Mitigating Algorithmic Complexity Attacks

Kai Walberg, Macalester College
Christophe Hauser, USC Information Sciences Institute

Problem Statement

Problem:

- Algorithmic complexity vulnerabilities provide a vector for low-rate DoS attacks
- Vulnerability is inherent to several widely used algorithms and data structures

Proposed Solution:

- We developed a model to detect these vulnerabilities in executables using binary analysis
- Our solution focuses specifically on vulnerable usage of the quicksort algorithm

Background: Algorithmic Complexity Attacks

- Some algorithms are fast in general, but slow in the worst case
  - ex: quicksort ($O(n \log n)$ vs $O(n^2)$)
- Exploitable with crafted input
- Low-rate traffic can trigger denial of service
- Vulnerable algorithms include quicksort, hash tables, regular expression parsers

Vulnerability Detection

Vulnerability characteristics:

1. Vulnerable function used in executable
2. Function handles user input
3. User input is unfiltered

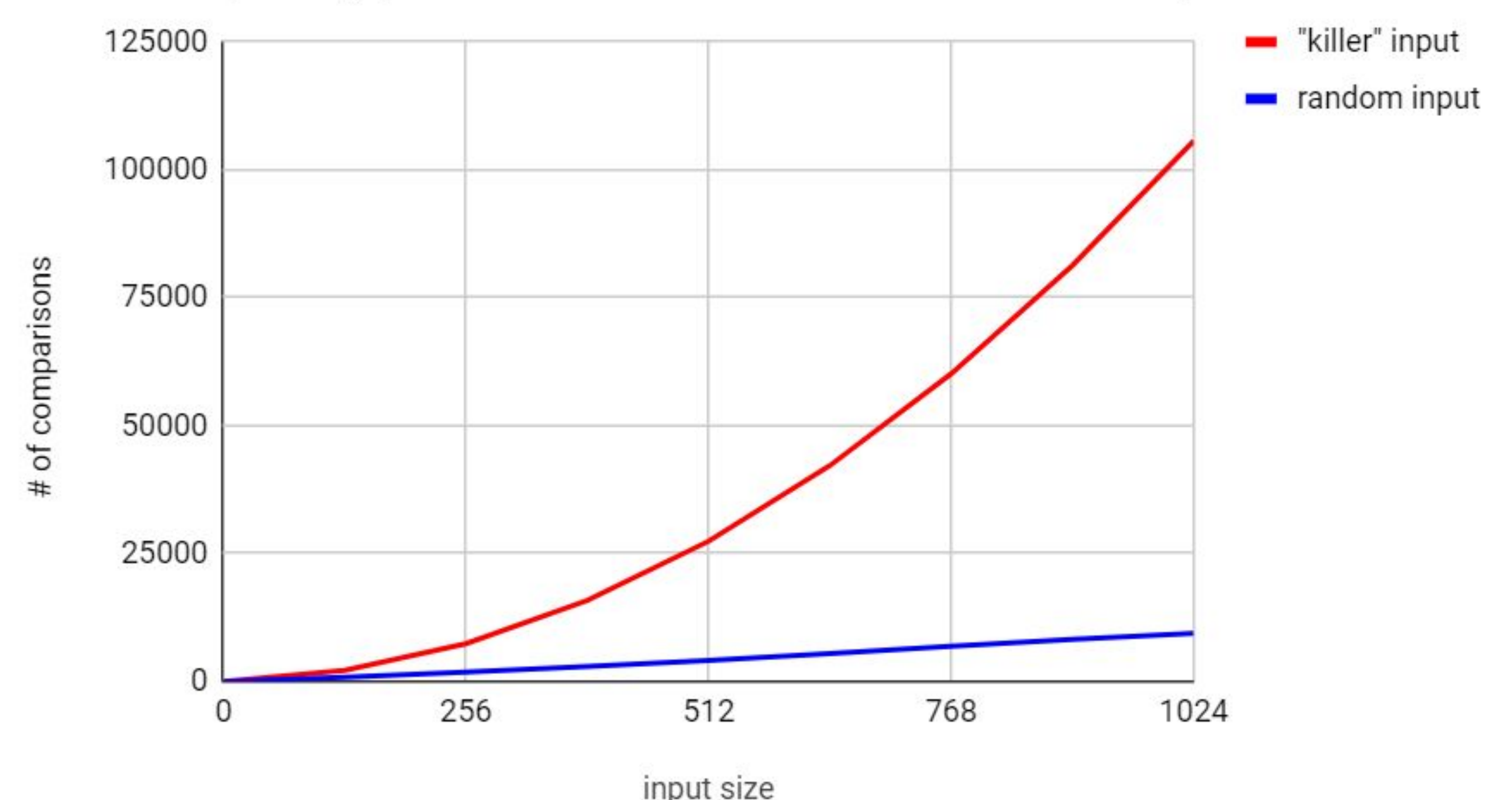
Detection mechanism:

1. Locate input and vulnerable function
2. Check for possible path from input
3. Simulate execution from input to function
4. Confirm that simulated input data reaches vulnerable function and check for filtering

Exploiting Quicksort

- Used McIlroy's "killer" input generator
- Demonstrated slowdowns in multiple *libc* versions
  - Tested on 10000-item sorts
  - Larger inputs give larger slowdowns

libc version	Slowdown (killer vs random input)
glibc	~130x
dietlibc	~94x
FreeBSD	~115x



Evaluation

Results

- Executed our tool on test binaries and on Linux user-space binaries
- Test binary results:
  - Our tool detected input passed to quicksort for simple test cases
- Real-world results:
  - Analyzed ~2000 binaries from Ubuntu 16.04 desktop install
  - Found 91 binaries with paths from input to quicksort function
  - Of these, 5 had paths of 100 nodes or less -- good candidates for further analysis

Future Work

- Detection steps 3-4 are currently too slow and memory-intensive to be practical on real binaries
  - *Path Explosion*: branches cause exponential increase in time and space requirements during symbolic execution
  - Symbolic execution efficiency could be improved by ignoring untainted functions (functions which do not perform any work on user input)

If interested contact Kai Walberg (kwalberg@macalester.edu)
Work performed under REU Site program
supported by NSF grant #1659886

USC Viterbi
School of Engineering
Information Sciences Institute