dippy_gram: Grammar-Aware, Coverage-Guided Differential Fuzzing (WIP)

Ben Kallus, Sean W. Smith, James Utley

Dartmouth College

May 25, 2023

Overview

- Single-target fuzzing is good at finding bugs that cause crashes and memory errors.

Overview

- Single-target fuzzing is good at finding bugs that cause crashes and memory errors.
- It's not so good at finding bugs that don't.

Overview

- Single-target fuzzing is good at finding bugs that cause crashes and memory errors.
- It's not so good at finding bugs that don't.
- Differential fuzzing is the search for inputs that cause implementations of the same specification to diverge.

Overview

- Single-target fuzzing is good at finding bugs that cause crashes and memory errors.
- It's not so good at finding bugs that don't.
- Differential fuzzing is the search for inputs that cause implementations of the same specification to diverge.
- dippy_gram is a differential fuzzer that uses coverage information, grammar-based mutations, and a novel bug minimization scheme to detect crashing and non-crashing bugs.

Overview

- Single-target fuzzing is good at finding bugs that cause crashes and memory errors.
- It's not so good at finding bugs that don't.
- Differential fuzzing is the search for inputs that cause implementations of the same specification to diverge.
- dippy_gram is a differential fuzzer that uses coverage information, grammar-based mutations, and a novel bug minimization scheme to detect crashing and non-crashing bugs.
- We apply dippy_gram to a suite of URL parsers, and have discovered numerous parser differentials, both crashing and non-crashing.

Results

So far, we've applied dippy_gram only to URL parsers. Work is ongiong to apply it to other domains. A summary of some bugs that we've found using dippy_gram:

Results

So far, we've applied dippy_gram only to URL parsers. Work is ongiong to apply it to other domains. A summary of some bugs that we've found using dippy_gram:

- urllib3 (∼350m downloads/month, the most popular package on PyPI)
  - Reported 4 bugs. We patched 2, and 2 were patched by others. One of those was awarded a $300 bounty.

Results

So far, we've applied dippy_gram only to URL parsers. Work is ongiong to apply it to other domains. A summary of some bugs that we've found using dippy_gram:

- urllib3 (∼350m downloads/month, the most popular package on PyPI)
  - Reported 4 bugs. We patched 2, and 2 were patched by others. One of those was awarded a $300 bounty.
- rfc3986 (∼13m downloads/month)
  - Reported and patched 3 bugs.

Results

So far, we've applied dippy_gram only to URL parsers. Work is ongiong to apply it to other domains. A summary of some bugs that we've found using dippy_gram:

- urllib3 (∼350m downloads/month, the most popular package on PyPI)
  - Reported 4 bugs. We patched 2, and 2 were patched by others. One of those was awarded a $300 bounty.
- rfc3986 (∼13m downloads/month)
  - Reported and patched 3 bugs.
- CPython standard library
  - Reported and patched 3 bugs. Another patch is underway.
  - Currently writing a proposal to deprecate CPython's URL shotgun parser and replace it with something more principled.

Results

So far, we've applied dippy_gram only to URL parsers. Work is ongiong to apply it to other domains. A summary of some bugs that we've found using dippy_gram:

- urllib3 (∼350m downloads/month, the most popular package on PyPI)
  - Reported 4 bugs. We patched 2, and 2 were patched by others. One of those was awarded a $300 bounty.
- rfc3986 (∼13m downloads/month)
  - Reported and patched 3 bugs.
- CPython standard library
  - Reported and patched 3 bugs. Another patch is underway.
  - Currently writing a proposal to deprecate CPython's URL shotgun parser and replace it with something more principled.
- We have also found bugs in yarl, furl, hyperlink, and others, but our PRs have not yet been merged.

Relationship to Prior Work

Our fuzzer draws heavily from NEZHA (Petsios et. al, 2017). We distinguish our work by

Relationship to Prior Work

Our fuzzer draws heavily from NEZHA (Petsios et. al, 2017). We distinguish our work by

- Using grammar-based mutations.

Relationship to Prior Work

Our fuzzer draws heavily from NEZHA (Petsios et. al, 2017). We distinguish our work by

- Using grammar-based mutations.
- Examining not just exit statuses, but also program stdout.

Relationship to Prior Work

Our fuzzer draws heavily from NEZHA (Petsios et. al, 2017). We distinguish our work by

- Using grammar-based mutations.
- Examining not just exit statuses, but also program stdout.
- Minimizing results to avoid duplicate bug reporting.

Relationship to Prior Work

Our fuzzer draws heavily from NEZHA (Petsios et. al, 2017). We distinguish our work by

- Using grammar-based mutations.
- Examining not just exit statuses, but also program stdout.
- Minimizing results to avoid duplicate bug reporting.
- Uses AFL instrumentation, and is thus compatible with many interpreted languages through python-afl, Kelinci, and ruby-afl.

Relationship to Prior Work

Our fuzzer draws heavily from NEZHA (Petsios et. al, 2017). We distinguish our work by

- Using grammar-based mutations.
- Examining not just exit statuses, but also program stdout.
- Minimizing results to avoid duplicate bug reporting.
- Uses AFL instrumentation, and is thus compatible with many interpreted languages through python-afl, Kelinci, and ruby-afl.
- Pretty simple; $\sim$500 loc (10x fewer than NEZHA)

The Fuzzing Loop

1. Dequeue an input I from the input queue (initially a seed corpus).

The Fuzzing Loop

1. Dequeue an input I from the input queue (initially a seed corpus).
2. Run I through a group of instrumented programs.

The Fuzzing Loop

1. Dequeue an input *I* from the input queue (initially a seed corpus).
2. Run *I* through a group of instrumented programs.
3. Deduplicate each program's control flow trace into a sequence of sets of CFG edges.

The Fuzzing Loop

1. Dequeue an input *I* from the input queue (initially a seed corpus).
2. Run *I* through a group of instrumented programs.
3. Deduplicate each program's control flow trace into a sequence of sets of CFG edges.
4. If a meaningful differential is observed, report and GOTO 1.

The Fuzzing Loop

1. Dequeue an input I from the input queue (initially a seed corpus).
2. Run I through a group of instrumented programs.
3. Deduplicate each program's control flow trace into a sequence of sets of CFG edges.
4. If a meaningful differential is observed, report and GOTO 1.
5. If this sequence has not been encountered previously, mutate I a few times and place the mutants onto the queue.

The Fuzzing Loop

1. Dequeue an input *I* from the input queue (initially a seed corpus).
2. Run *I* through a group of instrumented programs.
3. Deduplicate each program's control flow trace into a sequence of sets of CFG edges.
4. If a meaningful differential is observed, report and GOTO 1.
5. If this sequence has not been encountered previously, mutate *I* a few times and place the mutants onto the queue.
6. GOTO 1.

What is a meaningful differential?

- We want to avoid reporting results that are expected due to support for optional parts of a specification.

What is a meaningful differential?

- We want to avoid reporting results that are expected due to support for optional parts of a specification.
- For example, RFC 3986 permits a URL parser to ignore or reject password fields from URLs, because their use is deprecated.

What is a meaningful differential?

- We want to avoid reporting results that are expected due to support for optional parts of a specification.
- For example, RFC 3986 permits a URL parser to ignore or reject password fields from URLs, because their use is deprecated.
- We use configurable program output comparators to ensure that the fuzzer does not report these uninteresting differences.

What is a meaningful differential?

- We want to avoid reporting results that are expected due to support for optional parts of a specification.
- For example, RFC 3986 permits a URL parser to ignore or reject password fields from URLs, because their use is deprecated.
- We use configurable program output comparators to ensure that the fuzzer does not report these uninteresting differences.
- This allows us to choose an equivalence that suits our target specification. For example, we can specify that a portion of program output is to be considered case insensitively when determining whether a meaningful difference has been observed.

What is a meaningful differential?

- We also want to avoid reporting duplicate results.

What is a meaningful differential?

- We also want to avoid reporting duplicate results.
- We provide support for minimization modules that reduce bug-inducing inputs to a minimal form in which the bug is still reproduced. The trace sets from these minimal bug-inducing inputs can then be used for classification.

What is a meaningful differential?

- We also want to avoid reporting duplicate results.
- We provide support for minimization modules that reduce bug-inducing inputs to a minimal form in which the bug is still reproduced. The trace sets from these minimal bug-inducing inputs can then be used for classification.
- For URL, one such module iteratively deletes byte sequences (similar to *afl-tmin*) from a bug-inducing input until we arrive at a minimal length input that reproduces the differential.

What is a meaningful differential?

- We also want to avoid reporting duplicate results.
- We provide support for minimization modules that reduce bug-inducing inputs to a minimal form in which the bug is still reproduced. The trace sets from these minimal bug-inducing inputs can then be used for classification.
- For URL, one such module iteratively deletes byte sequences (similar to *afl-tmin*) from a bug-inducing input until we arrive at a minimal length input that reproduces the differential.
  - For parser differentials, this means maintaining parser exit statuses and parse tree equivalence.

What is a meaningful differential?

- We also want to avoid reporting duplicate results.
- We provide support for minimization modules that reduce bug-inducing inputs to a minimal form in which the bug is still reproduced. The trace sets from these minimal bug-inducing inputs can then be used for classification.
- For URL, one such module iteratively deletes byte sequences (similar to *afl-tmin*) from a bug-inducing input until we arrive at a minimal length input that reproduces the differential.
  - For parser differentials, this means maintaining parser exit statuses and parse tree equivalence.
- The minimized input's trace is recorded, and future inputs with the same trace after minimization are ignored.

Mutations

We employ two types of mutation operations:

Mutations

We employ two types of mutation operations:

- Random mutations:
    - Random byte deletion
    - Random byte insertion
    - Random byte replacement

Mutations

We employ two types of mutation operations:

- Random mutations:
  - Random byte deletion
  - Random byte insertion
  - Random byte replacement
- Grammar-based mutations: (requires a grammar)
  - Random parse subtree replacement
  - Random parse subtree duplication

Too-permissive scheme validation

`.://example.com`

Parser	Scheme	Host	Path
CPython	`.`	`example.com`	
rfc3986			`.://example.com`
urllib3		`.`	`//example.com`

Bad IPv6 hostname validation

`http://[::1]example.com`

Parser	Host
CPython	`::1`

Bad IPv6 hostname validation

`http://[::1]example.com`

Parser	Host
CPython	`::1`
everything else	rejects

Bad scheme validation

`evil.com://good.com`

Parser	Scheme	Host	Path
CPython	`evil.com`	`good.com`	
urllib3		`evil.com`	`//good.com`

Bad port validation

`http://example.com:  +8_0`

Parser	Scheme	Host	Port
CPython	http	`example.com`	80
Hyperlink	http	`example.com`	80
rfc3986	http	`example.com`	80

Improper Unicode handling

`http://example.com:1\u06F0`

Parser	Scheme	Host	Port	Path
CPython	http	`example.com`	10	
Hyperlink	http	`example.com`	10	/
rfc3986	http	`example.com`	10	

What’s left to do

What’s left to do

- Detecting bug composition

What's left to do

- Detecting bug composition

What's left to do

- Detecting bug composition
  - If we have encountered bugs A and B, the presence of both at once should not be considered a new bug.

What's left to do

- Detecting bug composition
  - If we have encountered bugs A and B, the presence of both at once should not be considered a new bug.
  - We currently solve this by ensuring that mutations are small enough that multiple bugs are not likely to be introduced in the same mutation step.

What's left to do

- Detecting bug composition
  - If we have encountered bugs A and B, the presence of both at once should not be considered a new bug.
  - We currently solve this by ensuring that mutations are small enough that multiple bugs are not likely to be introduced in the same mutation step.
- Experiments

What's left to do

- Detecting bug composition
    - If we have encountered bugs A and B, the presence of both at once should not be considered a new bug.
    - We currently solve this by ensuring that mutations are small enough that multiple bugs are not likely to be introduced in the same mutation step.
- Experiments
    - We have a lot of experiments left to run, including
        - Evaluation of different mutation combinations.
        - Evaluation of differential fuzzing across programming language boundaries.
        - Comparison to symbolic execution-based approaches.

What's left to do

- Detecting bug composition
  - If we have encountered bugs A and B, the presence of both at once should not be considered a new bug.
  - We currently solve this by ensuring that mutations are small enough that multiple bugs are not likely to be introduced in the same mutation step.
- Experiments
  - We have a lot of experiments left to run, including
    - Evaluation of different mutation combinations.
    - Evaluation of differential fuzzing across programming language boundaries.
    - Comparison to symbolic execution-based approaches.
- Extending our approach to other formats

What's left to do

- Detecting bug composition
    - If we have encountered bugs A and B, the presence of both at once should not be considered a new bug.
    - We currently solve this by ensuring that mutations are small enough that multiple bugs are not likely to be introduced in the same mutation step.
- Experiments
    - We have a lot of experiments left to run, including
        - Evaluation of different mutation combinations.
        - Evaluation of differential fuzzing across programming language boundaries.
        - Comparison to symbolic execution-based approaches.
- Extending our approach to other formats
    - HTTP (ongoing)

What's left to do

- Detecting bug composition
  - If we have encountered bugs A and B, the presence of both at once should not be considered a new bug.
  - We currently solve this by ensuring that mutations are small enough that multiple bugs are not likely to be introduced in the same mutation step.
- Experiments
  - We have a lot of experiments left to run, including
    - Evaluation of different mutation combinations.
    - Evaluation of differential fuzzing across programming language boundaries.
    - Comparison to symbolic execution-based approaches.
- Extending our approach to other formats
  - HTTP (ongoing)
- Fuzzing to enumerate differences between standards.

What's left to do

- Detecting bug composition
  - If we have encountered bugs A and B, the presence of both at once should not be considered a new bug.
  - We currently solve this by ensuring that mutations are small enough that multiple bugs are not likely to be introduced in the same mutation step.
- Experiments
  - We have a lot of experiments left to run, including
    - Evaluation of different mutation combinations.
    - Evaluation of differential fuzzing across programming language boundaries.
    - Comparison to symbolic execution-based approaches.
- Extending our approach to other formats
  - HTTP (ongoing)
- Fuzzing to enumerate differences between standards.
- Differential fuzzing across architecture-specific code using AFL's QEMU mode.

What's left to do

- Detecting bug composition
  - If we have encountered bugs A and B, the presence of both at once should not be considered a new bug.
  - We currently solve this by ensuring that mutations are small enough that multiple bugs are not likely to be introduced in the same mutation step.
- Experiments
  - We have a lot of experiments left to run, including
    - Evaluation of different mutation combinations.
    - Evaluation of differential fuzzing across programming language boundaries.
    - Comparison to symbolic execution-based approaches.
- Extending our approach to other formats
  - HTTP (ongoing)
- Fuzzing to enumerate differences between standards.
- Differential fuzzing across architecture-specific code using AFL's QEMU mode.
- A better name!

Thank You.

Contact me! (benjamin.p.kallus.gr@dartmouth.edu)

This work was funded by the DARPA GAPS and SafeDocs programs.

https://github.com/kenballus/url_differential_fuzzing