A Journey Through

Exploit Mitigation Techniques on iOS

Max Bazaliy

August 4-7, 2016

DEFCON

About me

- From Kiev, Ukraine
- Staff Engineer at Lookout
- Focused on XNU, Linux and LLVM internals
- Interested in jailbreak techniques
- Worked on obfuscation and DRM in a past
- Member of Fried Apple team

Agenda

- iOS security mechanisms
- Function hooking
- iOS 8 & 9 exploit mitigations
- Bypassing code signatures
- Future attacks

iOS security mechanisms

- Memory protections
- Code signing
- Sandbox
- Secure boot process
- Privilege separation
- Kernel Patch Protection

Memory protections

- No way to change existing page permission
- Pages can never be both writable and executable
- No dynamic code generation without JIT
- Non executable stack and heap
- ASLR / KASLR

Allocating new regions

```
kern_return_t vm_map_enter(…){
...
#if CONFIG_EMBEDDED
    if (cur_protection & VM_PROT_WRITE){
        if ((cur_protection & VM_PROT_EXECUTE) && !entry_for_jit){
            printf("EMBEDDED: curprot cannot be write+execute.
               turning off execute\n”);
            cur_protection &= ~VM_PROT_EXECUTE;
        }
    }
#endif /* CONFIG_EMBEDDED */
...
}
```

http://opensource.apple.com//source/xnu/xnu-3248.20.55/osfmk/vm/vm_map.c

Changing existing regions

```
kern_return_t vm_map_protect(…){
...
#if CONFIG_EMBEDDED
    if (new_prot & VM_PROT_WRITE) {
        if ((new_prot & VM_PROT_EXECUTE) && !(curr->used_for_jit)) {
            printf("EMBEDDED: %s can't have both write and exec at
                    the same time\n", __FUNCTION__);
            new_prot &= ~VM_PROT_EXECUTE;
        }
    }
#endif
...
}
```

http://opensource.apple.com//source/xnu/xnu-3248.20.55/osfmk/vm/vm_map.c

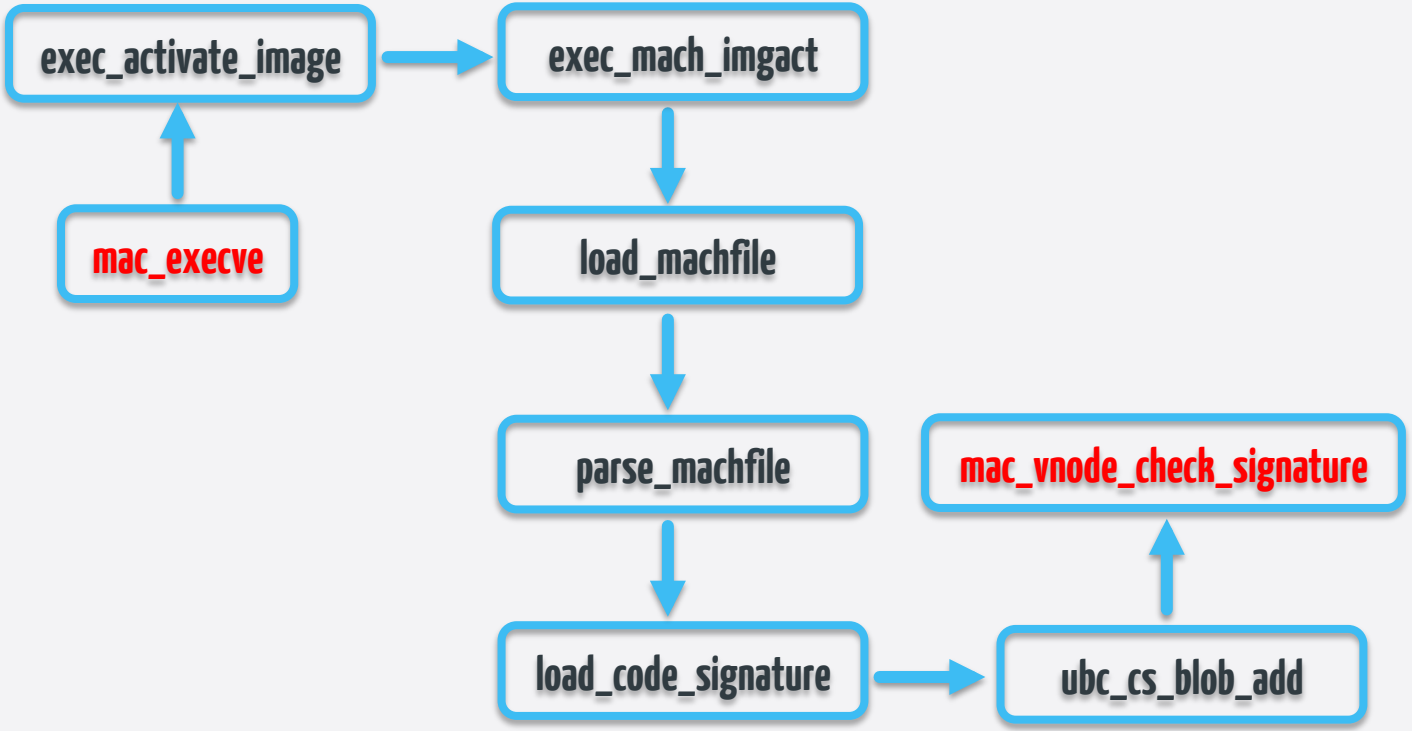
Code signing

- Mandatory Access Control Framework (MACF)
- Code must be signed by trusted party
- Signed page hashes match running code

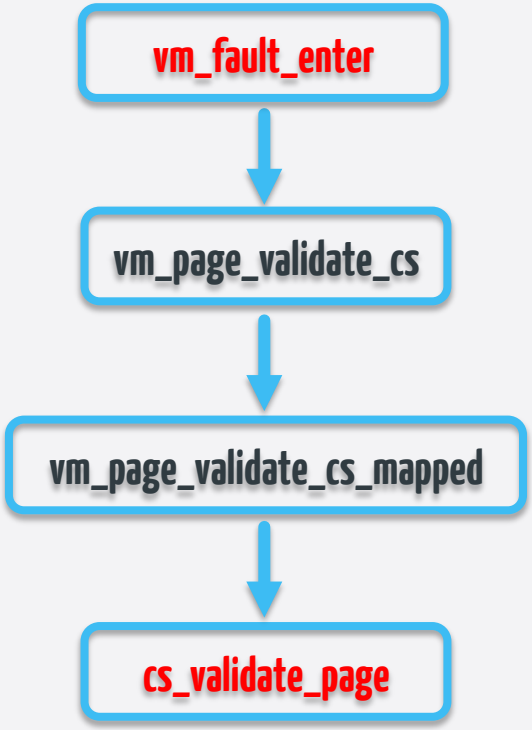
Code signature format

- LC_CODE_SIGNATURE command points to a csblob
- Key component of blob is the Code Directory
- File page hashes are individually hashed into slots
- Special slots (_CodeResources, Entitlements etc)
- CDHash: Master hash of code slots hashes

CS on load validation in kernel



CS page validation in kernel



Verifying pages

- vm_fault called on a page fault
- A page fault occurs when a page is loaded
- Validated page means that page has hash in CSDir
- Tainted page calculated hash != stored hash
- Process with invalid pages will be killed

When to verify ?

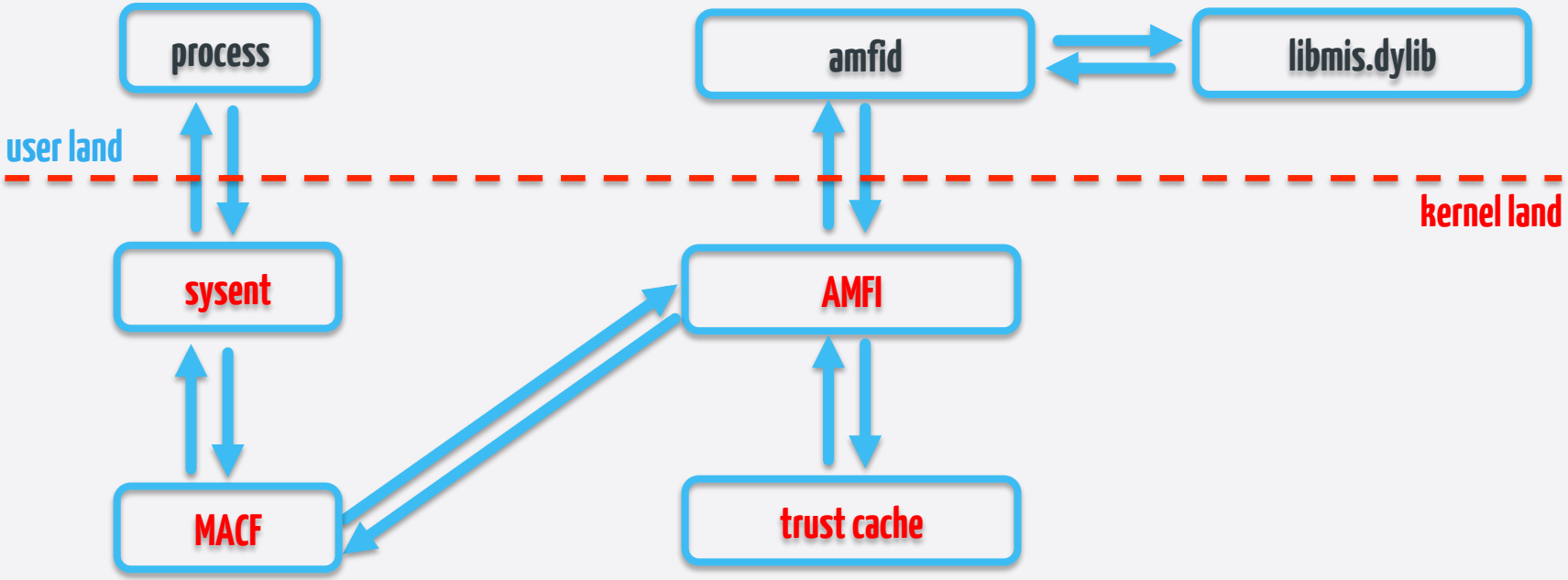
```
/*
 * CODE SIGNING:
 * When soft faulting a page, we have to validate the page if:
 * 1. the page is being mapped in user space
 * 2. the page hasn't already been found to be "tainted"
 * 3. the page belongs to a code-signed object
 * 4. the page has not been validated yet or has been mapped for write.
 */

#define VM_FAULT_NEED_CS_VALIDATION(pmap, page)                \
        ((pmap) != kernel_pmap /*1*/ &&                        \
        !(page)->cs_tainted /*2*/ &&                           \
        (page)->object->code_signed /*3*/ &&                   \
        (!(page)->cs_validated || (page)->wpmapped /*4*/))
```

Code sign enforcement

- Apple Mobile File Integrity (AMFI)
- Registering hooks in MACF
  - mpo_proc_check_get_task
  - mpo_vnode_check_signature
  - mpo_vnode_check_exec
  - …

Code sign enforcement



The story about function hooking

- Add new security features
- Debugging 3rd party code
- Logging and tracing API calls
- Reverse engineering and deobfuscation
- Interposing to the rescue

Interposing - DYLD_INFO and LINKEDIT

- Rebase Info - contains rebasing opcodes
- Bind Info - for required import symbols
- Lazy Bind Info - symbol binding info for lazy imports
- Weak Bind Info - binding info for weak imports
- Export Info - symbol binding info for exported symbols

Details - http://newosxbook.com/articles/DYLD.html

Having fun with bind info

```
case BIND_OPCODE_SET_SEGMENT_AND_OFFSET_ULEB:
      segIndex = immediate;
      address = segOffsets[segIndex] + read_uleb128(&p, end);
      break;
case BIND_OPCODE_ADD_ADDR_ULEB:
      address += read_uleb128(&p, end);
      break;
case BIND_OPCODE_DO_BIND:
      *((void **)address) = new_impl;
      address += sizeof(void *);
      break;
case BIND_OPCODE_DO_BIND_ADD_ADDR_ULEB:
      *((void **)address) = new_impl;
      address += read_uleb128(&p, end) + sizeof(void *);
      break;
```

https://opensource.apple.com/source/dyld/dyld-360.18/src/ImageLoaderMachOCompressed.cpp

dyld_shared_cache

- All frameworks and libraries
- Loaded into each process space
- Used for performance and security reasons
- ASLR slide randomized at boot time

Fixed offset in a cache

iOS 8

```
ssize_t send(int a1, const void *a2, size_t a3, int a4)
{
    return __sendto_shim(a1, (int)a2, a3, a4, 0, 0);
}
```

iOS 9

```
ssize_t send(int a1, const void *a2, size_t a3, int a4)
{
    return MEMORY[0x340480C8](a1, a2, a3, a4, 0, 0);
}
```

Fixed offset in a cache

iOS 8

```
ssize_t send(int a1, const void *a2, size_t a3, int a4)
{
    return __sendto_shim(a1, (int)a2, a3, a4, 0, 0);
}
```

iOS 9

```
ssize_t send(int a1, const void *a2, size_t a3, int a4)
{
    return MEMORY[0x340480C8](a1, a2, a3, a4, 0, 0);
}
```

What if trampolines ?

- How to change memory to RW ?
- How to switch back to RX ?
- How to pass a codesign check ?

Change a memory to RW

- What if mmap new page on a same address ?

```
void *data =
        mmap(addr & (~PAGE_MASK),
        PAGE_SIZE,
        PROT_READ|PROT_WRITE,
        MAP_ANON|MAP_PRIVATE|MAP_FIXED,
        0, 0);
```

Change a memory to RX

- What if mprotect ?

```
mprotect(addr & (~PAGE_MASK),
         PAGE_SIZE,
         PROT_READ|PROT_EXEC);
```

Sounds like a plan

- ✓ Copy original page content
- ✓ mmap new RW page over
- ✓ Copy original content back
- ✓ Write trampoline
- ✓ mprotect to RX
- ○ Do something with codesign(?)

Codesign bypass

- Page is checked on page fault
- How we can prevent page fault ?
- What if we mlock page ...

```
mlock(data & (~PAGE_MASK)), PAGE_SIZE);
```

- ... and it works!

Full attack

- ✓ Get function pointer, get page base
- ✓ memcpy page contents to temporary buffer
- ✓ mmap new RW page over
- ✓ memcpy original content back
- ✓ mlock page
- ✓ memcpy trampoline code
- ✓ mprotect page to RX

We need to go deeper

- Hook fcntl in dyld to skip codesign validation

```
fsignatures_t siginfo;
siginfo.fs_file_start=offsetInFatFile;
siginfo.fs_blob_start=(void*)(long)(codeSigCmd->dataoff);
siginfo.fs_blob_size=codeSigCmd->datasize;
int result = fcntl(fd, F_ADDFILESIGS_RETURN, &siginfo);
```

https://opensource.apple.com/source/dyld/dyld-360.18/src/ImageLoaderMachO.cpp

Loading unsiged code

- mlock all pages with executable permission during mapping

```
if ( size > 0 ) {
        if ( (fileOffset+size) > fileLen ) {
       ...
        }
        void* loadAddress = xmmap((void*)requestedLoadAddress, size,
       protection, MAP_FIXED | MAP_PRIVATE, fd, fileOffset);
        ...
        }
   }
```

https://opensource.apple.com/source/dyld/dyld-360.18/src/ImageLoaderMachO.cpp

cs_bypass

- ✓ Hook fcntl and return -1
- ✓ Hook xmmap and mlock all regions that have execution permission
- ✓ dlopen unsigned code ☺

https://github.com/kpwn/921csbypass

Future codesign attacks on dyld

- Hide executable segment
- Hook dyld functions
- Hook libmis.dylib functions

@mbazaliy