DARWIN: Survival of the Fittest
Fuzzing Mutators

Patrick Jauernig, Domagoj Jakobovic, Stjepan Picek, Emmanuel Stapf, Ahmad-Reza Sadeghi
Motivation

- Fuzzing research is quite mature

- Key drivers for adoption:
  - Enabling technologies (firmware rehosting, ...)
  - Platforms (OSS-Fuzz, ClusterFuzz)
- Lots of technical improvements (fast snapshots, coverage tracing)
Motivation

- Fuzzing research is quite mature

- Key drivers for adoption:
  - Enabling technologies (firmware rehosting, ...)
  - Platforms (OSS-Fuzz, ClusterFuzz)
- Lots of technical improvements (fast snapshots, coverage tracing)

- Algorithmic improvements can increase efficiency across targets
Background – Fuzzing

- Dynamic analysis technique
  - Applies random inputs (testcases) to a target to see if it crashes

- Traditional separation: grammar-based vs. mutational

\[
\begin{align*}
S & \rightarrow xA | yS \\
A & \rightarrow yA | zB \\
B & \rightarrow z
\end{align*}
\]
Background – Mutational Fuzzers
Background – Mutational Fuzzers

Input → Deterministic Stage → Program
Background – Mutational Fuzzers

Input → Deterministic Stage → Havoc Stage → Program
Background – Mutational Fuzzers

Input → Deterministic Stage → Havoc Stage → Splicing → Program
Background – Mutational Fuzzers

Diagram:
- Input
- Deterministic Stage
- Havoc Stage
- Splicing
- Program
Background – Mutational Fuzzers

Diagram:
- Input
  - Deterministic Stage
  - Havoc Stage
  - Splicing
  - Program
  - Feedback

Diagram describes the stages of mutation-based fuzzing.
Background – Mutation Scheduling

- Input
- Deterministic Stage
- Havoc Stage
- Splicing
- Program
- Feedback
- Selected Mutation
- Mutation Scheduler

28.02.2023
Related Work in Algorithmic Improvements
## Related Work in Algorithmic Improvements

**Mutation Schedulers**

| MOPT [Lyu et al., USENIX Security 2019] |
|----------------------------------------|
| Fuzzergym [Drozd et al., arXiv 2018]   |
| [Böttinger et al., SPW 2018]           |
Related Work in Algorithmic Improvements

Mutation Schedulers

- MOPT [Lyu et al., USENIX Security 2019]
- Fuzzergym [Drozd et al., arXiv 2018]
  [Böttinger et al., SPW 2018]

Location Optimization

- FairFuzz [Lemieux et al., ASE 2018]
- Steelix [Li et al., ESEC/FSE 2017]
  [Rajpal et al., arXiv 2017]
### Related Work in Algorithmic Improvements

| Mutation Schedulers          |
|-----------------------------|
| MOPT [Lyu et al., USENIX Security 2019] |
| Fuzzergym [Drozd et al., arXiv 2018] |
| [Böttinger et al., SPW 2018]  |

| Location Optimization       |
|-----------------------------|
| FairFuzz [Lemieux et al., ASE 2018] |
| Steelix [Li et al., ESEC/FSE 2017] |
| [Rajpal et al., arXiv 2017]  |

| Other Scheduling Approaches |
|-----------------------------|
| EcoFuzz [Yue et al., USENIX Security 2020] |
| NeuFuzz [Wang et al., IEEE Access 2019]  |
Related Work in Algorithmic Improvements

Mutation Schedulers

- **MOPT** [Lyu et al., USENIX Security 2019]
- **Fuzzergym** [Drozd et al., arXiv 2018]
  - [Böttinger et al., SPW 2018]

Location Optimization

- **FairFuzz** [Lemieux et al., ASE 2018]
- **Steelix** [Li et al., ESEC/FSE 2017]
  - [Rajpal et al., arXiv 2017]

Other Scheduling Approaches

- **EcoFuzz** [Yue et al., USENIX Security 2020]
- **NeuFuzz** [Wang et al., IEEE Access 2019]

- Fail to show improvements in practice
- Introduce per-target parameters
## Related Work in Algorithmic Improvements

| Mutation Schedulers | Location Optimization | Other Scheduling Approaches |
|---------------------|-----------------------|-----------------------------|
| MOPT [Lyu et al., USENIX Security 2019] | FairFuzz [Lemieux et al., ASE 2018] | EcoFuzz [Yue et al., USENIX Security 2020] |
| Fuzzergym [Drozd et al., arXiv 2018] | Steelix [Li et al., ESEC/FSE 2017] | NeuFuzz [Wang et al., IEEE Access 2019] |
| [Böttinger et al., SPW 2018] | [Rajpal et al., arXiv 2017] | |

- Fail to show improvements in practice
- Introduce per-target parameters

- Optimize location, but not the associated operation
- Expensive (to integrate)
Related Work in Algorithmic Improvements

**Mutation Schedulers**
- MOPT [Lyu et al., USENIX Security 2019]
- Fuzzergym [Drozd et al., arXiv 2018]
  [Böttinger et al., SPW 2018]

**Location Optimization**
- FairFuzz [Lemieux et al., ASE 2018]
- Steelix [Li et al., ESEC/FSE 2017]
  [Rajpal et al., arXiv 2017]

**Other Scheduling Approaches**
- EcoFuzz [Yue et al., USENIX Security 2020]
- NeuFuzz [Wang et al., IEEE Access 2019]

- Fail to show improvements in practice
- Introduce per-target parameters
- Optimize location, but not the associated operation
- Expensive (to integrate)
- Optimization goal applied very early in fuzzing loop
- Interesting: combining seed selection and mutation scheduling
DARWIN Mutation Scheduler

Havoc Stage

Select Mutation

8% Bitflip
1% ...
0% ...
9% Overwrite Bytes

New Probability Distribution

Testcase

Feedback
DARWIN – Evolution Strategy

Parent

8%
1%
0%
9%

Perturbation

Children

5% 1% 0% 9%
8% 1% 2% 9%
8% 1% 5% 9%
8% 3% 0% 9%
DARWIN – Evolution Strategy

Parent

Children

Perturbation

Fitness function determines next parent

fitness = #new unique paths
DARWIN – Evolution Strategy

Parent

Children

Perturbation

Fitness function determines next parent

fitness = #new unique paths
DARWIN – Evolution Strategy

- Very simple and efficient
- Problem: very local algorithm
  - Low probability of escaping local optima
DARWIN – Evolution Strategy

- Very simple and efficient
- Problem: very local algorithm
  - Low probability of escaping local optima

- Solution: multi-parent ES
  - $\mu$ parents, $\lambda$ children
    - 5 parents, 4 children seemed best
  - Cycle through best parent solutions
  - In addition: Binary representation
DARWIN – Evolution Strategy

- Very simple and efficient
- Problem: very local algorithm
  - Low probability of escaping local optima

- Solution: multi-parent ES
  - \( \mu \) parents, \( \lambda \) children
    - 5 parents, 4 children seemed best
    - Cycle through best parent solutions
    - In addition: Binary representation
DARWIN - Contributions

- Leveraging Evolution Strategy to optimize mutation scheduling
- Keeping execution speed high
- No target-dependent parameters
- Easy to integrate into mutational fuzzers
Evaluation

- Is mutation scheduling a dynamic problem?
- Does it make sense to trade in speed for efficiency?
- Is there an improvement in
  - Coverage?
  - Time to coverage?
  - Bugs?
Evaluation - Coverage

- Binutils suite, bsdtar, djpeeg, jhead, tcpdump
- Edge coverage: +6.77% vs. MOPT, +1.73% vs. AFL
  - +4.38% vs. static variant (AFL-S)!
- At disadvantage for targets expecting highly-structured input
Evaluation – Mutation Histories

cxxfilt  DARWIN  MOPT  AFL
Evaluation – Mutation Histories

- Optimum is relatively static
- Cycles are still wasted on optimization
Evaluation – Mutation Histories

- Optimum is relatively static
- Cycles are still wasted on optimization
Evaluation – Mutation Histories

cxxfilt

DARWIN

MOPT

AFL

- Optimum is relatively static
- Cycles are still wasted on optimization

size

- Still adapting with more diverse parents
- Further optimization useful here
Evaluation - FuzzBench

FuzzBench
Evaluation - FuzzBench
Evaluation - MAGMA

- MAGMA: Benchmark suite to find backported bugs

- Different reports, in this case: survival analysis ("time to bug")

- DARWIN finds 15/21 bugs fastest

Magma: A ground-truth fuzzing benchmark, Hazimeh et al., 2020
Evaluation - Crashes

- Crash experiment based on coverage targets
  - Max: unique bugs within one run
  - Uniq: unique bugs over all ten runs

- DARWIN variants outperform MOPT, AFL, EcoFuzz, and AFL-S
- One novel bug in objcopy: memory leak

|               | DARWIN | AFL | AFL-S | MOPT | EcoFuzz-D | EcoFuzz |
|---------------|--------|-----|-------|------|-----------|---------|
| Max           | 7      | 4   | 5     | 1    | 18        | 1       |
| Unique        | 20     | 12  | 12    | 2    | 26        | 1       |
Conclusion

- DARWIN is the first ES-based mutation scheduler
- Adaptive optimization outperforms static optimization
- Significant improvement in bug-finding capabilities

Contact: darwin@sanctuary.dev