A

The Incredible Mutating Test Files! Smart Mutation for Binary Fuzzing

Clark Wood, Florida State University

Categories and Subject Descriptors: CNT 5605 [**Fall 2013**]: Dynamic Taint Analysis, Mutation, Fuzzing

ACM Reference Format:
Clark Wood, Paper CNT 5605 V, N, Article A (January YYYY), 2 pages.
DOI: http://dx.doi.org/10.1145/0000000.0000000

1. FUZZING OVERVIEW

white/black/graybox, mutation/generation, Dynamic Taint Analysis,

As programs grow larger, so do the test suites designed to exercise execution paths within the program. White box approaches have the advantage of source code, allowing developers to write test cases which exercise paths in a higher level language. With black box approaches, however, test cases would typically target assembly block-level code paths [A,...NEEDED].

1.1. Dynamic Taint Analysis

Dynamic taint analysis (DTA) is the process of locating sources of taint and following their propagation through a binary to data sinks. Since, for exploitation, bugs must be triggered by user input, DTA usually identifies all user input (files, command line arguments, UI interactions, ...) as tainted, and marks it accordingly.

The granularity of taint marking is variable. Assigning everything from a generic binary value of tainted/not tainted to bit-level marking of which part of user input affected which variable is conceivable, although finer-grained taint analysis is more computationally expensive and difficult to determine [CITE]. Consider the following pseudocode:

```
x = userInput & 1
if x % 2 == 0:
vuln()
```

TODO Taint can propagate via explicit or implicit flows (Figure 1), sometimes also called data and control dependencies respectively [G]. Explicit flows involve a tainted variable, x, which is used in an assignment expression to compute a new variable, y. In this situation, x taints y, and if y is involved in any further assignment to a variable z, then x taints z by transitivity. In contrast, implicit data flows involve a tainted variable used to affect control flow within a program which subsequently sets the value of another variable, for instance at a branch [D].

Because of subtle implicit flow cases referenced by Clause, J. et all [D], which can be difficult to spot (Page 2, Figure 2b). Implicit data flows are not always considered by dynamic tainting techniques [E].

This is a problem BECAUSE TODO

```
# Explicit taint

# Implicit taint
```

Propagation

2. MACHINE LEARNING

While research has applied machine learning techniques to successfully extrapolate vulnerability patterns, previous efforts have required source code and training examples containing known vulnerabilities to drive learning [F]. In contrast, we implement ...

3. DYNAMIC BINARY INSTRUMENTATION WITH INTEL PIN

Pintools

4. WAYS TO MUTATE TEST CASES

Machine learning in lieu of symbolic execution?

5. IMPLEMENTATION

5.1. Dynamic Taint Analysis

We implemented dynamic taint analysis for multiple input sources. When performing certain actions, like reading a file, which makes use of the read() syscall in Linux, it was possible to watch for all reads after an specific opens which suggested user involvement, and follow the buffers read in. For user input directed to STDIN or through command line arguments however, it was necessary to hook into c library functions which allowed user input, like strcpy.

5.2. Pintools for Finding Vulnerable Conditions

5.3. Mutation Algorithm

Fuzzing optimization techniques are evolving to deal with the path explosion problem [CITES] in various ways, focusing on ...

Researchers at VUPEN Security mention the value of discovering test case sets of equivalent coverage, but smaller size, via test suite reduction algorithms [A].

6. FUTURE WORK

7. REFERENCES

[A] Bekrar, Sofia, et al. "A taint based approach for smart fuzzing." Software Testing, Verification and Validation (ICST), 2012 IEEE Fifth International Conference on. IEEE, 2012. [B] DeMott, J., Enbody, R., and Punch, W. "Revolutionizing the Field of Grey-box Attack Surface Testing with Evolutionary Fuzzing", BlackHat and Defcon 2007. [C] Schwartz, E.J., Avgerinos, T., Brumley, D. "All You Ever Wanted to Know About Dynamic Taint Analysis and Forward Symbolic Execution (but might have been afraid to ask)." 2010 IEEE Symposium on Security and Privacy. [D] Clause, J. Li, W., Orso, A. "Dytan: a generic dynamic taint analysis framework". 2007 Int'l symposium on Software testing and analysis. ACM, 2007. [E] ... "Beyond Instruction Level Taint Propagation". [F] Yamaguchi, Fabian, Felix Lindner, and Konrad Rieck. "Vulnerability extrapolation: assisted discovery of vulnerabilities using machine learning." Proceedings of the 5th USENIX conference on Offensive technologies. USENIX Association, 2011. [G] Bao, Tao, et al. "Strict control dependence and its effect on dynamic information flow analyses." Proceedings of the 19th international symposium on Software testing and analysis. ACM, 2010.