78MHz CPU

32KB RAM

2KB ROM

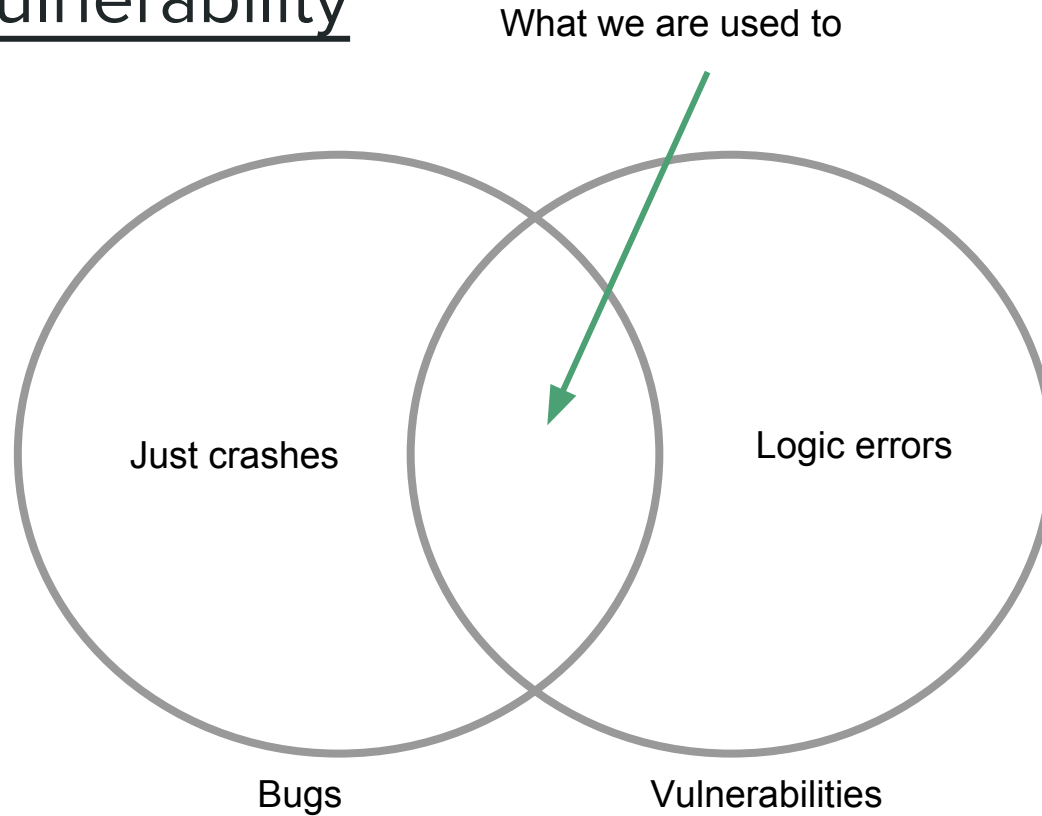
450 Rubles

[Wikipedia-RU](https://ru.wikipedia.org/wiki/СРС_1986)





What's a vulnerability





Amazon 🍄 Apple

“In short, the very four digits that Amazon considers unimportant enough to display in the clear on the web are precisely the same ones that Apple considers secure enough to perform identity verification.”

- <http://www.wired.com/2012/08/apple-amazon-math-honan-hacking/all/>



It is not enough to just be careful with your interfaces. You must also have have mitigations and continuous analysis that includes “outsiders”.

Security considerations and reviews should be part of every step of development lifecycle.



Where are the vulns?!

Memory corruption - just won't go away!

That's what a lot of CTFs seem to be focusing on.

[History thereof](#)

[Memory Errors](#)

“Special feature”

Backdooring yourself.

Someone will eventually discover it.



Network

man on the side

<http://www.wired.com/2015/04/researchers-uncover-method-detect-nsa-quantum-insert-hacks/>

web

where did I leave that session key again?

https://www.owasp.org/index.php/Top_10_2013-Table_of_Contents



Miscommunications

The root of all bugs.

Don't be too paranoid

It's not healthy, but always ask:

*“what do you do if someone compromises
this component?”*



Targeting

Classic:

Browser, Remote, Phishing

A little more advanced:

Via AWS - managed services (Exploiting external relationships)

USB - <https://srlabs.de/badusb/> i.e. Stuxnet



Beg, borrow and steal

Finding vulnerabilities

Fuzzing (AFL, Many frameworks)

Code reading (SourceInsight, Understand)

Dynamic/Static analysis (Qira, Panda)



Exploit

Control EIP

Doesn't have to be 100%

Gain execution

Binary protections like ASLR and DEP



Infect

Run shell code

Might have some ROPing to do

And, stack pivoting

Find the egg

Bigger shellcode.

Download implant

Gain persistence i.e. launch daemon



No Disclosure

Private Communities

Full disclosure

Responsible Disclosure

Coordinated Disclosure

Private Bug bounties: Google, Microsoft, Facebook

Managed Bug Bounties: Bugcrowd, HackerOne, SYNACK



Black Market Bug Bounties:

Zerodium, Vupen

Cosinc ([link](#))

HackingTeam (Probably defunct)

MitnickSecurity

Lots of secretive companies ([link](#))

A few not so secretive ([link](#))



SYNACK

Private Targets

Think easy targets

Fortune 500 Companies

Several Categories

Host, Web, Mobile

Average payout: \$690

We provide a cyber platform, Hydra!

<https://www.synack.com/red-team/>



Requires passing an assessment

SYNACK Red Team entry

If unable to pass try

BugCrowd or HackerOne



Let's say you gained execution



Goals

Build shellcode that

- Downloads a dylib.

- Injects the dylib into process.

Target OS X and iOS



Get initial info - OSX

```
mikhail — bash — 80x6
sOMEMac:~ mikhail$ uname -an
Darwin somemac 14.5.0 Darwin Kernel Version 14.5.0: Wed Jul 29 02:26:53 PDT 2015
; root:xnu-2782.40.9~1/RELEASE_X86_64 x86_64
sOMEMac:~ mikhail$
```

```
mikhail — bash — 80x6
sOMEMac:~ mikhail$ strings /usr/lib/dyld | grep PROJECT
@(#)PROGRAM:dyld PROJECT:dyld-353.2.3
sOMEMac:~ mikhail$
```



Get initial info - iOS

```
iphone-root_5s — ssh — 80x6
iPhone:~ root# uname -an
Darwin iPhone 14.0.0 Darwin Kernel Version 14.0.0: Wed Jun 24 00:50:15 PDT 2015;
root:xnu-2784.30.7~30/RELEASE_ARM64_S5L8960X iPhone6,1 arm64 N51AP Darwin
iPhone:~ root#
```

```
iphone-root_5s — bash — 80x6
sOMEMac:iphone-root_5s mikHail$ strings ./usr/lib/dyld | grep PROJECT
@(#)PROGRAM:dyld PROJECT:dyld-324.1
@(#)PROGRAM:dyld PROJECT:dyld-324.1
sOMEMac:iphone-root_5s mikHail$
```



Partial source

XNU kernel

<https://opensource.apple.com/tarballs/xnu/>

Dyld source

<https://opensource.apple.com/tarballs/dyld/>

Can be compiled



ARM64 Registers

31 General purpose registers

X0 ... X30 or W0 ... W30

X31 - (zr) The Zero register

X30 - (lr) Procedure Link Register (RIP)

X29 - (fp) Frame pointer (RBP)

X18 - Reserved on iOS



ARM64 Instructions

Conditional Branches

B.EQ, B.NE, TBNZ (Test bit and Branch if Nonzero), etc.

Unconditional Branches

B, RET, SVC

Conditional Select

CSEL W9, W9, W10, EQ



“W9 = EQ?W9:W10”




Introducing: IDARef <https://github.com/nologic/idaref>

```
loc_12001E344                                ; CODE
MOV     W0, #0
ADD     X8, X8, X20
STR     X8, [X19]

loc_12001E350                                ; CODE
ge_gettimeofday+130 (Synchronized with Hex View-
```



 Instruction Reference

STR: Store Register (immediate) calculate 32-bit word or a 64-bit doubleword to the memory. For more details see Load/Store addressing modes

Post-index



Introducing: HopperRef <https://github.com/zbuc/hopperref>



Manual loaded for architecture: arm

STR: Store Register (immediate) calculates an address from a base register and stores a 32-bit word or a 64-bit doubleword to the calculated address, from memory

accesses see Load/Store addressing modes on page C1-122.



Making system calls



Calling Convention

On ARM64:

X0 ... X8 Contain function parameters

X16 has the system call number

Positive for Posix

Negative for Mach Ports

0x80000000 for thread_set_self

SVC 0x80; jumps to kernel



Let's make a system call

```
; ===== S U B R O U T I N E =====  
  
__kernelrpc_mach_vm_allocate_trap      ; CODE XREF: __mach_vm_allocate+24`p  
    MOV            X16, #0xFFFFFFFFFFFFFFF6  
    SVC            0x80  
    RET  
; End of function __kernelrpc_mach_vm_allocate_trap  
  
; ===== S U B R O U T I N E =====  
  
__pthread_set_self                      ; CODE XREF: __pthread_set_self+54`j  
    MOV            X3, #2  
    MOV            X16, #0x80000000  
    SVC            0x80  
    RET  
; End of function __pthread_set_self
```



; ----- S U B R O U T I N E -----

__kill ; CODE XREF: __pthread_abort+28'p
; _kill+4'j

var_10 = -0x10

MOV X16, #0x25
SVC 0x80
B.CC locret_12001EBD4
STP X29, X30, [SP, #var_10]!
MOV X29, SP
BL _cerror_nocancel
MOV SP, X29
LDP X29, X30, [SP+0x10+var_10], #0x10

locret_12001EBD4 ; CODE XREF: __kill+8'j

RET

; End of function __kill



Syscall numbers

OSX:

0x01000000 - mach ports

0x02000000 - Posix

0x03000003 - pthread_set_self

IOS

0x00000000 and below - mach ports

0x00000000 and above - Posix

0x80000000 - pthread_set_self



Loading Mach-O's



Who does what?

- Kernel:
 - Maps the main executable
 - Maps the loader
 - Passes control to the loader

- DYLD:
 - “Maps” itself and the main executable
 - Maps and links dependency libraries.



File Structure: Header

/usr/include/mach-o/loader.h

```
/*
 * The 64-bit mach header appears at the very beginning of object files for
 * 64-bit architectures.
 */
struct mach_header_64 {
    uint32_t    magic;        /* mach magic number identifier */
    cpu_type_t  cputype;      /* cpu specifier */
    cpu_subtype_t  cpusubtype; /* machine specifier */
    uint32_t    filetype;     /* type of file */
    uint32_t    ncmds;        /* number of load commands */
    uint32_t    sizeofcmds;   /* the size of all the load commands */
    uint32_t    flags;        /* flags */
    uint32_t    reserved;     /* reserved */
};

/* Constant for the magic field of the mach_header_64 (64-bit architectures) */
#define MH_MAGIC_64 0xfeedfacf /* the 64-bit mach magic number */
#define MH_CIGAM_64 0xcffaedfe /* NXSwapInt(MH_MAGIC_64) */
```

```
somemac:shellcode mikhail$ hexdump -C test_syscall | head -2
00000000  cf fa ed fe 0c 00 00 01  00 00 00 00 02 00 00 00  |.....|
00000010  0f 00 00 00 f0 02 00 00  85 00 20 00 00 00 00 00  |.....|
```



File Structure: Commands

```
struct load_command {  
    uint32_t cmd;        /* type of load command */  
    uint32_t cmdsize;    /* total size of command in bytes */  
};
```

- Follow the header
- 'cmdsize' is a multiple of 8 bytes.



Mach-O commands

- LC_SEGMENT and LC_SEGMENT_64
 - From file to virtual memory: __DATA, __TEXT, etc.
- LC_UNIXTHREAD
 - Sets up initial registers and stack
 - Entry point
- LC_LOAD_DYLINKER
 - Specifies the loader i.e. /usr/lib/dyld
- LC_MAIN
 - Sets up the stack
- etc



```
shellcode — bash — 80x22

    cryptoff 16384
    cryptsize 16384
    cryptid 0
    pad 0
Load command 12
    cmd LC_LOAD_DYLIB
    cmdsize 56
    name /usr/lib/libSystem.B.dylib (offset 24)
    time stamp 2 Wed Dec 31 19:00:02 1969
    current version 1225.0.0
compatibility version 1.0.0
Load command 13
    cmd LC_FUNCTION_STARTS
    cmdsize 16
    dataoff 32816
    datasize 8
Load command 14
    cmd LC_DATA_IN_CODE
    cmdsize 16
    dataoff 32824
    datasize 0
sOMEMac:shellcode mikhail$ otool -l test_syscall
```

Setting up the stack

OSX

Arguments

Environment variables

Apple variable

Generated by the kernel

(LC_UNIXTHREAD)

```
(lldb) x -c 612 $rsp
0x7fff5fbffda0: 00 00 00 00 01 00 00 00 01 00 00 00 00 00 00 00
0x7fff5fbffdb0: 58 fe bf 5f ff 7f 00 00 00 00 00 00 00 00 00 00 argp
0x7fff5fbffdc0: 60 fe bf 5f ff 7f 00 00 6c fe bf 5f ff 7f 00 00
0x7fff5fbffdd0: 7b fe bf 5f ff 7f 00 00 84 fe bf 5f ff 7f 00 00
0x7fff5fbffde0: 95 fe bf 5f ff 7f 00 00 eb fe bf 5f ff 7f 00 00
0x7fff5fbffdf0: 2e ff bf 5f ff 7f 00 00 3c ff bf 5f ff 7f 00 00 envp
0x7fff5fbffe00: 44 ff bf 5f ff 7f 00 00 58 ff bf 5f ff 7f 00 00
0x7fff5fbffe10: 62 ff bf 5f ff 7f 00 00 72 ff bf 5f ff 7f 00 00
0x7fff5fbffe20: 00 00 00 00 00 00 00 00 50 fe bf 5f ff 7f 00 00
0x7fff5fbffe30: 98 ff bf 5f ff 7f 00 00 ab ff bf 5f ff 7f 00 00
0x7fff5fbffe40: ca ff bf 5f ff 7f 00 00 00 00 00 00 00 00 00 00 apple
0x7fff5fbffe50: 2f 62 69 6e 2f 6c 73 00 2f 62 69 6e 2f 6c 73 00 /bin/ls./bin/ls.
0x7fff5fbffe60: 43 4f 4c 55 4d 4e 53 3d 31 32 38 00 48 4f 4d 45 COLUMNS=128.HOME
0x7fff5fbffe70: 3d 2f 76 61 72 2f 72 6f 6f 74 00 4c 49 4e 45 53 =/var/root.LINES
0x7fff5fbffe80: 3d 33 38 00 4f 4c 44 50 57 44 3d 2f 76 61 72 2f =38.OLDPWD=/var/
0x7fff5fbffe90: 72 6f 6f 74 00 50 41 54 48 3d 2f 75 73 72 2f 6c root.PATH=/usr/l
0x7fff5fbffea0: 6f 63 61 6c 2f 62 69 6e 3a 2f 75 73 72 2f 62 69 ocal/bin:/usr/bi
0x7fff5fbffeb0: 6e 3a 2f 62 69 6e 3a 2f 75 73 72 2f 73 62 69 6e n:/bin:/usr/sbin
0x7fff5fbffec0: 3a 2f 73 62 69 6e 3a 2f 6f 70 74 2f 58 31 31 2f :/sbin:/opt/X11/
0x7fff5fbffed0: 62 69 6e 3a 2f 75 73 72 2f 6c 6f 63 61 6c 2f 4d bin:/usr/local/M
0x7fff5fbffee0: 61 63 47 50 47 32 2f 62 69 6e 00 50 57 44 3d 2f acGP62/bin.PWD=/
0x7fff5fbffef0: 55 73 65 72 73 2f 6d 69 6b 68 61 69 6c 2f 67 69 Users/mikhail/gi
0x7fff5fbfff00: 74 2e 73 79 6e 61 63 6b 2f 41 70 70 56 69 72 74 t.synack/AppVirt
0x7fff5fbfff10: 75 61 6c 69 7a 65 72 2f 71 65 6d 75 2f 62 69 6e ualizer/qemu/bin
0x7fff5fbfff20: 2f 64 65 62 75 67 2f 6e 61 74 69 76 65 00 53 48 /debug/native.SH
0x7fff5fbfff30: 45 4c 4c 3d 2f 62 69 6e 2f 73 68 00 53 48 4c 56 ELL=/bin/sh.SHLV
0x7fff5fbfff40: 4c 3d 31 00 54 45 52 4d 3d 78 74 65 72 6d 2d 32 L=1.TERM=xterm-2
0x7fff5fbfff50: 35 36 63 6f 6c 6f 72 00 55 53 45 52 3d 72 6f 6f 56color.USER=roo
0x7fff5fbfff60: 74 00 5f 3d 2f 75 73 72 2f 62 69 6e 2f 6c 6c 64 t.__=/usr/bin/lld
0x7fff5fbfff70: 62 00 5f 5f 43 46 5f 55 53 45 52 5f 54 45 58 54 b.__CF_USER_TEXT
0x7fff5fbfff80: 5f 45 4e 43 4f 44 49 4e 47 3d 30 78 30 3a 30 3a _ENCODING=0x0:0:
0x7fff5fbfff90: 30 00 00 00 00 00 00 00 70 66 7a 3d 30 78 37 66 0.....pfz=0x7f
0x7fff5bffffa0: 66 66 66 66 65 62 30 30 30 00 73 74 61 63 6b fffffeb0000.stack
0x7fff5bffffb0: 5f 67 75 61 72 64 3d 30 78 31 30 30 30 64 36 _guard=0x10003d6
0x7fff5bffffc0: 36 36 34 35 33 31 61 64 35 00 6d 61 6c 6c 6f 63 664531ad5.malloc
0x7fff5bffffd0: 5f 65 6e 74 72 6f 70 79 3d 30 78 61 33 63 34 37 _entropy=0xa3c47
0x7fff5bffffe0: 65 62 36 66 61 33 61 63 30 33 32 2c 30 78 39 63 eb6fa3ac032,0x9c
```


Setting up the stack

iOS

Arguments

Environment variables

Apple variable

Generated by the kernel

(LC_UNIXTHREAD)

```
(lldb) x -c 612 $sp
0x16fd67ef0: 00 80 09 00 01 00 00 00 03 00 00 00 00 00 00 00 .....
0x16fd67f00: 70 7f d6 6f 01 00 00 00 7f 7f d6 6f 01 00 00 00 p.....
0x16fd67f10: 8c 7f d6 6f 01 00 00 00 00 00 00 00 00 00 00 00 ..
0x16fd67f20: 00 00 00 00 00 00 00 00 50 7f d6 6f 01 00 00 00 .....P.
0x16fd67f30: 90 7f d6 6f 01 00 00 00 a7 7f d6 6f 01 00 00 00 ..
0x16fd67f40: c6 7f d6 6f 01 00 00 00 00 00 00 00 00 00 00 00 .....
0x16fd67f50: 2f 70 72 69 76 61 74 65 2f 76 61 72 2f 72 6f 6f /private/var/roo
0x16fd67f60: 74 2f 73 68 65 6c 6c 68 61 72 6e 65 73 73 00 00 t/shellharness..
0x16fd67f70: 2e 2f 73 68 65 6c 6c 68 61 72 6e 65 73 73 00 63 ./shellharness.c
0x16fd67f80: 6f 6e 6e 65 63 74 6f 75 74 2e 62 00 64 73 00 00 onnectout.b.ds..
0x16fd67f90: 70 66 7a 3d 30 78 66 66 66 66 66 66 66 66 66 66 pfz=0xfffffffffff
0x16fd67fa0: 66 66 66 66 66 66 66 73 74 61 63 6b 5f 67 75 61 fffffff.stack_gua
0x16fd67fb0: 72 64 3d 30 78 39 30 30 33 30 36 30 62 32 37 rd=0x59003060b27
0x16fd67fc0: 36 65 39 64 30 00 6d 61 6c 6c 6f 63 5f 65 6e 74 6e9d0.malloc_ent
0x16fd67fd0: 72 6f 70 79 3d 30 78 30 34 35 32 61 31 32 31 64 ropy=0x0452a121d
0x16fd67fe0: 35 62 37 66 64 65 31 2c 30 78 61 61 64 30 34 61 5b7fde1,0xaad04a
0x16fd67ff0: 30 34 66 34 34 64 66 36 33 32 00 00 00 00 00 00 04f44df632.....
0x16fd68000: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
0x16fd68010: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
0x16fd68020: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
0x16fd68030: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
0x16fd68040: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
0x16fd68050: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
0x16fd68060: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
0x16fd68070: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
0x16fd68080: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
0x16fd68090: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
0x16fd680a0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
0x16fd680b0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
0x16fd680c0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
0x16fd680d0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
0x16fd680e0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
0x16fd680f0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
0x16fd68100: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
0x16fd68110: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
0x16fd68120: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
0x16fd68130: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
```



Segments

```
/*
 * The 64-bit segment load command indicates that a part of this file is to be
 * mapped into a 64-bit task's address space.  If the 64-bit segment has
 * sections then section_64 structures directly follow the 64-bit segment
 * command and their size is reflected in cmdsize.
 */
struct segment_command_64 { /* for 64-bit architectures */
    uint32_t    cmd;        /* LC_SEGMENT_64 */
    uint32_t    cmdsize;    /* includes sizeof section_64 structs */
    char        segname[16]; /* segment name */
    uint64_t    vmaddr;     /* memory address of this segment */
    uint64_t    vmsize;     /* memory size of this segment */
    uint64_t    fileoff;    /* file offset of this segment */
    uint64_t    filesize;   /* amount to map from the file */
    vm_prot_t    maxprot;    /* maximum VM protection */
    vm_prot_t    initprot;   /* initial VM protection */
    uint32_t    nsects;     /* number of sections in segment */
    uint32_t    flags;      /* flags */
};
```




Let's build some shellcode.

(live exercise)

[Recorded session](#)



```
#include <stdint.h>
```

```
int main() {  
    register uint64_t _num asm("x16") = 37; // POSIX Kill  
    register uint64_t _arg1 asm("x0") = 9; // SIGKILL  
    register uint64_t _arg2 asm("x1") = 1337; // PID  
  
    register uint64_t _ret asm("x0");  
  
    asm volatile (  
        "svc 0x80;\n"  
        "neg x1, x0;\n"  
        "csel x0, x0, x1, cc;\n"  
        : "=r"(_ret)  
        : "r"(_arg1), "r"(_arg2), "r"(_num)  
        : "x16", "x17");  
  
    return _ret;  
}
```

otool -t -v test_syscall

```
`xcrun --sdk iphoneos --find gcc` -Os -fno-stack-protector -  
fomit-frame-pointer -fno-exceptions -Wimplicit-isysroot  
`xcrun --sdk iphoneos --show-sdk-path` -F`xcrun --sdk  
iphoneos --show-sdk-path`/System/Library/Frameworks -  
F`xcrun --sdk iphoneos --show-sdk-  
path`/System/Library/PrivateFrameworks -arch arm64  
test_syscall.c -o test_syscall
```

```
00000000100007f9c      movz      w0, #0x9  
00000000100007fa0      movz      w1, #0x539  
00000000100007fa4      movz      w16, #0x25  
00000000100007fa8      svc       #0x80  
00000000100007fac      neg       x1, x0  
00000000100007fb0      csel      x0, x0, x1, lo  
00000000100007fb4      ret
```



Build it using GCC

- Easy to represent complex logic
- Excellent way to learn assembly skills
- Assist with reverse engineering
- Port to different architectures
- Optimization hints
- Does 90% of the work for you



Cons of using GCC

- Can get hard to make GCC avoid outputting certain bytes.
- Give up a level of control
- Can get into dependency hell
 - All the usual problems with C.
- Optimizer could get too aggressive



The Challenge



ShellCC

- <https://github.com/nologic/shellcc>
- shellcode
 - extractshell.py: Gets the bytetimes from the macho as raw shellcode
 - Makefile has build commands for shellcode ARM/x86_64
- shellharness
 - reads a file and executes the buffer.



The challenge

- Understand *injectdyld_file.c*
 - Figure out how to dynamically load a dylib
 - Can use *injectdyld_file.c* as a base
-
- Hint: you will need to read the DYLD sourcecode.



Where to learn about security?

- <https://seccasts.com/>
- <http://www.opensecuritytraining.info/>
- <https://www.corelan.be>
- youtube for conference
- Security meetups
- Just practice
 - Read/follow walkthroughs
- follow the reddit:
 - netsec
 - reverseengineering
 - malware
 - lowlevel
 - blackhat
 - securityCTF
 - rootkit
 - vrd



Getting started with iOS

- Get iPhone 5s
 - Swappa
- Apply Jailbreak
 - Install OpenSSH via Cydia
 - Use tcprelay to SSH over USB
- Start exploring
 - [debugserver](#)
- <https://github.com/iosre/iOSAppReverseEngineering>
- <https://nabla-c0d3.github.io/blog/2014/12/30/tcprelay-multiple-devices/>



Thank you!

Mikhail Sosonkin

mikhail@synack.com

[Google+](#)