Author: S. Voigtlander

Date: September, 2018

Location: Eindhoven

Table of Contents

# Concept

UKERN is an acronym for user land kernel.

It aims to provide a high-level layer that operates just like a kernel, but running from within a sandboxed app container.

UKERN mainly focusses on getting one programmable, accessible and fairly open modular interface inside a sandboxed system. It is designed mainly for developers and researchers of the iOS Operating System and users who are missing administrative features on their devices.

UKERN’s strength is of that small solutions can be written such as extensions to the system for server-side capabilities or RFC protocols. UKERN does not follow the design guidelines of Apple Inc. but never was intended to. It opens up for developers to code binaries in the native language of the iOS operating system: C, C++ and Objective-C. Unlike other existing open environments that are created for iOS, it does not depend on specific system features or vulnerabilities and exploits. It fully integrates with iOS’ ecosystem and the restrictions of the sandbox still apply. To simulate a Unix-like system and file structure, kernel like features are written to simulate the authentication of users and process creation. In UKERN a highly insecure but extremely useful feature is integrated for executing arbitrary c functions, interpreted by name and arguments.

This makes UKERN unique in the way that it can use all loaded library functions in a user controllable environment.

UKERN by design is highly insecure but because it is designed for the highly restricted sandboxed environment no vulnerabilities exist of where private user data from the iOS System exists.

UKERN uses low latency system API’s to get the best experience and performance, therefore UKERN is currently limited only to the iOS Sandbox and the iOS System but in the future support for other sandboxed environments can be added.

As UKERN’s core principal is to bring more freedom to sandboxed environments UKERN should never be used for financial gain and therefore is provided free and open source under the MIT license.

# UKern design internals

UKERN is written in ANSI C, C++ and Objective-C.

It makes use of under more Capstone, liblorgnette and the OpenSSL library that provide basic analytic, cryptographic and logic functions that UKERN depends on.

Liblorgnette, a library for address space and library related analysis is at this time is designed for mac and iOS but a replacement could be written for Windows and other platforms as well.

## The loader

UKERN uses the MACH-O format and the dynamic library loader to load custom extensions into the runtime at load, this not only opens up for developers to code their own extensions to the app but also forms a great way for hijacking system library calls and debugging them. It is up to the developer of these extensions to choose what category the extension should be in.

In the initialization stage UKERN defines where to log standard output to, this by default is a file that is read into a buffer upon the detection of a write event to the file system.

UKERN chose this way because it makes it easier for software engineers to debug their processes, for security this might be rather insecure because software may disclose user data in the output logs.

However, no person will be able to read out this information as this is stored in the secure file storage of the sandbox.

Loading extensions works by mapping the files into the address space at runtime, but some requirements come with it because of Apple’s code signing enforcement.

A breakout for code signing is being worked on that will use Return Oriented Programming and a custom Mach-O loader to execute arbitrary code in the address space, however, to Apple Inc. this is considered a creditable vulnerability eligible for bounty. I do not have plans to claim this bounty but understand that this code signing requirement needs to be there and that my application should not depend on bypasses for it.

Therefore I came up with the following problem:

* Developer License holders of Apple are able to sign and distribute libraries and apps through websites using the iTunes services protocol to install them
* SSL encryption on the host is required for distribution
* An application from the same developer with the same bundle ID as an installed Application from the same developer is updated rather than installed over, which leaves the previous user data there.
* You cannot launch code signed libraries from within the documents directory or temporary directory of the sandbox, the libraries need to be in the same directory as the App container which is marked as non-writable after installation.

The solution for this is as following:

* License holders of Apple Inc. can install their exported signing certificate bundle and private key into the UKERN application.
* UKERN integrates an Webserver with SSL support and the user is asked and required to install a self-signed SSL certificate, this is to match the requirement of having an SSL encryption for installing code signed web distributed apps
* UKERN contains a writeable copy of itself in the documents directory of the app, this copy is downloaded directly from Github’s releases so that always the latest version will be used. The application will be (gpg, **NOT** codesigned) signed by me, [s.voigtlander@jailed.ml](mailto:s.voigtlander@jailed.ml) , to verify the integrity of the download.
* New modules can be downloaded into the documents directory of UKERN’s app bundle.
* A new build of UKERN with the same bundle ID is made with the dynamic libraries signed into the new application that is in the documents directory. This is distributed over the local webserver with ssl and UKERN will request the user to install the new application.
* The new UKERN application is installed over the old UKERN application and the user now has UKERN with the old user data still intact and the new modules installed.

# The logic

## Process simulation

Because the sandbox in iOS does not allow the execution or even mapping of executable files for execution, forking or even virtual forks, UKERN works with thread based process separation. To simulate an actual process environment, logic is written for simulating the real kernel’s logic of process creation.

A system call is used to read out the identifier of a thread and this automatically managed number is then used as the process ID.

Process structures have been simulated as well making UKERN support processes running with different simulated privileges and even sub processes or forked processes are possible.

Killing processes works by getting the thread id of a process by ID or name, which is done via a lookup in the global process listing table and then getting a reference to the thread by thread id, killing the thread with an implementation of pthread\_kill.

When time and support are, the simulation of MACH task ports will also be written so that processes can manage each other’s fake stack / address space. Because of the layout of a task, this will work by still using the real process its task but locking the functionality limited to only the memory belonging to a certain thread.

UKERN supports getting the processor register values for threads by using an implementation of mach\_thread\_self and mach functionality for getting thread states.

This is great for debugging execution between fake (simulated) processes.

## Privilege simulation

Just like in any kernel privileges should be separated, but this not fully possible due to sandbox limitations, yet the simulation of it is possible and therefore makes a more comfortable, but **insecure** environment where users indeed need to enter there password for certain operations such as killing processes running under the fake “root” user, but through memory no separation is possible due to the design of a processor and the limitation of working in the same process it’s address space inside of the sandbox.

The password of the user is stored in a hashing database, /etc/passwd on the simulated file system structure.

The hashing algorithm used is SHA512 and the implementations of the algorithm are provided by the OpenSSL foundation in libssl for iOS.

This hash is loaded into a structure upon initiating a login session request, in theory this makes the application vulnerable to timing attacks as well for enumerating the users if from a remote server-side authentication perspective such as via the SSH protocol, yet this is not seen as a major issue.

## Code Execution

Code Execution is done using an algorithm written on top of liblorgnette.

A symbol is lookup in the address space of the real process when a request to run a binary is done, if it is not in the process space, UKERN will try to load a dynamic library with that name from the extensions path, if no dylib is found with the symbol UKERN will return that the request to execute has failed.

Otherwise the address (binaryname)\_main entry is looked up and a function pointer with arbitrary arguments is set to that address, the parsed arguments are stored in an argv[] buffer and passed to the function and then the function is executed.

The addition of \_main to the “command” (in total it makes: command\_main) ensures that no arbitrary function can be called.

UKERN has its own improved implementations of strcat for merging strings, based on an Objective-C bridge.

Binaries are spawned as the user they are ran with, there is no privilege protection written for reading / writing to files yet but in the future this can be done when the virtual file system of UKERN reaches its final development state.

## Simulated (Virtual) File System

## Network protocols and services

## Launch daemons