++

Needle – User Guide

Black Hat Arsenal USA 2016

MWR INFOSECURITY

Contents page

1. Introduction

Assessing the security of an iOS application typically requires a plethora of tools, each developed for a specific need and all with different modes of operation and syntax. The Android ecosystem has tools like "drozer" that have solved this problem and aim to be a ‘one stop shop’ for the majority of use cases, however iOS does not have an equivalent.

"Needle" is an open source modular framework which aims to streamline the entire process of conducting security assessments of iOS applications, and acts as a central point from which to do so. Given its modular approach, Needle is easily extensible and new modules can be added in the form of python scripts. Needle is intended to be useful not only for security professionals, but also for developers looking to secure their code. A few examples of testing areas covered by Needle include: data storage, inter-process communication, network communications, static code analysis, hooking and binary protections. The only requirement in order to run Needle effectively is a jailbroken device.

2. Installation

2.1.1 Setup the Device

The only prerequisite is a Jailbroken device, with the following packages installed:

- Cydia
- OpenSSH
- Apt 0.7 Strict

2.1.2 Setup Kali

Get a copy of Needle

```
git clone https://github.com/mwrlabs/needle
```

Install Prerequisites

```
# Debian dependencies
cat requirements-debian.txt | xargs apt-get install -y
# Python dependencies
pip install -r requirements.txt
```

N.B. Needle has only been tested on Kali 2.0.

3. Usage

3.1.1 Standard

```
python needle.py
```

3.1.2 Automated, using a resource file

First, create a resource file with the commands you want to have automatically executed. For example:

```
$ cat config.txt
# This is a comment, it won't be executed
set DEBUG False
set VERBOSE False
# If SETUP_DEVICE is set to True,
# Needle will automatically install all the required tools on the device
set SETUP_DEVICE False

set IP 192.168.0.10
set PORT 5555
set APP com.example.app
use binary/metadata
```

Then, launch Needle and instruct it to load the resource file:

```
python needle.py -r config.txt
```

4. Features

Area	What	Command	Description
[CORE]	CLI interface	`python needle.py`	
[CORE]	Do Resource	`python -r <path to file>`	Executes commands from a resource file
[CORE]	Session manager		SSH, USB over SSH
[CORE]	Device auto-configuration	`set SETUP_DEVICE True`	On launch, Needle checks if all the tools needed are already on the device, otherwise it will install them

[CORE]	Modular approach	`show modules`
		`use <module_name>`
		`show [options | source | info | globals]`
			Show details of a particular module, once selected

[CORE]	Background jobs	`jobs` `kill <num>`	List running jobs and kill them
[CORE]	Search	`search <query>`	Search available modules
[CORE]	Local command	`<cmd>`	Execute a command on the local workstation
[CORE]	Drop shell	`shell`	Drop a shell on the remote device
[CORE]	Do command	`exec_command <cmd>`	Execute a single command on the remote device
[CORE]	Push/pull	`<push|pull> <src> <dst>`	Push/pull files on the device
[BINARY]	Class Dump	`use binary/class_dump`	Dump the class interfaces
[BINARY]	Compilation Checks	`use binary/compilation_checks`	Check for protections (PIE, ARC, stack canaries, binary encryption)
[BINARY]	Install IPA	`use binary/install`	Automatically upload and install an IPA on the device
[BINARY]	App Metadata	`use binary/metadata`	Display the app's metadata (UUID, app name/version, bundle name/id, bundle/data/binary directory, binary path/name, entitlements, url handlers, architectures, platform/sdk/os version)
[BINARY]	Pull IPA	`use binary/pull_ipa`	Decrypt and pull the application's IPA from the device
[BINARY]	Shared Libraries	`use binary/shared_libraries`	List the shared libraries used by the application
[BINARY]	Strings	`use binary/strings`	Find strings in the (decrypted) application binary, then try to extract URIs and ViewControllers

[COMMS]	Delete Installed Certificates	`use comms/certs/delete_ca`	Delete one (or more) certificates installed on device
[COMMS]	Export Installed Certificates	`use comms/certs/export_ca`	Export one (or more) certificates installed on device
[COMMS]	Import Installed Certificates	`use comms/certs/import_ca`	Import a certificate from a file in PEM format
[COMMS]	Install MitmProxy CA Certificate	`use comms/certs/install_ca_mitm`	Install the CA Certificate of MitmProxy on the device
[COMMS]	List Installed Certificates	`use comms/certs/list_ca`	List the certificates installed on device
[COMMS]	Intercepting Proxy	`use comms/proxy/proxy_regular`	Intercept the traffic generated by the device
[DYNAMIC]	Jailbreak Detection	`use dynamic/detection/jailbreak_detection`	Verify that the app cannot be run on a jailbroken device
[DYNAMIC]	URI Handler	`use dynamic/ipc/open_uri`	Test IPC attacks by launching URI Handlers
[DYNAMIC]	Heap Dump	`use dynamic/memory/heap_dump`	Dump memory regions of the app and look for strings
[DYNAMIC]	Monitor File changes	`use dynamic/monitor/files`	Monitor the app data folder and keep track of modified files
[DYNAMIC]	Monitor OS Pasteboard	`use dynamic/monitor/pasteboard`	Monitor the OS Pasteboard and dump its content
[DYNAMIC]	Syslog Monitor	`use dynamic/monitor/syslog`	Monitor the syslog in background and dump its content
[DYNAMIC]	Syslog Watch	`use dynamic/watch/syslog`	Watch the syslog in realtime
[HOOKING]	Cycript shell	`use hooking/cycript/cycript_shell`	Spawn a Cycript shell attached to the target app
[HOOKING]	Frida launcher	`use hooking/frida/frida_launcher`	Run Frida scripts (JS payloads)
[HOOKING]	Frida shell	`use hooking/frida/frida_shell`	Spawn a Frida shell attached to the target app
[HOOKING]	Frida trace	`use hooking/frida/frida_trace`	Trace the specified functions using frida-trace
[HOOKING]	Enumerate All Methods	`use hooking/frida/script_enum-all-methods`	Enumerate all methods from all classes in the application
[HOOKING]	Enumerate Classes	`use hooking/frida/script_enum-classes`	Enumerate available classes
[HOOKING]	Enumerate Methods	`use hooking/frida/script_find-class-enum-methods`	Find the target class specified and enumerate its methods
[STATIC]	Code Checks	`use static/code_checks`	Static analysis of the apps's source code. Aims to find usage of potentially insecure functions. Can be applied to a whole folder or, if

			SECONDARY_FOLDER is specified, only to the diffs computed among the 2 versions of the same codebase.
[STORAGE]	Screenshot Caching	`use storage/caching/screenshot`	Test if a screenshot of the application's main window is cached when the application's process is moved to the background
[STORAGE]	Binary Cookies Files	`use storage/data/files_binarycookies`	List Binary Cookies files contained in the app folders, alongside with their Data Protection Class. Plus, offers the chance to pull and inspect them with *BinaryCookieReader*
[STORAGE]	Cache.db Files	`use storage/data/files_cachedb`	List Cache.db files contained in the app folders, alongside with their Data Protection Class. Plus, offers the chance to pull and inspect them with *SQLite3*
[STORAGE]	Plist Files	`use storage/data/files_plist`	List plist files contained in the app folders, alongside with their Data Protection Class. Plus, offers the chance to inspect them with *Plutil*
[STORAGE]	SQL Files	`use storage/data/files_sql`	List SQL files contained in the app folders, alongside with their Data Protection Class. Plus, offers the chance to pull and inspect them with *SQLite3*
[STORAGE]	Dump Keychain	`use storage/data/keychain_dump`	Dump the keychain
[VARIOUS]	Clean Storage	`use various/clean_storage`	Clean device storage from leftovers artefacts of other tools (e.g., Frida)