NEZHA: Efficient Domain-Independent Differential Testing

Theofilos Petsios*, Adrian Tang*, Salvatore Stolfo, Angelos D. Keromytis, and Suman Jana

IEEE Security & Privacy (Oakland) 2017
*Joint primary authors

Columbia University
Differential Testing
Differential Testing

• Fuzzing: memory corruption bugs

• Differential testing: logic bugs
Differential Testing

- Multiple apps of the same functionality
- Applications usually follow some specification/standard
Differential Testing

- Multiple apps of the same functionality
- All usually to follow some specification/standard
- Deviations from the specifications/standards likely to be bugs
Differential Testing

- Multiple apps of the same functionality
- All usually to follow some specification/standard
- Deviations from the specifications/standards likely to be bugs
Differential Testing

- Multiple apps of the same functionality
- All usually to follow some specification/standard
- Deviations from the specifications/standards likely to be bugs
- Applicable in different domains (e.g., compiler testing)
Key challenges

• Existing tools are domain-specific

• Inefficient input generation
Goal of **NEZHA**

Efficient domain-independent differential testing
Domain-Independent Evolutionary Testing

Seed Inputs

Input Generation
Guidance

New Inputs

Application
Domain-Independent Evolutionary Testing

Seed Inputs

Input Generation Guidance

New Inputs

Runtime Monitoring

...
Domain-Independent Evolutionary Testing

Seed Inputs → State Information → New Inputs → Application → Runtime Monitoring → State Information → Input Generation Guidance → Seed Inputs

- New Inputs
- Application
- State Information
- Runtime Monitoring
- Input Generation Guidance
- Seed Inputs
Evolve an input corpus that is \textit{guided} based on an \textit{analysis engine}
Evolve an input corpus that is \textit{guided} based on an \textit{analysis engine}
Code Coverage - Single-App

All possible code paths
Code Coverage - Single-App

Input Corpus

Per-Input Coverage

All possible code paths

Code coverage - Global

Code coverage - Input
Code Coverage - Single-App

Input Corpus

Per-Input Coverage

All possible code paths
Code coverage - Global
Code coverage - Input

Input 1
Code Coverage - Single-App

Input Corpus

Per-Input Coverage

Input 1

All possible code paths

Code coverage - Global

Code coverage - Input
Code Coverage - Single-App

Input Corpus

Per-Input Coverage

All possible code paths

Code coverage - Global

Code coverage - Input
Code Coverage - Single-App

Input Corpus

Per-Input Coverage

Input 1

All possible code paths

Code coverage - Global

Code coverage - Input
Code Coverage - Single-App

Input Corpus

Per-Input Coverage

Input 1

Input 2

All possible code paths

Code coverage - Global

Code coverage - Input
Code Coverage - Single-App

Input Corpus

Input 1

Per-Input Coverage

Input 2

All possible code paths

Code coverage - Global

Code coverage - Input
Code Coverage - Single-App

Input Corpus

Per-Input Coverage

Input 1

Input 2

Input 2

All possible code paths

Code coverage - Global

Code coverage - Input
Code Coverage - Single-App

Input Corpus

<table>
<thead>
<tr>
<th>Input</th>
<th>Coverage</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input 1</td>
<td>![Coverage Icon]</td>
</tr>
<tr>
<td>Input 2</td>
<td>![Coverage Icon]</td>
</tr>
</tbody>
</table>

Per-Input Coverage

- All possible code paths
- Code coverage - Global
- Code coverage - Input
Code Coverage - Single-App

Input Corpus

Per-Input Coverage

Input 1
Input 2

All possible code paths

Code coverage - Global

Code coverage - Input

Input 3
Code Coverage - Single-App

Input Corpus

Per-Input Coverage

Input 1

Input 2

Input 3

All possible code paths

Code coverage - Global

Code coverage - Input
Code Coverage - Single-App

Input Corpus

<table>
<thead>
<tr>
<th>Input 1</th>
<th>Input 2</th>
<th>Input 3</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image" alt="Coverage" /></td>
<td><img src="image" alt="Coverage" /></td>
<td><img src="image" alt="Coverage" /></td>
</tr>
</tbody>
</table>

- **Per-Input Coverage**
  - Input 1
  - Input 2
  - Input 3

- **All possible code paths**
  - Code coverage - Global
  - Code coverage - Input

Input 4
Code Coverage - Single-App

- All possible code paths
- Code coverage - Global
- Code coverage - Input

Input Corpus

Input 1
Input 2
Input 3

Per-Input Coverage

Input 4

Discard!
Domain-Independent Evolutionary Testing

Seed Inputs → Input Generation Guidance → State Information → New Inputs

Runtime Monitoring

Application
Evolutionary Differential Testing - Multiple-Apps

What are the options to driving input generation?

1. Use program states solely from *single application*, like most modern fuzzers

2. Use *global* program states *combined* across all applications

3. Re-design guidance engine *geared towards differential testing*
Evolutionary Differential Testing - Multiple-Apps

What are the options for driving input generation?

1. Use program states solely from *single application*, like most modern fuzzers

2. Use *global* program states *combined* across all applications

3. Re-design guidance engine *geared towards differential testing*
Key Insight

Techniques that work well in the context of single application testing may not be optimal for differential testing!
Multi-App Code Coverage

Input Corpus

All possible code paths

Code coverage - Global

Code coverage - Input

Per-Input Coverage
Multi-App Code Coverage

Input Corpus

Per-Input Coverage

All possible code paths

Code coverage - Global

Code coverage - Input
Multi-App Code Coverage

App1

App2

Input 1

Input Corpus

Per-Input Coverage

All possible code paths

Code coverage - Global

Code coverage - Input
Multi-App Code Coverage

App1

App2

Input Corpus

Per-Input Coverage

Input 1

All possible code paths

Code coverage - Global

Code coverage - Input
Multi-App Code Coverage

Input Corpus

Per-Input Coverage

App1

App2

Input 1

All possible code paths

Code coverage - Global

Code coverage - Input
Multi-App Code Coverage

Input Corpus

Input 1

Per-Input Coverage

App1

App2

Input 2

All possible code paths

Code coverage - Global

Code coverage - Input
Multi-App Code Coverage

App1

Input Corpus

Input 1
Input 2

Per-Input Coverage

App2

All possible code paths

- Code coverage - Global
- Code coverage - Input
Multi-App Code Coverage

App1

App2

Input Corpus

Input 1

Input 2

Per-Input Coverage

Input 3

All possible code paths

Yellow Code coverage - Global

Gray Code coverage - Input
Multi-App Code Coverage

Input Corpus  Input 1  Input 2

Per-Input Coverage

All possible code paths
- Code coverage - Global
- Code coverage - Input

Discard!
Multi-App Code Coverage

Input Corpus

<table>
<thead>
<tr>
<th>Input 1</th>
<th>Input 2</th>
</tr>
</thead>
</table>

Per-Input Coverage

All possible code paths
- Code coverage - Global
- Code coverage - Input
Multi-App Code Coverage

Input Corpus: All possible code paths

<table>
<thead>
<tr>
<th>Input 1</th>
<th>Input 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Per-Input Coverage</td>
<td></td>
</tr>
</tbody>
</table>

App1

App2

Input 4: Discard!
Multi-App Code Coverage

App1

Input 3

App2

Input 3
Multi-App Code Coverage

App 1

App 2

Input 3

Input 4
Multi-App Code Coverage

- These inputs exercise disproportionate code regions in the two apps
- This disproportion might imply differences in handling logic
- Retaining them in corpus speed up process of finding discrepancies
Relative program behavior is important in this context!
δ-diversity: a new approach to guided differential testing
Differential Testing under $\delta$-diversity

• Obtain State Information
  – White-box (e.g., at compile time)
  – Gray-box (e.g., using Dynamic Binary Instrumentation)
  – Black-box (e.g., only examining system response to inputs)

• Behavioral Diversity
Differential Testing under $\delta$-diversity
Differential Testing under $\delta$-diversity
Differential Testing under $\delta$-diversity
Differential Testing under $\delta$-diversity

Behavioral Asymmetries

OpenSSL

LibreSSL

wolfSSL

GnuTLS
Differential Testing under $\delta$-diversity

Behavioral Asymmetries
Differential Testing under $\delta$-diversity

- Two examples:
  - Gray-box
  - Black-box
- Both outperform code coverage
Path $\delta$-diversity: gray-box

Keep track of **unique edges**
Path $\delta$-diversity: gray-box

Input 1 ✓

A_1 A_2 A_3
B_1

(3,1)

Input 2 ✓

A_1 A_2
B_1 B_2 B_3

(2,3)

Input 3 X

A_1 A_2 A_3
B_1 B_1

(3,1)

Keep track of unique edges
## Output \( \delta \)-diversity: black-box

<table>
<thead>
<tr>
<th>Input</th>
<th>App1</th>
<th>App2</th>
</tr>
</thead>
<tbody>
<tr>
<td>✓</td>
<td>0x0</td>
<td>0x0</td>
</tr>
<tr>
<td>✓</td>
<td>0x1</td>
<td>0xdead</td>
</tr>
<tr>
<td>✗</td>
<td>0x0</td>
<td>0x0</td>
</tr>
<tr>
<td>✓</td>
<td>0x0</td>
<td>0xbeef</td>
</tr>
<tr>
<td>✓</td>
<td>0x0</td>
<td>0xbeef</td>
</tr>
<tr>
<td>✓</td>
<td>0xbabe</td>
<td>0x0</td>
</tr>
<tr>
<td>✗</td>
<td>0x0</td>
<td>0xbeef</td>
</tr>
</tbody>
</table>

- All possible code paths
- Return values
- Error codes
- Exception messages
δ-diversity

- Domain Independence
- Efficient differential guidance
Implementation

- NEZHA prototype
- Gray-box and black-box $\delta$-diversity metrics
  - Path $\delta$-diversity (fine & coarse)
  - Output $\delta$-diversity
- Domain-independent input generation
  - Evolutionary, feedback-guided
- Built upon libFuzzer with NEZHA-specific hooks
- 1545 lines of C++
Use cases

- SSL libraries
  - LibreSSL
  - OpenSSL
  - wolfSSL
  - GnuTLS

- PDF readers
  - pdf
  - e

- ClamAV & XZ Parsers
Use cases

- SSL libraries
  - LibreSSL
  - OpenSSL
  - wolfSSL
  - GnuTLS

- PDF readers
  - PDF
  - e

- ClamAV & XZ Parsers
Certificate Verification Discrepancies

One library accepts one certificate, while another rejects it with an error code.

<table>
<thead>
<tr>
<th></th>
<th>LibreSSL</th>
<th>BoringSSL</th>
<th>wolfSSL</th>
<th>mbedTLS</th>
<th>GnuTLS</th>
</tr>
</thead>
<tbody>
<tr>
<td>OpenSSL</td>
<td>10</td>
<td>1</td>
<td>8</td>
<td>33</td>
<td>25</td>
</tr>
<tr>
<td>LibreSSL</td>
<td>-</td>
<td>11</td>
<td>8</td>
<td>19</td>
<td>19</td>
</tr>
<tr>
<td>BoringSSL</td>
<td>-</td>
<td>-</td>
<td>8</td>
<td>33</td>
<td>25</td>
</tr>
<tr>
<td>wolfSSL</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>6</td>
<td>8</td>
</tr>
<tr>
<td>mbedTLS</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>31</td>
</tr>
</tbody>
</table>

Unique pair-wise discrepancies (based on error code tuples)
NEZHA vs domain-specific frameworks

- 52x more discrepancies than Frankencerts
- 27x more discrepancies than Mucerts
**NEZHA vs popular evolutionary fuzzers**

- Adapted popular evolutionary fuzzers for differential testing
  - Code coverage in single application
  - Global code coverage
- 6x more discrepancies than testing on a single application
- 30% more discrepancies than modified libFuzzer
Sample Bugs uncovered by NEZHA
*(disclosed and patched)*
### Experimental Setting

<table>
<thead>
<tr>
<th>Application Category</th>
<th>Tests</th>
</tr>
</thead>
<tbody>
<tr>
<td>SSL Libraries</td>
<td>OpenSSL, LibreSSL, BoringSSL, GnuTLS, wolfSSL, mbedTLS</td>
</tr>
<tr>
<td>PDF Readers</td>
<td>Evince PDF, MuPDF, Xpdf</td>
</tr>
<tr>
<td>Parsers</td>
<td>ClamAV vs binutils, ClamAV vs xz</td>
</tr>
</tbody>
</table>
Bug 1: Malicious ELF can evade ClamAV detection

ClamAV (ELF parsing engine)
Bug 1: Malicious ELF can evade ClamAV detection

**ClamAV (ELF parsing engine)**

```c
static int cli_elf_fileheader(...) {

  switch(file_hdr->hdr64.e_ident[4]) {
    case 1:
      ...
    case 2:
      ...
    default:
      ...
    return CL_EFORMAT;
  }

  ...

```
Bug 1: Malicious ELF can evade ClamAV detection

ClamAV (ELF parsing engine)

```c
static int cli_elf_fileheader(...) {
    switch(file_hdr->hdr64.e_ident[4]) {
        case 1:
            ...
        case 2:
            ...
        default:
            ...
            return CL_EFORMAT;
    ...
```

Linux ELF loader

```c
static int load_elf_binary(struct linux_binprm *bprm) {
    ...
    retval = -ENOEXEC;
    if (memcmp(loc->elf_ex.e_ident, ELFMAG, SELFMAG) != 0)
        goto out;
    if (loc->elf_ex.e_type != ET_EXEC &&
        loc->elf_ex.e_type != ET_DYN)
        goto out;
    if (!elf_check_arch(&loc->elf_ex))
        goto out;
    ...
```
Bug 1: Malicious ELF can evade ClamAV detection

ClamAV (ELF parsing engine)

Linux ELF loader
**Bug 2: LibreSSL misinterprets time in ASN.1 format**

Time fields can be formatted in 2 ways:

- **UTC**: YYMMDDHHMMSSSZ (13 char long)
- **GMT**: YYYYMMDDHHMMSSSZ (15 char long)
Bug 2: LibreSSL misinterprets time in ASN.1 format

Time fields can be formatted in 2 ways:

**UTC**: YYMMDDHHMMSSSZ (13 char long)

**GMT**: YYYYMMDDHHMMSSSZ (15 char long)

LibreSSL ignores the ASN.1 time format tag, and determines format based on length of field
Bug 2: LibreSSL misinterprets time in ASN.1 format

LibreSSL ignores the ASN.1 time format tag, and determines format based on length of field
Bug 2: LibreSSL misinterprets time in ASN.1 format

LibreSSL ignores the ASN.1 time format tag, and determines format based on length of field.
Bug 2: LibreSSL misinterprets time in ASN.1 format

LibreSSL ignores the ASN.1 time format tag, and determines format based on length of field

Jan 1 01:01:00 2012 GMT can interpreted as Dec 1 01:01:01 2020 GMT
Conclusions

- δ-diversity outperforms code coverage for differential testing
- NEZHA: Domain independent, efficient differential testing
- Differential testing should be integrated, when possible, into the testing cycle

https://github.com/nezha-dt
Backup Slides
**NEZHA: Architecture**

Diagram showing the architecture of NEZHA, including:
- Instruments Module
- Programs (Original)
- Initial Seeds
- Input Corpora
- Application Address Space
- NEZHA Runtime Library
  - Dynamic Coverage Information
  - Program Return Values
- NEZHA Engine
  - Differential Execution
  - Input Mutation
  - Corpus Refinement
  - Discrepancy Logging
  - Guidance Engines
#include <openssl/evp.h>

extern "C"
int LLVMFuzzerTestOneInput(const uint8_t *buf, size_t len) {
    const uint8_t *bufp = buf;
    EVP_PKEY_free(d2i_AutoPrivateKey(NULL, &bufp, len));
    return 0;
}

c++ -c -g -O2 -std=c++11 Fuzzer/*.cpp -IFuzzer
ar rv libFuzzer.a Fuzzer*.o
NEZHA: Architecture

Engine

1. Fuzz_TestStart
2. Process_i (Data)
3. LLVMTestOneInput
4. Fuzz_TestEnd
5. LLVMFuzzerRetVals
6. LLVMFuzzerCovBuffers
7. RunOne
8. UpdateDiff

libFuzzer backend
Components added

Input corpus

Tested Applications
Discrepancy Distribution for SSL/TLS Libs

Same Inputs / Different mode SSL libraries tested

- Output $\delta$-diversity: 4
- Path $\delta$-diversity: 348
- Global Coverage: 143
- 26