Place Your Locks Well: Understanding and Detecting Lock Misuse Bugs

Yuandao Cai, Peisen Yao, Chengfeng Ye, Charles Zhang
The Hong Kong University of Science and Technology
Increasingly Complex Modern Software

- Massive codebases and high concurrency

<table>
<thead>
<tr>
<th>Software</th>
<th>MLoC</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mac OS X &quot;Tiger&quot;</td>
<td>85</td>
</tr>
<tr>
<td>Facebook</td>
<td>61</td>
</tr>
<tr>
<td>Microsoft Visual Studio 2012</td>
<td>50</td>
</tr>
<tr>
<td>Windows Vista</td>
<td>50</td>
</tr>
<tr>
<td>Microsoft Office 2013</td>
<td>44</td>
</tr>
<tr>
<td>Firefox</td>
<td>11</td>
</tr>
<tr>
<td>Google Chrome</td>
<td>8</td>
</tr>
<tr>
<td>Photoshop C.S. 6</td>
<td>7</td>
</tr>
<tr>
<td>Linux Kernel 2.2.0</td>
<td>4</td>
</tr>
</tbody>
</table>

MLoC (Millions of Lines Of Code) 0 20 40 60 80 100

- Poor software quality costs the US economy $2.41 trillion annually
Concurrency bugs are hard-to-avoid and extremely harmful!

Synchronization primitives are in place to synchronize concurrent code
  • preventing various concurrency bugs and vulnerabilities

Existing works focus on concurrency bug detection (insufficient synchronization)
  • data races[AI Thokair POPL’23] [chabbi et.al., PLDI’22]…
  • concurrency memory corruption bugs [Yuan et.al., Security’23] [Cai et.al., PLDI’21]…
  • concurrency typestate bugs [ASPLOS’11]…

Our work focuses on the misuses of synchronization APIs themselves
  • currently focusing on locks
  • also causing serious reliability and security issues
Research Goal and Contributions

1. Understanding the common misuses of locks
   • through a CVE-ID-based empirical study

2. Designing techniques to detect the lock misuses

3. Evaluating and advancing the state-of-the-art bug-finding tools
An Empirical Study: Setup

- Locks are common synchronization primitives
  - with explicit disciplines for initialization, use, and destruction.
- Study Question 1: What are the common lock misuses?
- Study Question 2: What are the common causes of those lock misuses?

- Study Dataset: 32 CVE IDs assigned between 2010-2021
  - search keywords: e.g., mutex, lock
  - manual validation for CVE ID description
An Empirical Study: Finding I

1. **Identifying** five general locking discipline violations
   - under both sequential and concurrent circumstances
   - covering a single thread and multiple threads

2. **Defining** the bug patterns by revealing their characteristics

<table>
<thead>
<tr>
<th>No.</th>
<th>Misuse Pattern</th>
<th>Bug Description</th>
<th>Concurrency</th>
</tr>
</thead>
<tbody>
<tr>
<td>①</td>
<td>Missing lock releases</td>
<td>A lock is not released after its effective lifetime.</td>
<td></td>
</tr>
<tr>
<td>②</td>
<td>Double locking</td>
<td>A lock is acquired twice.</td>
<td></td>
</tr>
<tr>
<td>③</td>
<td>Using uninitialized locks</td>
<td>A lock is not initialized before using it. A concurrency error occurs when the lock is initialized non-deterministically.</td>
<td>✓</td>
</tr>
<tr>
<td>④</td>
<td>Releasing unacquired locks</td>
<td>A lock is released without acquiring it first. A concurrency error occurs when there is another thread holding the lock.</td>
<td>✓</td>
</tr>
<tr>
<td>⑥</td>
<td>Cyclic lock acquisitions</td>
<td>Different locks are not acquired in the same order. A concurrency error occurs when each thread in a set waits for the other to release a lock.</td>
<td>✓</td>
</tr>
</tbody>
</table>
An Empirical Study: Five General Lock Misuses

1. Missing lock releases (OpenSSL)

2. Double locking (Cen64)

3. Using uninitialized locks (Cherokee)

4. Releasing unacquired locks (OpenSSL)

5. Cyclic lock acquisitions (MariaDB)
An Empirical Study: Finding II

• Wreaking severe havoc by triggering lock misuses
  • denial-of-service with system hang (concurrent cyclic acquisitions, double locking)
    • CVE-2013-4553, CVE-2014-8131, CVE-2019-14763, CVE-2021-41213,…
  • memory exhaustion with memory leak (missing lock releases)
    • CVE-2004-2650, CVE-2018-14660, CVE-2020-12658,…
  • memory corruption; system crash (releasing unacquired locks, using uninitialized locks)
    • CVE-2014-1453, CVE-2015-8767, CVE-2017-6353, CVE-2020-10573,…
  • even privilege escalation and other unidentified issues
    • CVE-2010-4210, CVE2014-9748, …

• Relating to other security bugs
  • atomicity violations (CVE-2020-10573)
  • use-after-free (CVE-2019-14034)
  • double free (CVE-2017-6353)

```c
98 int search_makelist(search_t *results,...){
145   pthread_mutex_unlock(&conn->lock);
146   int tmp = conn_setup(conn);
147   pthread_mutex_unlock(&conn->lock);
203 }
```

Releasing unacquired locks that leads to atomicity violations (Axel)
Detecting the Five Lock Misuses with Lockpick

• **Lock misuse formulation:** characterizing lock misuses with a *finite-state machine (FSM)*

1. Model the states of lock objects using typestates
2. Capture the state transitions of lock objects with a new FSM
3. Capture the lock misuses by tracking the state transitions

• **Lock misuse detection:** detecting lock misuses with *several customized techniques*

1. Path-sensitively track the typestates of locks
2. Reason about the MHP relations of statements
3. Flag the lock misuses *based on typestate violations*
Implementation and Experiment Setup

• Lockpick is built upon the LLVM infrastructure and the Z3 SMT solver
  • a soundy implementation to reach both high efficiency and precision
    • unrolling loops twice, ignoring inline assembly, pointer arithmetic
  • a value-flow-based pointer analysis
    • on-demand flow-, context-sensitive pointer analysis
  • path conditions are encoded as first-order logic formulae over bit-vectors

• Question 1: How effective and practical is Lockpick at uncovering lock misuses in mature open-source software systems?
• Question 2: How does Lockpick perform compared to the state-of-the-art tools?
(1) Highlights: Effectiveness on Bug Finding

- Finding **203 developer-confirmed bugs** in over 80 well-checked software programs
  - **184** of them have been fixed (at the time of publication)
  - finding various kinds of bugs
  - hiding for an average of **7.4 years**

The distributions of bug type

- **①** Missing lock releases
- **②** Double locking
- **③** Using uninitialized locks
- **④** Releasing unacquired locks
- **⑤** Cyclic Lock Acquisition

The distributions of hidden time (Year)

- **Y≤1**
- **1<Y≤5**
- **5<Y≤10**
- **Y>10**

- **16 CVE IDs** have been assigned for multiple bugs with high security impacts
(2) Highlights: Advancement over Previous Tools

- **Baselines:** SVF, L2D2 (built on Infer), Clang Static Analyzer
- **Benchmarks:** ten popular software programs with 35.8 MLoC
- **Efficiency:** being able to analyze big programs like Linux kernel in about five hours
- **Precision:** embracing better precision than other tools
- **Recall:** being able to discover 26 past CVE IDs in C/C++ programs (2010-2021)
  - other tools cannot reach

---

![Graph showing time (s) vs. number of vulnerabilities found for different projects](image)

<table>
<thead>
<tr>
<th>Project</th>
<th>KLoC</th>
<th>SVF</th>
<th>L2D2</th>
<th>CSA</th>
<th>LOCKPICK</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>#FP</td>
<td>#R</td>
<td>#FP</td>
<td>#R</td>
<td>#FP</td>
</tr>
<tr>
<td>Cherokee</td>
<td>55</td>
<td>3</td>
<td>6</td>
<td>99%</td>
<td>483</td>
</tr>
<tr>
<td>Curl</td>
<td>135</td>
<td>3</td>
<td>4</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>PJSP</td>
<td>434</td>
<td>32</td>
<td>43</td>
<td>98%</td>
<td>505</td>
</tr>
<tr>
<td>OpenSIPS</td>
<td>477</td>
<td>25</td>
<td>55</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>OpenSSL</td>
<td>490</td>
<td>66</td>
<td>68</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>WolfSSL</td>
<td>944</td>
<td>16</td>
<td>20</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>MySQL</td>
<td>4,152</td>
<td>0</td>
<td>0*</td>
<td>100%</td>
<td>1,157</td>
</tr>
<tr>
<td>MariaDB</td>
<td>4,697</td>
<td>96%</td>
<td>141*</td>
<td>100%</td>
<td>4,993</td>
</tr>
<tr>
<td>FreeBSD</td>
<td>8,457</td>
<td>66</td>
<td>81</td>
<td>NA</td>
<td>NA</td>
</tr>
<tr>
<td>Linux</td>
<td>14,987</td>
<td>88%</td>
<td>328*</td>
<td>NA</td>
<td>NA</td>
</tr>
</tbody>
</table>

FPR: 85.1%, 99.2%, 93.7%, 27.5%
Thank you for your listening!

Questions & Answers

More details can be found in our paper:

Bug and CVE ID lists can be found:
https://drive.google.com/file/d/1HY7PydeDga-850ZOn3YPACnX7hRws8DG/view