Do Not Blame Users for Misconfigurations

Tianyin Xu, Jiaqi Zhang, Ryan Huang, Jing Zheng, Tianwei Sheng, Ding Yuan, Yuanyuan Zhou, Shankar Pasupathy*

UC San Diego          *NetApp
How many of you have made mistakes when configuring systems?
You Are Not Alone!

Root causes of “high-severity” customer issues in a major storage company [Yin et al, SOSP’11]
How many of you think your misconfig. was your fault?
Unfortunately, many developers think they are users’ faults!

“It is not a bug, but an invalid setting.”

Developers of a mature open-source server app.
Configuration Is a User Interface!

datadir = /var/lib/mysql
tmpdir = /tmp
lc-messages-dir = /usr/share/mysql
skip-external-locking
#
# Instead of skip-networking the default is now to listen only on
# localhost which is more compatible and is not less secure.
bind-address = 127.0.0.1
#
# * Fine Tuning
#
key_buffer = 16M
max_allowed_packet = 16M
thread_stack = 192K
thread_cache_size = 8
# This replaces the startup script and checks MyISAM tables if needed
# the first time they are touched
myisam-recover = BACKUP
#max_connections = 100
#table_cache = 64
#thread_concurrency = 10
#
# * Query Cache Configuration
#
query_cache_limit = 1M
query_cache_size = 16M
#
# * Logging and Replication
Goal #1: React Gracefully to Misconfig.

- Today’s systems are **vulnerable** to misconfig.

<table>
<thead>
<tr>
<th>Software Systems</th>
<th>Crashes &amp; Hangs w/o Message</th>
</tr>
</thead>
<tbody>
<tr>
<td>Storage-A</td>
<td>8.4%</td>
</tr>
<tr>
<td>CentOS</td>
<td>6.7%</td>
</tr>
<tr>
<td>MySQL</td>
<td>16.4%</td>
</tr>
<tr>
<td>Apache</td>
<td>5.0%</td>
</tr>
<tr>
<td>OpenLDAP</td>
<td>4.8%</td>
</tr>
</tbody>
</table>

The impact distribution of misconfigurations [Yin et al, SOSP’11]
Goal #2: Intuitive & Less Error-prone

/* A Commercial Storage System*/

InitiatorName = iqn_DEV_domain

Error!

Lower-case only

Error-prone constraint 😞

Several customers made the same mistakes.
Our Contributions

1. **Spex**: automatically infer config. constraints by statically analyzing source code (for developers)

2. Use cases
   - Expose misconfig. vulnerabilities
   - Detect error-prone config. design & handling

3. Improve config. design of real-world systems
   - 1 commercial and 6 open-source systems
   - Expose 743 vulnerabilities (364 confirmed/fixed)
   - Detect 112 error-prone constraints (80 fixed)

4. Experience in interacting with developers
   - Improve Squid’s config. lib (benefit 150+ parameters)
Spex Overview

- Source Code
- Mapping
- Inference
- Config Constraints

# Config file
Max_threads=50
...=...
...=...

# Source code
if (ThreadNum<100) {
    ...
}
Spex Overview

Source Code

Mapping

Inference

Config Constraints

var x

var y
Mapping Is Non-trivial

- Cannot ask developers to annotate every parameter
- Investigated 18 software projects, all but one use one of the following mapping conventions.

<table>
<thead>
<tr>
<th>Mapping convention</th>
<th>What need to be annotated?</th>
<th># Software projects</th>
</tr>
</thead>
<tbody>
<tr>
<td>Structure-based</td>
<td>Data structure(s)</td>
<td>9</td>
</tr>
<tr>
<td>Comparison-based</td>
<td>Parsing function</td>
<td>4</td>
</tr>
<tr>
<td>Container-based</td>
<td>Getter functions</td>
<td>4</td>
</tr>
</tbody>
</table>
Structure-based Mapping

- PostgreSQL-9.2.1

```c
struct config_int
{ConfigureNamesInt[] =
{
    {"deadlock_timeout", ...
    &DeadlockTimeout, ...
},
...
    {"max_connections", ...
    &MaxConnections, ...
},
...
} 80 more mappings
```

Annotation

@STRUCT =ConfigureNamesInt
@PAR = [config_int, 1]
@VAR = [config_int, 3]

deadlock_timeout = 10
What Constraints Can Be Inferred?

1. Data type
   - e.g., integer, float, string, boolean
   - file path, IP address, port

2. Data range
   - e.g., [10, 100], {'yes', 'no'}

3. Control dependency
   - e.g., X dominates Y’s executions

4. Value relationship
   - e.g., X < Y
Data Type Inference

• Methodology
  – Check the variable’s data type and how the variable is used in syscall/libcall

```c
int ft_init_stopwords(...) {
    fd = my_open(ft_stopword_file, ...);
    ...
}
```

```c
File my_open(const char * FileName, ...) {
    ...
    fd = open((char*) FileName, Flags);
}
```

/* MySQL-5.5.29 */

# Config parameter "ft_stopword_file"

A file path
Data Range Inference

• Methodology
  – If the variable is compared with a constant value, inspect the branch block to decide the range.

```c
static int config_generic (...)
{
    ...
    if(c->value_int < 4)
        c->value_int = 4;
    else if(c->value_int > 255)
        c->value_int = 255;
    ...
}
/* OpenLDAP-2.4.33 */
```

# Config parameter “index_intlen”

Data range: [4, 255]
Control Dependency Inference

- Methodology
  - Check if the config variable’s usage is controlled by another config variable

```c
static TransactionId RecordTransactionCommit() {
    ...
    if(enableFsync && MinimumActiveBackends(CommitSiblings)){
        ...
    }
} /* PostgreSQL-9.2.1 */
```

*All commit_siblings’s use sites are inside the func call.*

# Config parameter:

“fsync”

“commit_siblings”

“commit_siblings” takes effect iff “fsync” is enabled

• Methodology
- Check if the config variable’s usage is controlled by another config variable
Use Case of Constraints #1

- Expose misconfig. vulnerabilities
  - Misconfig. injection testing

<table>
<thead>
<tr>
<th>Type</th>
<th>Constraint</th>
<th>Config. Error</th>
</tr>
</thead>
<tbody>
<tr>
<td>Basic type</td>
<td>A is an integer</td>
<td>A := 2XX</td>
</tr>
<tr>
<td>Semantic type</td>
<td>B is a file path</td>
<td>B := invalid path</td>
</tr>
<tr>
<td>Data range</td>
<td>C ∈ [10, 100]</td>
<td>C := 1000</td>
</tr>
<tr>
<td>Ctrl dep.</td>
<td>D depends on E</td>
<td>D := yes, E := no</td>
</tr>
<tr>
<td>Value rel.</td>
<td>F &gt; H</td>
<td>F &lt; H</td>
</tr>
</tbody>
</table>

Implemented as a plugin framework (easy to extend)
Program received signal SIGSEGV, Segmentation fault.
my_mb_cctype_8bit (cs=0x1226760, ctype=0x7fffffffde00, s=0x1ad5000 <Address 0x1ad5000 out of bounds>, e=0x10185a67f <Address 0x10185a67f out of bounds>) at ./strings/ctype-simple.c:1299 1299 *ctype=cs->ctype[*s + 1];
......
Use Case of Constraints #2

- Detect error-prone config. design & handling
  - Design inconsistency (case sensitivity, unit)
  - Silent overruling
  - Undocumented constraints
# Software Evaluated

<table>
<thead>
<tr>
<th>Software</th>
<th>Proprietary</th>
<th>LOC</th>
<th># Parameters</th>
<th>LOA*</th>
</tr>
</thead>
<tbody>
<tr>
<td>Storage-A</td>
<td>Commercial</td>
<td>--</td>
<td>&gt; 1000</td>
<td>5</td>
</tr>
<tr>
<td>Apache</td>
<td>Open source</td>
<td>148K</td>
<td>103</td>
<td>4</td>
</tr>
<tr>
<td>MySQL</td>
<td>Open source</td>
<td>1.2M</td>
<td>272</td>
<td>29</td>
</tr>
<tr>
<td>PostgreSQL</td>
<td>Open source</td>
<td>757K</td>
<td>231</td>
<td>7</td>
</tr>
<tr>
<td>OpenLDAP</td>
<td>Open source</td>
<td>292K</td>
<td>86</td>
<td>4</td>
</tr>
<tr>
<td>VSFTP</td>
<td>Open source</td>
<td>16K</td>
<td>124</td>
<td>5</td>
</tr>
<tr>
<td>Squid</td>
<td>Open source</td>
<td>180K</td>
<td>335</td>
<td>2</td>
</tr>
</tbody>
</table>

*LOA: lines of annotation
## Exposed Misconfig. Vulnerabilities

<table>
<thead>
<tr>
<th>Software</th>
<th>Crash/Hang</th>
<th>Early Termina.</th>
<th>Function Failure</th>
<th>Silent Violation</th>
<th>Silent Ignor.</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>Storage-A</td>
<td>0</td>
<td>0</td>
<td>7</td>
<td>74</td>
<td>83</td>
<td>164</td>
</tr>
<tr>
<td>Apache</td>
<td>5</td>
<td>4</td>
<td>9</td>
<td>29</td>
<td>5</td>
<td>52</td>
</tr>
<tr>
<td>MySQL</td>
<td>5</td>
<td>10</td>
<td>12</td>
<td>71</td>
<td>16</td>
<td>114</td>
</tr>
<tr>
<td>PostgreSQL</td>
<td>1</td>
<td>10</td>
<td>2</td>
<td>1</td>
<td>35</td>
<td>49</td>
</tr>
<tr>
<td>OpenLDAP</td>
<td>1</td>
<td>3</td>
<td>6</td>
<td>7</td>
<td>0</td>
<td>17</td>
</tr>
<tr>
<td>VSFTPD</td>
<td>12</td>
<td>5</td>
<td>18</td>
<td>23</td>
<td>68</td>
<td>126</td>
</tr>
<tr>
<td>Squid</td>
<td>2</td>
<td>3</td>
<td>29</td>
<td>173</td>
<td>14</td>
<td>221</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td><strong>26</strong></td>
<td><strong>35</strong></td>
<td><strong>83</strong></td>
<td><strong>378</strong></td>
<td><strong>221</strong></td>
<td><strong>743</strong></td>
</tr>
</tbody>
</table>
## Detected Inconsistency

<table>
<thead>
<tr>
<th>Software</th>
<th>Case Sensitivity</th>
<th>Fixed parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Sensitive</td>
<td>Insensitive</td>
</tr>
<tr>
<td>Storage-A</td>
<td>32 (7.1%)</td>
<td>453 (92.3%)</td>
</tr>
<tr>
<td>Apache</td>
<td>3 (11.5%)</td>
<td>26 (88.5%)</td>
</tr>
<tr>
<td>MySQL</td>
<td>1 (1.7%)</td>
<td>58 (98.3%)</td>
</tr>
<tr>
<td>PostgreSQL</td>
<td>0 (0%)</td>
<td>92 (100%)</td>
</tr>
<tr>
<td>OpenLADP</td>
<td>0 (0%)</td>
<td>9 (100%)</td>
</tr>
<tr>
<td>VSFTP</td>
<td>0 (0%)</td>
<td>73 (100%)</td>
</tr>
<tr>
<td>Squid</td>
<td>85 (52.8%)</td>
<td>76 (47.2%)</td>
</tr>
<tr>
<td>Total</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
### Can We Help Real-world Misconfig.? 

<table>
<thead>
<tr>
<th>Software</th>
<th>Real-world misconfig.</th>
<th>Bad reactions that can be potentially avoided</th>
</tr>
</thead>
<tbody>
<tr>
<td>Storage-A</td>
<td>246</td>
<td>68 (27.6%)</td>
</tr>
<tr>
<td>Apache</td>
<td>50</td>
<td>19 (38.0%)</td>
</tr>
<tr>
<td>MySQL</td>
<td>47</td>
<td>14 (29.8%)</td>
</tr>
<tr>
<td>OpenLDAP</td>
<td>49</td>
<td>12 (24.5%)</td>
</tr>
</tbody>
</table>
## Inference Accuracy

<table>
<thead>
<tr>
<th>Software</th>
<th>Basic Type</th>
<th>Semantic Type</th>
<th>Data Range</th>
<th>Control Dep.</th>
<th>Value Dep.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Storage-A</td>
<td>97.0%</td>
<td>95.7%</td>
<td>87.1%</td>
<td>84.1%</td>
<td>94.7%</td>
</tr>
<tr>
<td>Apache</td>
<td>96.1%</td>
<td></td>
<td></td>
<td>100.0%</td>
<td>81.8%</td>
</tr>
<tr>
<td>MySQL</td>
<td>100.0%</td>
<td></td>
<td></td>
<td>94.7%</td>
<td>71.4%</td>
</tr>
<tr>
<td>PostgreSQL</td>
<td>100.0%</td>
<td></td>
<td></td>
<td>91.7%</td>
<td>85.7%</td>
</tr>
<tr>
<td>OpenLDAP</td>
<td>88.2%</td>
<td>93.7%</td>
<td>73.1%</td>
<td>N/A</td>
<td>50.0%</td>
</tr>
<tr>
<td>VSFTP</td>
<td>100.0%</td>
<td>100.0%</td>
<td>100.0%</td>
<td>63.9%</td>
<td>100.0%</td>
</tr>
<tr>
<td>Squid</td>
<td>77.0%</td>
<td>100.0%</td>
<td>100.0%</td>
<td>77.8%</td>
<td>100.0%</td>
</tr>
</tbody>
</table>

**Avg. 90.6%**
Experience (Positive ☺)

• **Storage-A:**
  – Slides sent to all the developers

• **Squid:**
  – Improve the config. lib. (150 parameters benefit)

• 364 detected misconfig. vulnerabilities have been confirmed or fixed by developers.

• 80 detected error-prone constraints have been fixed by developers.
“It is not a bug, but an invalid setting.”

“Those who do (configuration) typically read the code.”

“If you work exactly and carefully it does not matter; if not, you should not maintain a server at all.”
Limitations

1. We cannot infer all the constraints, e.g., domain-specific, cross-software

2. The inference is not 100% accurate

3. More fundamental approach is to rethink and redesign of configuration
Related Work

• Detection and Diagnosis
  - Detection: [Feamster NSDI’05], [Yuan USENIX’11],
  - Diagnosis: [Wang OSDI’04], [Witaker OSDI’04],
    [Attariyan OSDI’10], [Attariyan OSDI’12]

• Testing system resilience to config errors
  - Mutation testing: [Keller DSN’08]

• Extract source code information for config
  - Type information: [Rabkin ICSE’11]
Conclusions

- Take a more active role in handling misconfig.
  - Configuration is a user interface!

- Spex: a tool that automatically infers config. constraints from source code.
  - Exposed 741 vulnerabilities (364 confirmed/fixed)
  - Detected 112 error-prone constraints (80 fixed)