Porject

Monday, May 18, 2020 3:40 PM

Stages of Fuzzing

Pre-Fuzzing: Program Knowledge.

Mutation-based fuzzers (Zulu)

- an input corpus, or a set of program inputs to feed to the indicated interfaces, in order to effecti
- Initial input corpora are generally created manually, but large sets of inputs do pre-exist for som
- Corpus are repeatedly modified or corrupted to produce new test cases
- the fuzzer has no specific knowledge of the program being fuzzed or the correct format

Generation-based fuzzers (Sulley)

- input specifications, such as ex- pected file formats or protocol descriptions. Generation- based better code coverage and deeper program state exploration than mutation-based fuzzers

Evolutionary fuzzers

- an input corpus
- test cases are created based on feedback from the instrumentation so that they evolve
- can be an extension of either mutation or generation fuzzing, as either technique can be test cases.

fuzzers auto- matically iterate over the next three stages of the fuzzing process. A fuzzer will generate order inputs to send to the program, monitor the program for interesting program states, and repeat, terminates the fuzzer.

Stage 1: Generate Inputs.

Mutation-based fuzzers

- In the first iteration of this stage, a fuzzer manipulates input provided in its input corpus. Later, t collection of inter- esting inputs based on results from monitoring the program and evaluating in
- Mutation-based fuzzers mutate an interesting input by altering some portion of the input.
  - 1) where to mutate (including the length of the mutation)
  - 2) what new value to use for the mutation
  - Fuzzers use many techniques to make these decisions. Common techniques include rando sections), specific bit flips, integer increments, and integer bound analysis and substitutio
  - Walking bit flipping and byte flipping involves inverting sequences of bits at differ [81]. Deleting segments of the test case and also splicing in sections from other te

vely generate inputs
e applications

or syntax to use for inputs

fuzzers typically achieve

towards a specific goal
e guided to create the new

new inputs, down-select and
generally until the user

the fuzzer may adjust its
nput performance.

omization (from bits to entire
n.
ent places in the test case
est cases are common

mutation approaches [80]. Another approach is to select sections of the test case
inserting copies at random positions [100]. A recent example of a mutation fuzzer
[81].

Generation-based fuzzers

- generate inputs by creating a new input according to the input specification (like a grammar of
- Because the input search space is finite, generation-based fuzzers can explore all possible specifi generation-based fuzzers to accurately measure how much of the input space they have explore

Evolutionary fuzzers

- like mutation-based fuzzers, generate new inputs by manipulating previous interesting inputs
- Generally, evolutionary fuzzers either mutate one input or select two or more inputs and perfor components of the selected inputs to make a new input
- infinite input search space makes it difficult to estimate how much of the input space has been e
- computationally challenging

Symbolic Execution

- a static analysis technique that can help to generate new inputs that increase fuzzer coverage
- generating new inputs for mutation- based and evolutionary fuzzers
- analyzes a program to find constraints, or limitations, on data values inside the program
- works by abstracting input data into symbolic values, or data that might take on many concrete the program using these sym- bolic values
- *A symbolic state*
  - includes constraints representing the se- ries of branch decisions necessary to reach that
- the symbolic state is modeled as a set of constraints that encode, for each input variable, the ra path.
- *A con- straint*
  - solver then solves these constraints, either returning a valid concrete assignment prov- ing that no such assignment exists
- computationally costly

Challenges

- low coverage, even when pairing their specialized input generation techniques with symbolic ex
- Effective input generation remains a research challenge

Stage 2: Select Inputs

a fuzzer must use its input corpus effectively and minimize the computation spent to discover new pro

Input test scheduling, or seed selection

- scheduling ranks and selects inputs and input or- der, anticipating which new inputs are most lik interesting program states.

corpus minimization, reducing the number of inputs

input minimization, reduc- ing the size of each input

and repeat them by
r is Zulu from NCC Group

a programming language).
ied inputs, which allows
d.

m crossover, combining

explored

values, and stepping through

state, i.e., a path.
nge of valid values along the

to the input variables or

ecution.

gram paths.

ely to lead to new and disjoint

Stage 3: Monitor Program

In the third stage of the fuzzing process, the fuzzer feeds the program chosen inputs and monitors the
interesting program states.

- "Interesting program states" exhibit a specific program behav- ior in that state, like a crash
- provide descriptions of that state back to the user for analysis

Stage 4: Evaluate Inputs

- Many fuzzers use code coverage to measure the utility of an input: if the input causes execution (generally, a new basic block), that input has increased cov- erage and is rated highly.
- data coverage
- bug discovery

Evolutionary fuzzers

- valuate inputs and rank them based on feedback
- use input rank both in generating new inputs and in selecting and sending inputs

Mutation-based and generation-based fuzzers

- do not generally require input performance feedback, but the same metrics can help to evaluate overall.

Post-Fuzzing: Interesting Program States.

- user analyzes interesting program states output
- the user analyzes the state to deter- mine the root cause of the interesting behavior in that state
- perform triage to decide which interesting program states merit further investigation.
- Challenges
  - Some automated tools attempt to deduplicate fuzzer outputs [28] or root causes, but the triage and root cause analysis remain research challenges

Comparing Fuzzers

Difficulties

- inability to explore the entire input search space which biases performance measures
- lack of ground truth, which makes validation a difficult and manual process
- randomness exploited in the fuzzing process, which introduces the need for statistical testing
- Effective fuzzer comparison remains a significant research challenge.

Machine learning in Fuzzing

- AFL: unsupervised learning; integrating genetic algorithms (GAs) into the input generation pro
- All three ML methods applied in symbolic execution, to constraint equation solve times

program to identify

of a new part of code

e a fuzzer's performance

e.

se are often imperfect, and

ocess

- supervised and unsupervised learning have been applied to post-fuzzing processes primarily for and root cause categorization

Input generation

Genetic Algorithms

- unsupervised ML
- Often in evolutionary fuzzers
- Steps
  - 1) generate a small base population of inputs
    - a set of seed program inputs
    - the GA will mutate these seed inputs to explore the code space with the goal of dis new paths in the code
  - 2) perform transformations on inputs
  - 3) measure the performance of the transformed inputs.
- Fitness function
  - 1) the performance of the fuzzer
  - 2) the ability of the fuzzer to identify certain types of bugs
  - 3) the tendency to get stuck in local minima.
  - For example
    - Code coverage
    - Dynamic Markov Model (DMM) heuristic
      - In con- trast to code coverage, this setup allows for more precise control of t guiding it towards specific parts of the code that may contain a bug or flaw.

Deep Learning and Neural Networks

generation-based and mutation- based fuzzers

- Recurrent neural networks
- integrated DL into AFL to increase fuzzer coverage by selecting input bytes to mutate

Post-fuzzing

Interesting Program States

- Users triage program states by
  - 1) evaluating for uniqueness
  - 2) analyzing triaged states to determine reproducibility and a root cause
  - 3) when fuzzing for vulnerability assessment, determining whether a root cause is exploita
- Usage
  - categorize crashes (triage)
    - grouping crashes with similar call stacks
  - categorize bugs (root cause analysis)

Challenges

ML is rarely applied to post-fuzzing tasks for two reasons

- 1) ML results and algorithms are often difficult to interpret crash triage

scovering

the fuzzer,

able.

- 2) appropriate training data sets are sparse.

Instrumentation

- detecting when the program crashes
- logging which test case caused the crash
- detect subtle forms of memory corruption
- measure code coverage, which parts of the target application have been tested so far a have not

it should be noted that all forms of instrumentation slow the target program down, often sign [78]. It is therefore important to consider what is needed for each specific fuzzing scenario as trade- off between more speed and better instrumentation.

File format fuzzers

- used for fuzzing any application that receives a file as an input
- Applications: Image viewers, movie players, PDF readers, word processors and audio file
- Example: AFL

Protocal fuzzers

- Protocol fuzzers are used to fuzz clients and servers that communicate over a network, using TCP/IP although any protocol stack can be fuzzed.
- Applications: Mail servers, web servers and FTP clients
- Example: SPIKE

1. genetic algorithm https://trace.tennessee.edu/utk_graddiss/1347/
   Novelty: write a parser for specific file format

nd which

nificantly
there is a

e players

typically