Reversing the Apple Sandbox
OWASP EEE 2015

Razvan Deaconescu
razvan.deaconescu@cs.pub.ro

Recent iOS Attacks

Jekyll Attacks
Celebrity Leaks
XcodeGhost

Apple iOS Defense Mechanisms

Private/public framework separation
Apple Vetting Process
Privacy Settings
Sandboxing
Trusted BSD security layer

Aims

Better understanding of Apple security mechanisms

Improve security

Part of joint research work with TU Darmstadt (CASED) and North Carolina State University

Apple Sandboxing

- Limit attack surface for a given app
- An app is provided a sandbox profile
- Sandbox profiles consist of sandbox profile rules
  - Scheme-like rules
  - SBPL format (Sandbox Profile Language)
  - SBPL format is compiled into binary format
- Little documentation on internals
- Default “container” sandbox profile for 3rd party iOS aps

Reversing Apple Sandbox

- Reverse “container” sandbox profile
- Get an understanding of the rules inside the defaul container
  - Analyze how they could be bypassed or improved
- Make use of very little documentation on the internals
  - No official documentation on SBPL operations
  - No official documentation on the inner workings
  - No official documentation on the binary format

Sample SBPL File

```
[…]

(allow ipc-posix-shm
    (ipc-posix-name "apple.shm.notification_center"))

(allow mach-lookup
        (global-name "com.apple.networkd")
        (global-name "com.apple.NetworkSharing")
        (global-name "com.apple.pfd"))

(allow mach-per-user-lookup)

(system-network)
(allow network* (local ip))

[…]
```

How Sandboxing Works

- SBPL consists of rules (operations and filters)
- Each rule is a deny or allow
- Kernel loads profile for an app
- Hooks inside the kernel check the rules inside the profile and allow or deny acces to the app
- Works similarly for iOS and Mac OS X
- Implemented in the sandbox kernel extension (Sandbox.kext)

Creating an Apple Sandbox Profile

- Write an SBPL file
- Use sandbox-exec command or sandbox_init() function load an app using given profile
- Use sandbox_compile() to compile a binary format
- The binary format is used by the app
- sandbox_* functions are fairly undocumented and used internally
  - Implementation in libsandbox.dylib

Anatomy of the Apple Sandbox Profile

- Each rule consists of an operation, filter and action
- Operation is a class of action (file-read*, network-inbound, process-exec)
- Filter is an argument to the operation (file name, socket address, process ID)
  - Filters may be regular expressions
- Action may be allow or deny
  - Flags may be part of it (such as debug)

Need to Know

- What is inside an .sb file?
- Where are the builtin binary sandbox profiles stored?
- What is the format of the binary sandbox profile file?
- How can one reverse the format?

Previous Work

- Dionysus Blazakis (Dion)
  - The Apple Sandbox (BlackHat 2011)
  - 5th Chapter in “The iOS Hacker's Handbook”
  - https://github.com/dionthegod/XNUSandbox/
- Stefan Esser (Stefan)
  - “iOS8 Containers, Sandboxes and Entitlements” (Ruxcon 2014)
  - https://github.com/sektioneins/sandbox_toolkit

Methodology Overview

- Get complete list of operations and filters
- Get a good understanding of the sandbox workflow (create/compile, apply)
- Extract builtin binary sandbox profiles
- Thorough understanding of the binary format
- Reverse a binary format sandbox profile file to its initial SBPL format

Building Blocks

- Compile SBPL format file to binary format
- Use sandbox profile
- The intermediary “even more Scheme-like” format
- Well documented by Dion, though one needs multiple read throughs to have a good picture

Full List of Filters and Operations

- List of operations provided by Dion and Stefan
- Methodology: look into Sandbox.kext
  - Updated methodology: extract strings from libsandbox.dylib and look for “%operations”
- No methodology for filters in previous work
  - As with operations, use strings in libsandbox.dylib

Intermediary Format

Show samples

Intermediary Format

- Slightly updated TinyScheme interpreter inside libsandbox.dylib
- SBLP → Intermediary Format → Binary Format
- By “hooking” into the interpreter one can dump the intermediary format

```
$ cat osx_sbpl_stub.scm osx_sbpl_init.scm
osx_sbpl_v1.scm require-in-require-allow-deny.sb
display_rules.scm | ./as
```

Extract Builtin Binary Sandbox Profiles

- Located in the sandboxd executable file
- Start from the profile string (i.e. “container”)
- Do “offset-based computing” and locate start of binary profile and region length
- Nice implementation by Stefan
  - https://github.com/sektioneins/sandbox_toolkit/tree/master/extract_sbprofiles
  - Stefan's implementation wasn't available at the time I started this :-(

The Apple Sandbox Binary Format

- Initial work by Dion (for iOS v5)
- Updated work by Stefan (for iOS v8)
  - All work by Dion
  - Insight on regular expressions format and the operations list
- Methodology: create SBPL format files, compile and check

Binary Format Header

- Header version (2 bytes)
- Offset to regular expression section (2 bytes)
- Number of regular expressions (2 bytes)
- Table of offsets (NUM_OPERATIONS * 2 bytes)
  - Offset to action nodes for each operation
- All offsets multipied by 8

Sample Regular Expression File

```
(version 1)
(allow default)
(deny file-read-data
  (regex #"^/[ab]$")
  (regex #"^/(a)?bc$")
  (regex #"^/(ab)?cd$")
  (regex #"^/(ab|cd)$")
  (regex #"^/.a$"))
```

```
00000210: 4300 4f00 5d00 6f00 5a00 0000 0000 0003  C.O.].o.Z.......
00000220: 5400 2f49 002f 3a00 2f29 002f 1500 1902  T./I./:./)./....
00000230: 2f09 0261 2915 0019 022f 2f22 0002 6102  /..a)....//"..a.
00000240: 6229 1500 0263 0264 0a1f 0019 022f 2f33  b)...c.d.....//3
00000250: 0002 6102 6202 6302 6429 1500 1902 2f2f  ..a.b.c.d)....//
00000260: 4200 0261 0262 0263 2915 0019 022f 2b61  B..a.b.c)..../+a
00000270: 6162 6229 1500
```

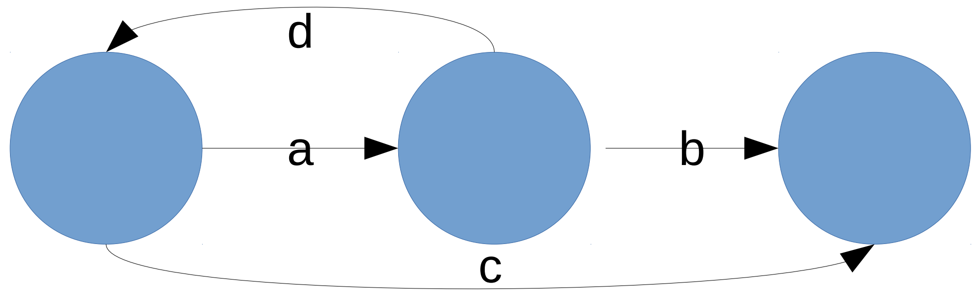
Liniarized Regular Expression

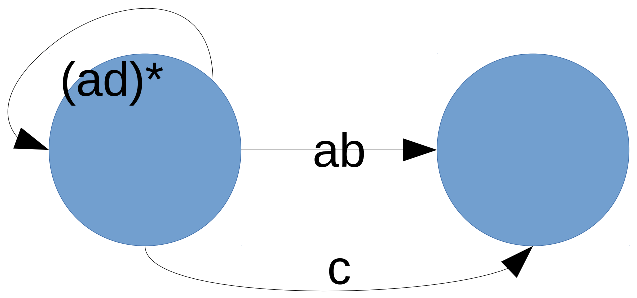
- regex → NFA (Non-deterministic Finite Automaton)
- NFA is “binarized”
- Representation for: characters, special characters (., ^, $), character sets, jumps
- Documented by Stefan (though some parts are missing)
  - Dion had done it, but encoding is different (as noticed by Stefan)

Regex Reversing Steps

- Create NFA from binary representation as a graph
  - Intermediary representation where vertice is a character and edges are possible “links”
- Use state removal algorithm
  - Leave initial and final states for last
  - Take care of * and + regex operand
  - Take care of ? Operand
  - Take care of complex expressions using (and)

Idea for State Removal Algorithm





TODOs for Regex Reversing

- Robust reversing when operation uses multiple regular expressions
  - They are part of a single binary representation but need to “split” them apart
- Remove builtin regular expressions in binary format
  - Sandbox compiler by default adds certain regular expressions to deny access to certain services irrespective of the initial file

Reminder: Binary Format Header

- Header version (2 bytes)
- Offset to regular expression section (2 bytes)
- Number of regular expressions (2 bytes)
- Table of offsets (NUM_OPERATIONS * 2 bytes)
  - Offset to action nodes for each operation
- All offsets multipied by 8

Operation Offsets

- Each operation gets and offset to an action node
  - There will always be at least one offset per operation
- Two types of action nodes (dubbed “operation nodes” by Dion and Stefan)
  - Terminal nodes: allow or deny
    - Dubbed result nodes by Stefan
  - Non-terminal nodes: do further processing
    - Dubbed decision nodes by Stefan

Terminal Action Nodes

- Padding (1 byte)
- Action (deny/allow) (2 bytes)
  - Flags: debug

Non-Terminal Action Nodes

- Filter type (1 byte)
- Filter argument (2 bytes)
- In case of match, offset to next action node (2 bytes)
- In case of unmatch, offset to next action node (2 bytes)

Reversing Filters

- Not fully done/documented by Stefan
- Extract all filters
- Create SBPL file with all of them and compile
  - Match filter IDs and filter arguments to actual filters

Match/Unmatch Options in Action Nodes

- Match is terminal, unmatch terminal
  - Current operation filter is denied/allowed
  - Terminate processing of operation
- Match is non-terminal, unmatch is terminal
  - Link current action to previous action
- Match is terminal, unmatch is non-terminal
  - Current operation filter is denied/allowed
  - If no match, link unmatch action to previous action
- Match is non-terminal, unmatch is non-terminal
  - “Split” in decision making, link both current and unmatch action to previous action

require-all/require-any

```
(version 1)
(deny default)
(allow file-read*
       (require-all (file-mode #o0004)
                    (require-any (require-all (literal "/etc")
                                              (require-any (regex #"/a.*$")
                                                           (vnode-type REGULAR-FILE)))
                                 (subpath "/Library/Filesystems/NetFSPlugins")
                                 (subpath "/System")
                                 (subpath "/private/var/db/dyld")
                                 (subpath "/usr/lib")
                                 (subpath "/usr/share"))))
```

```
0: (1e) non-terminal: (0e 0001 002a 0029)
1: (1f) non-terminal: (04 0004 0020 0029)
2: (20) non-terminal: (01 0047 002a 0021)
3: (21) non-terminal: (01 0043 002a 0022)
4: (22) non-terminal: (01 0041 002a 0023)
5: (23) non-terminal: (01 003c 002a 0024)
6: (24) non-terminal: (01 003a 0025 0027)
7: (25) non-terminal: (81 0001 002a 0026)
8: (26) non-terminal: (1d 0001 002a 0027)
9: (27) non-terminal: (01 0034 002a 0029)
10: (28) non-terminal: (81 0000 0029 002a)
11: (29) terminal: deny
12: (2a) terminal: allow
```

TODOs for Reversing Action Nodes

- Handle require-not
- Remove default action nodes rules
  - Operations not in initial SBPL file use implicit rules (deny, allow and others)
  - These rules need not be present in the reversed SBPL file
- Handle terminal flags (debug)

Current State of Things

- Draft reverse of builtin iOS “container” sandbox profile
  - See demo
- Scripts to do small little things
  - README and instructions for advanced user
- Need to make scripts more generic and usable
- Research paper under way
- Will most likely publish tools as open source

Lessons Learnt

- Reversing is fun and time consuming
- Previous work has been very helpful
  - Though I only figured some things out later
- Graphs are really useful IRL!
- You'll never know what you need to know when doing reversing: graphs, NFAs, regex, algorithms, functional programming