SAND2017-7574C

# The Center for Cyber Defenders

Expanding computer security knowledge

# Project Montana

Eric Buedel, Purdue University; Mukhil Murugasamy, University of Illinois Urbana-Champaign;
Ryan Jacobson, University of California, Santa Cruz; Khiem Tang, University of Texas at Austin

**Project Mentors: Jonathan Cooke, Org. 5836; Nolan Van Foeken, Org.5834**

## Problem

Binary fuzzing using AFL is 2-5x slower than fuzzing when source code is available.

## Objective

Improve the speed of fuzzing x86 and ARM binaries using AFL.

## Approach

When source code is available, AFL-gcc adds instrumentation to the code before compiling. We want to mimic this by injecting instrumentation code into the compiled binary. Need to disassemble binary, add instrumentation, then reassemble.

We have explored these tools so far:

- AFL-Dyninst – Uses Dyninst binary instrumentation library to produce binaries compatible with AFL fuzzing. Does not work with ARM binaries.
- Unicorn – Built off of QEMU for emulation of x86 and ARM architectures for 32 and 64-bit
- Angr – Lifts binaries to VEX IR for analysis. Allows functions to be hooked but does not support saving an instrumented form of the binary to disk for use outside of Angr.
- BAP – Supports binary lifting to a custom IR. Disassembly output cannot be easily reassembled.
- Patcherex – Uses Angr to support disassembly of x86 and ARM binaries. Developed for use on special binaries, so it does not always produce recompilable assembly.
- DynamoRIO – Runtime x86 and ARM binary manipulation tool. Still investigating this tool.

Tools such as IDA and Objdump can disassemble x86 and ARM binaries but not in a format that allows for reassembly. We did not research the use of such tools.

## Impact and Benefits

AFL is an industry standard for fuzzing testing. By exploring binary instrumentation we can use AFL without the source code and potentially improve the speed of fuzzing.

## Problem

Identifying malicious applications on mobile devices is difficult due to the relative lack of user control over app behavior, and lack of standardized verification of app developer ethics and design quality.

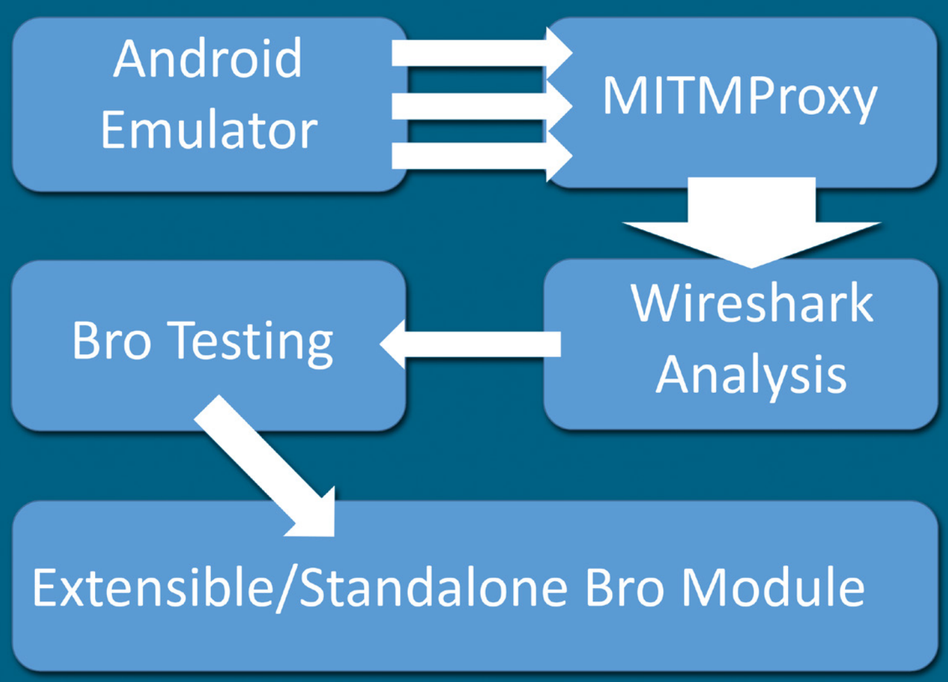
## Objective

Build an extendible Bro IDS-based traffic analyzer to categorize and attribute all traffic from an Android device, including protocols, endpoints, packet headers, etc. Provide a means to identify nefarious apps by confirming the network traffic falls within the scope advertised by the app's features.

## Approach

Environment and tools in the analysis phase:

- Android Open Source Project Emulator for a flexible Android environment from which to generate traffic.
- MITMProxy – "Man In The Middle" Proxy – to *collect, isolate,* and *decrypt* datastream from the Android Emulator.
- Wireshark to investigate traffic for behavior patterns and identifying signatures.




In the coming weeks we will write a Bro module to create logs, summaries and alerts for traffic coming from specific devices and apps. Our final goal is to build a sample profile written in Bro's scripting language of an app that can identify whether its network behavior is consistent with its stated features.

## Impact and Benefits

This framework can be extended to more apps as profiles are created for them, which will enable administrators with networked Android devices to better protect their infrastructure from mobile threats.

Sandia National Laboratories is a multimission laboratory managed and operated by National Technology and Engineering Solutions of Sandia, LLC., a wholly owned subsidiary of Honeywell International, Inc., for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-NA0003525. SAND2017-XXXXC.