SYNACKTIV

Jailbreak detection mechanisms and how to bypass them

Pass The Salt 2021

2021/07/05

Whoami

- Eloi Benoist-Vanderbeken
- @elvanderb on twitter

- Working for Synacktiv
  - Offensive security company
  - 90 ninjas
  - 3 departments: pentest, reverse engineering, development
  - Pass The Salt sponsor!

- Reverse engineering technical leader
  - 30 reversers
  - Focus on low level dev, reverse, vulnerability research/exploitation
  - If there is software in it, we can own it :)
  - We are hiring!

Introduction

JailBreak detection

- iOS
  - Closed operating system
  - No easy way to get root
  - JailBreaks bypass iOS security to get (almost) full access
- JailBreak detection
  - Used by banking applications and games
  - To make sure that the environment is “safe”…
  - …or to block cheats/cracks
- Security researchers need to
  - Assess / reverse protected applications

iOS specificities

- Signature
  - All the code must be signed by Apple (enforced by the system)
  - All the data is also signed (enforced by the App Store)
- Memory protection
  - W^X
  - Only WebContent process can use JiT pages
- No side loading
  - “Apps may not […] download, install, or execute code which introduces or changes features or functionality of the app”
- Public API
  - “Apps may only use public APIs”
  - Theoretically enforced by the App Store review process
  - Actually only used to block malicious tracking methods or deprecated/buggys APIs

Frida

- https://frida.re
- “Dynamic instrumentation toolkit for developers, reverse-engineers, and security researchers”
- Allows you to inject JavaScript to instrument any process
  - iOS / Android / Windows / macOS / Linux / QNX...
- Lots of features
- Lots of bindings (.NET, Python, Node.js, Swift…)
- Low level C API
- Well known by Pass The Salt aficionados
  - PTS 2020 - Why are Frida and QBDI a Great Blend on Android?
  - PTS 2018 - Radare2 + Frida: Better Together

Debugging an iOS app

- **Without a JailBreak**
  - With ptrace (lldb / frida) → app needs the get-task-allow entitlement
  - By injecting code (frida) → app needs to be repackaged

    And you can only do data only instrumentation
  - In both case, you need to resign the application…
  - … but it has a lot of side effect

    Different Team ID

    File are modified
- **With a JailBreak**
  - No entitlements are required
  - Frida is able to attach to any process

    Except system ones on post A12 iPhones because of PPL

Case study

The target

- **A banking app**
- **Immediately crash when launched on a jailbroken device**
  - Exception Type: EXC_BAD_ACCESS (SIGSEGV)
  - Exception Subtype: KERN_INVALID_ADDRESS at 0x0000000000000200
- **Executable is quite large**
  - 31MB
- **Nothing special at first sight**
  - Methods name are not obfuscated
  - Strings are in cleartext
- **We tried a few scripts[1]**
  - But without luck

1: most notably this one: https://blog.spacepatroldelta.com/a?ID=01600-8a224e7e-6ceb-4e65-88b9-4545d6523275

Around the crash…


```
if ( all_is_all_right != 1 )
  ++*(_BYTE *)((unsigned __int64)&unk_101C767D0 & 0x20C);
return result;
```

Around the crash…


```
if ( all_is_all_right != 1 )
  ++*(_BYTE *)((unsigned __int64)&unk_101C767D0 & 0x20C);
return result;
```

Around the crash…


```
if ( all_is_all_right != 1 )
  ++*(_BYTE *)((unsigned __int64)&unk_101C767D0 & 0x20C);
return result;
```

```
  do
  {
    v31 = v102;
    v32 = (unsigned __int8)v101 + 1;
    v33 = (unsigned __int8)(v101 + 1);
    v34 = (unsigned __int8)v138[v33];
    v35 = v34 + (unsigned __int8)v103;
    v36 = (unsigned __int8)(v34 + v103);
    v138[v33] = v138[v36];
    v138[v36] = v34;
    encrypted_path[v31] ^= v138[v33] + (_BYTE)v34;
    v22 = (unsigned __int64)(v31 + 1) >= 0x11;
    v101 = v32;
    v102 = v31 + 1;
    v103 = v35;
    v100 = v31 - 16;
  }
  while ( v31 != 16 );
  path_is_decrypted = 1;
}
atomic_store(0, &dword_101CDDA8C);
v99 = encrypted_path;
v98 = 1LL;
v37 = mac_syscall(SYS_utimes, encrypted_path, (const timeval *)1);
```

Around the crash…


```
if ( all_is_all_right != 1 )
  ++*(_BYTE *)((unsigned __int64)&unk_101C767D0 & 0x20C);
return result;
```

```
  do
  {
    v31 = v102;
    v32 = (unsigned __int8)v101 + 1;
    v33 = (unsigned __int8)(v101 + 1);
    v34 = (unsigned __int8)v138[v33];
    v35 = v34 + (unsigned __int8)v103;
    v36 = (unsigned __int8)(v34 + v103);
    v138[v33] = v138[v36];
    v138[v36] = v34;
    encrypted_path[v31] ^= v138[v33] + (_BYTE)v34;
    v22 = (unsigned __int64)(v31 + 1) >= 0x11;
    v101 = v32;
    v102 = v31 + 1;
    v103 = v35;
    v100 = v31 - 16;
  }
  while ( v31 != 16 );
  path_is_decrypted = 1;
}
atomic_store(0, &dword_101CDDA8C);
v99 = encrypted_path;
v98 = 111;
v37 = mac_syscall(SYS_utimes, encrypted_path, (const timeval *)1);
```

Around the crash…


```
ADRL    X8, encrypted_path
MOV     W9, #1
MOV     X10, X9
STR     X8, [X19,#0x108]
STR     X10, [X19,#0x100]
LDR     X20, [X19,#0x108]
LDR     X21, [X19,#0x100]
MOV     X16, #0x8A
MOV     X0, X20
MOV     W1, W21
SVC     0x80
CSET    X23, CS
MOV     X22, X0
SUBS    W23, W23, #0
CSET    W24, EQ
SUBS    W22, W22, #0xE
CSET    W25, NE
ORR     W24, W24, W25
```

Syscalls

- **Syscalls are directly executed**
  - 400+ syscalls
  - Hooking APIs is not sufficient
  - Not very compliant with the “Apps may only use public APIs” policy…
- **Strings are decrypted on the fly**
  - Integrity checks
  - Impossible to just find and replace blacklisted paths
- **What we would like to do**
  - Intercept all the syscall with Frida
  - Manipulate the arguments
  - Replace the return value

Interception with Frida

Examples are from the doc: https://frida.re/docs/javascript-api/

- Classically used to intercept function arguments or return values

```
Interceptor.attach(Module.getExportByName('libc.so', 'read'), {
  onEnter(args) {
    this.fileDescriptor = args[0].toInt32();
  },
  onLeave(retval) {
    if (retval.toInt32() > 0) {
      /* do something with this.fileDescriptor */
    }
  }
});
```

- Or to completely replace its implementation

```
const openPtr = Module.getExportByName('libc.so', 'open');
const open = new NativeFunction(openPtr, 'int', ['pointer', 'int']);
Interceptor.replace(openPtr, new NativeCallback((pathPtr, flags) => {
  const path = pathPtr.readUtf8String();
  log('Opening "' + path + '"');
  const fd = open(pathPtr, flags);
  log('Got fd: ' + fd);
  return fd;
}, 'int', ['pointer', 'int']));
```

Interception with Frida

- But can also be used to intercept arbitrary instructions

```
let mainModule = Process.enumerateModules()[0];
let instructionAddress = mainModule.base.add(0x1247)
Interceptor.attach(instructionAddress, (args) => {
  console.log(`R0 = ${this.context.r0}`)
});
```

- Useful to dump process state in the middle of a function…
- But not magic nor perfect
  - May have to patch multiple instructions to redirect execution flow
  - May trash registers (an issue is open)

Using breakpoints

- Frida also allows to intercept exceptions!

```
Process.setExceptionHandler(function (exp) {
    console.log(`Exception ${exp.type} @ ${exp.address}`);
    Thread.sleep(1);
    return false;
});
```

- Replace all the syscall with breakpoints
  - Ensure that we only patch one instruction
- Catch the exception to intercept all the syscalls
- Modify the context to emulate them

Patch all the syscalls

```
function replaceSyscall(address, size){
    let count = 0
    let syscallIns = "01 10 00 d4"

    Memory.scanSync(address, size, syscallIns).forEach((match) => {
        let address = match.address;
        if (address.and(3).toInt32() !== 0)
            return;
        count += 1
        Memory.patchCode(address, 4, (address) => {
            let instructionWriter = new Arm64Writer(address);
            instructionWriter.putBrkImm(0);
        });
    });
    console.log(`[+] Found ${count} svc 0x80`);
}
```

The nasty crash…

- **After a few tries we implemented several syscalls**
- **In parallel we found that normal function are also used**
- **Process always crashed just after the checks**
  - Invalid deref, exit(0), objc_msgSend with invalid pointers etc.
  - Easy to find the check
- **But then the process started to crash…**
- **… this time with trashed PC / LR**
  - No easy way to find the underlying test

Stalker

- **Frida has a Dynamic Binary Instrumentation engine**
  - Stalker
- **Can be used to log all the basic blocks executed**
- **Idea**
  - Run the app until the last successfully bypassed check
  - Trace all the basic blocks
  - Wait for the program to crash
- **Make sure to use sync method**
  - Frida loses the buffered messages when the app crashes
- **This quickly gave us the culprit**
  - An API that we weren’t hooking yet

Stalker

```
function trace() {
    let tid = Process.getCurrentThreadId();
    console.warn('[+] attaching stalker on thread '+tid);
    Stalker.follow(tid, {
        events: {call: false, ret: false, exec: false, block: false, compile: true},
        transform(iterator) {
            let instruction = iterator.next();

            const startAddress = instruction.address;
            if ((startAddress.compare(mainModule.base) >= 0) &&
                (startAddress.compare(mainModule.base.add(mainModule.size)) < 0)) {
                function callback (context) {
                    console.log('executing ' + context.pc.sub(mainModule.base));
                }
                iterator.putCallout(callback);
            }
            do {
                iterator.keep();
            } while ((instruction = iterator.next()) !== null);
        }
    });
}
```

Protections

- **Try to find JailBreak files**
  - open, utimes, stat, pathconf, stat64, fopen
  - Both syscalls and functions
- **Try to block/detect debuggers**
  - ptrace(PT_DENY_ATTACH);
- **Check if the parent pid is launchd**
  - getppid() == 1
- **Try to detect if the rootfs is writable**
  - getfsstat64, statvfs

Solution

A generic API

- A generic interface to hook both functions and syscalls

```
}, {
    name: "ptrace",
    syscall: 26,
    hook(arg){
        if (arg == 0x1f) { // PT_DENY_ATTACH
            console.log("[+] ptrace(PT_DENY_ATTACH) -> NOK");
            return {retv: 0};
        }
        console.log("[+] ptrace(???) -> OK");
    }
}, {
    name: "utimes",
    syscall: 138,
    hook(arg){
        let path = arg.readUtf8String()
        if (!iswhite(path)) {
            console.log(`[+] utimes(${path}) -> NOK`);
            return {errno: 2}
        }
        console.log(`[+] utimes(${path}) -> OK`);
    }
}, {
```

A generic API

- Handle special cases

```
name: "open",
syscall: 5,
hook(arg) {
    let path = arg.readUtf8String()
    if (!iswhite(path)) {
        console.log(`[+] open(${path}) -> NOK`);
        return {
            errno: 2,
            onLeave(state) {
                let fd = state.context.x0.toInt32();
                console.log(`fd: ${fd}`);
                if (fd != -1) {
                    console.log(`closing fd ${fd}`);
                    close(fd);
                }
            }
        }
    }
    console.log(`[+] open(${path}) -> OK`);
}
```

Future

Other techniques

- **Try to load an invalid signature**
  - fcntl(F_ADDSIGS);
- **Check if some JailBreak libraries are loaded in your process**
  - /usr/lib/substitute-inserter.dylib for example
  - Can use dlopen / memory scanning / dyld internal structures etc.
- **Check if your process is instrumented**
  - Check code integrity

    CRC, derive constants from the code, check API entries, etc.
  - Time code execution
  - Try to detect Frida
- **Check signature state**
  - Via csops(CS_OPS_MARKKILL)
- **Crash later**
  - Use a global context
  - Put the crash long after the detection
  - Complicate the backtracing


SYNACKTIV
?

Bonus

Future of iOS instrumentation

- **Harder and harder to attack iOS devices**
  - Pointer signature (PAC)
    - Per process and per Team ID keys
    - A lot of kernel data pointers are now signed
  - API hardening
    - Impossible to manipulate a system process even with its task port
  - Sandboxing
    - More and more kernel API are sandboxed
      - ioctl, fcntl, syscalls, necp etc.
    - More and more services are sandboxed
  - Isolation
    - Kernel allocations segregation
- **Apple not only kills bugs but also exploit techniques**
- **JailBreaks are more and more precious**

PPL

- **All the memory management is done in a special CPU state**
  - Impossible to patch the page tables with an arbitrary kernel write
- **PPL also protect userland services**
  - PPL knows all the system services

    Hashes are hardcoded in its data
  - Forbid to inject third party executable code in a system process
- **Could be deployed for all the processes**
  - If they don’t have a special entitlement
- **Still possible to manipulate the process…**
  - With data only manipulation
  - Or by using hardware breakpoints
- **…but not that easy nor handy**
  - Needs to sign pointers with the distant process key
  - Not an infinite number of hardware breakpoint
  - All the tool will have to be recoded