CustomPro Initial Evaluation
PJR Corp
Chris Shiflet and Froilan Olivas

23 December 2020

Overview

CustomPro allows for binary customization based on network packet tainting

- Binary is run in emulated environment to allow recording of relevant features
- Network packets serve as the data source to track what feature/behavior is exercised
- Leverages guided symbolic execution (GSE) and binary rewriting technologies to produce new binary
- Research from George Washington University
  - Professors/PI – Tian Lan, Guru Venkataramani
  - Students – Kailash Gogineni, Yongsheng Mei, Yurong Chen

Initial Evaluation

Impressions

- CustomPro readily produces feature trace information
- Taking tainted trace output and producing new binary requires proficiency with Angr and DynInst software tools
- Tool relies on somewhat outdated platform/tooling (TEMU)
  - TEMU v1.0 (based on QEMU v0.9.1) ~10 year old versions
- Using recent binaries on tool may be a challenge as result of very outdated emulation tool

Initial Evaluation

Environment Setup:

- **Time Spent:** 1/2 day
- Tainting VM
  - Contains TEMU VM – Dynamic binary analysis platform (based on QEMU)
  - OS: Ubuntu 12.04.5 32-bit
  - Guest OS (TEMU): Ubuntu 9.04 32-bit
- Guided Symbolic Execution VM
  - Provides Angr which performs symbolic execution on tainted output
  - OS: Ubuntu 20.04 64-bit
- Binary rewriting VM
  - Contains DynInst instance, used to generate new specialized binary
  - OS: Ubuntu 16.04 32-bit

Initial Evaluation (Cont.)

Using CustomPro:

- **Time Spent: 4 days**
- Target binary must either be compiled in guest OS or matching system so it runs properly on guest OS (TEMU)
- Once target binary begins receiving tainted packets, feature recording begins
- Packet tainting supported:
  - Whole packet tainting (with option to taint on specific packet type via packet "pattern")
  - Partial packet tainting (with two additional configuration fields to specify which bytes within a packet to use for instruction tainting)

Initial Evaluation (Cont.)

Using CustomPro: (cont.)

Guided Symbolic Execution:

- Identifies additional code blocks related to desired feature
  - Helps guard against erroneously removing needed code
- User should be familiar with Angr APIs, binary disassembly, and assembly address fields
- High level process
  - Provided Jupyter notebook works on tainted instructions to:
    - Extract instructions based on given range (Seed for symbolic execution)
    - Convert assembly instructions from virtualized address space to objdump address space
  - Python/Angr application will perform symbolic execution on binary using tainted instructions and produce basic block list to be used in binary rewriting

Initial Evaluation (Cont.)

Using CustomPro: (cont.)

- Binary Rewriting:
  - Uses identified code blocks from Angr GSE and generates new binary with desired changes
  - User needs to be familiar with DynInst API (C++)
  - "Mutator" program needs to be written to produce new binary
    - Mutator uses DynInst API (C++) to modify the application (Mutatee)
    - Sample Mutator provided by researcher demonstrates replacing non-tainted basic blocks with “NOP” assembly instructions.
  - New binary is generated from successful execution
    - Functionality not marked during packet tainting not present in new binary
    - MD5 functionality successfully removed with “NOP” method
      - » Stubs could be added for better handling
    - Attempted to run binary on original environment, however new binary has dependency on libdyninstAPI_RT.so lib. Static linking may address this.

Estimation of Work

- Symbolic execution workflow improvement – **3-6 Months**
  - Create in-depth documentation to describe process
  - Automate handling of CustomPro input in Python/Angr application
    - Sample environment provided by research team a good start, but requires fair amount of user interaction and manual steps
- Binary rewriting workflow improvement – **3-6 Months**
  - Create in-depth documentation to outline binary re-writing
  - Write a robust, extensible mutator application
    - Sample mutator application provided by research team is a good start but also requires fair amount of user interaction to modify/recompile Mutator
- Tool Updates – **6-12 Months**
  - TEMU has had little to no support since 2009
  - Requires gcc 3.4 specifically
  - Community attempts to install in latest OS have been met with suggestions to use alternative tool, DECAF
  - Migration to DECAF could provided better long-term support

Estimation of Work

- Reduce number of VMs required – **0.5-1 Month**
  - May involve VM + Container solutions
  - Need to maintain support 32-bit and 64-bit tools (Angr/DynInst)