AURC: Detecting Errors in Program Code and Documentation

Peiwei Hu, Ruigang Liang, Ying Cao, Kai Chen, and Runze Zhang

SKLOIS, Institute of Information Engineering, CAS, China
School of Cyber Security, University of Chinese Academy of Sciences, China
Beijing Academy of Artificial Intelligence, China
1. Background

Open-source libraries greatly affect the security of downstream software.

Affect 1800+ products

- Log4Shell
- CVE-2021-44228
2. Previous Studies

How to detect the bugs in libraries:

- Based on the usage extracted from documents
- Based on majority voting
- Based on the behavior in the similar context
2. Previous Studies

How to detect the bugs in libraries:

- Based on the usage extracted from documents
- Based on majority voting
- Based on the behavior in the similar context

- Documents are not trustworthy.
- The majority voting is not trustworthy.
- The usage in the similar context is not trustworthy.
3.1 Our Solution

How experienced programmers find the usage facing the unfamiliar API

- Read documents
- Scan the source code of API
- Figure out how the existing code uses the API

API Usage Reference (AUR)

- Document
- Callee
- Caller
3.1 Our Solution

API Usage Reference (AUR)

- Document
- Pre-trained Classifier
- Mapping Table
- Correctness Inference
- Fix code or documents
- Detection

- Caller
- CBP
- RDT
- CoPS
- Construction

Focusing on Incorrect Return Check

```c
int BN_bn2binpad(const BIGNUM *a, unsigned char *to, int tolen) {
    if(tolen < 0)
        return -1;
    return BN2binpad(a, to, tolen, big);
}
```

`BN_bn2binpad()` returns the number of bytes written or -1 if the supplied buffer is too small.

- caller
- case1 if(BN_bn2binpad(r1, rbuf, r1_len) <= 0)
- case2 j = BN_bn2binpad(ret, buf, num);
  if (j < 0) {
      goto err;
  }
- case3 if(!BN_bn2binpad(bn, bnbuf, sz))
- case4 if(BN_bn2binpad(s1, sbuf, s1_len) <= 0)

......

**Rule 1:** documents and callees are consistent, fix callers
3.2 Analysis of Callee

**Challenge:**
1. Long call chain
2. Return statement location

- **heavy analysis**

```
int cms_main()
{
  /* ... 946 lines of code ... */
  ret = SMIME_write_CMS(out, cms);
  if (ret <= 0) {
    ret = 6;
    goto end;
  }
  ret = 0;
  end:
  /* ret equals to 0 or 6 */
  return ret;
}
```

- **Topological sorting**
  D E B C A

- **Context-sensitive Backtrace Prediction**
3.2 Analysis of Documents

**Challenge:** 1. Contain sentences that are irrelevant to the return values. 2. Documents may describe the return values in natural language.

extract return values from documents

### Mapping

<table>
<thead>
<tr>
<th>Word</th>
<th>Range</th>
</tr>
</thead>
<tbody>
<tr>
<td>nonzero</td>
<td>(-, 0) ∪ (0, +)</td>
</tr>
<tr>
<td>zero</td>
<td>0</td>
</tr>
<tr>
<td>length, size</td>
<td>(0, +)</td>
</tr>
<tr>
<td>negative</td>
<td>(-, 0)</td>
</tr>
</tbody>
</table>
3.3 AUR should be modified

Challenge: Which AUR should be modified facing inconsistency?

Table 3: Rules of Correctness Inference

<table>
<thead>
<tr>
<th>Consistency Check</th>
<th>Modified Subject</th>
</tr>
</thead>
<tbody>
<tr>
<td>Caller</td>
<td>Callee</td>
</tr>
<tr>
<td>✗</td>
<td>✓</td>
</tr>
<tr>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>✗</td>
<td>✓</td>
</tr>
<tr>
<td>Others</td>
<td>Manual Check</td>
</tr>
</tbody>
</table>
4 Experiments – Effectiveness

<table>
<thead>
<tr>
<th>Codebase</th>
<th>Inconsistency Detection</th>
<th>Inconsistency Type</th>
<th>Running Time (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Report</td>
<td>Code/True</td>
<td>Doc/True</td>
</tr>
<tr>
<td>OpenSSL</td>
<td>534</td>
<td>424/403</td>
<td>110/83</td>
</tr>
<tr>
<td>libzip</td>
<td>2</td>
<td>2/2</td>
<td>0/0</td>
</tr>
<tr>
<td>libwebsockets</td>
<td>8</td>
<td>0/0</td>
<td>8/8</td>
</tr>
<tr>
<td>GnuTLS</td>
<td>35</td>
<td>22/22</td>
<td>13/8</td>
</tr>
<tr>
<td>curl</td>
<td>2</td>
<td>2/2</td>
<td>0/0</td>
</tr>
<tr>
<td>mpg123</td>
<td>7</td>
<td>5/5</td>
<td>2/2</td>
</tr>
<tr>
<td>httpd</td>
<td>20</td>
<td>0/0</td>
<td>20/16</td>
</tr>
<tr>
<td>libgit2</td>
<td>129</td>
<td>46/37</td>
<td>83/73</td>
</tr>
<tr>
<td>libxml2</td>
<td>106</td>
<td>60/51</td>
<td>46/29</td>
</tr>
<tr>
<td>net-snmp</td>
<td>14</td>
<td>9/7</td>
<td>5/5</td>
</tr>
<tr>
<td>Average</td>
<td>–</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>All</td>
<td>857</td>
<td>570/529</td>
<td>287/224</td>
</tr>
</tbody>
</table>

False Positive: 12.1%
False Negative: 9.1%

529 code bugs 224 document defects
4 Experiments – Components

The average accuracy and recall are 95.5% and 94.3%, respectively.

<table>
<thead>
<tr>
<th>Group</th>
<th>Acc</th>
<th>Recall</th>
<th>Acc</th>
<th>Recall</th>
<th>Acc</th>
<th>Recall</th>
</tr>
</thead>
<tbody>
<tr>
<td>Group1</td>
<td>99.5%</td>
<td>99.2%</td>
<td>89.5%</td>
<td>82.3%</td>
<td>95.2%</td>
<td>91.8%</td>
</tr>
<tr>
<td>Group2</td>
<td>90.8%</td>
<td>98.8%</td>
<td>99.9%</td>
<td>99.8%</td>
<td>94.8%</td>
<td>94.8%</td>
</tr>
<tr>
<td>Group3</td>
<td>97.4%</td>
<td>96.1%</td>
<td>92.5%</td>
<td>86.0%</td>
<td>99.9%</td>
<td>100%</td>
</tr>
</tbody>
</table>

**Performance of Classifier**

\[
\text{Cov} = \frac{\text{LC}(\text{return statement}) - \text{LC}(\text{CBP finishes})}{\text{LC}(\text{return statement})}
\]

**PathRate**

\[
\text{PathRate} = \frac{\text{NumberOfPaths}(\text{replacement})}{\text{NumberOfPaths}(\text{no replacement})}
\]

= 12%  
88% of code does not need to be analyzed

= 0.06%  
CBP can save the analysis of 99.94% paths
5 Practical Effectiveness

- 236 code patches and 103 document patches have been merged by maintainers.

- We rank 25/810 in OpenSSL contributors with the help of findings of AURC.

- We get positive feedback from maintainers.
6 Summary

We propose a new method to detect defects in **both code and documents**. (cross-check three AURs)

We propose techniques to extract information from AURs including documents, callees, callers.

We test our prototype on real-world codebases. 236 code patches and 103 document patches have been merged by maintainers.
AURC: Detecting Errors in Program Code and Documentation

Thanks for your attention!