EA-CRT: An Error Analysis Method with Embedded Code Reverse Technology

LIU Jinshuo, CHEN Yusen
Computer School, Wuhan University,
Wuhan, 430072, China
E-mail: liujinshuo@whu.edu.cn

ZHANG Weixin†, XU Xiangyang, YAN Jingjing,
WANG Qingjiang
2Tianjin Electric Power Research Institute,
Tianjin, 300000, China
†Correspondence author: zhangweixin_cn@126.com

Abstract: The study of error analysis with embedded code reverse technology needs integration of decompiling, error module depict and comparison. This paper proposes an error analysis method for testing embedded code, and utilizes the smart meter’s fault, white and black screen, as the example. Our method has been validated 12 times on Renesas chips of the smart meter of Wasion Group, with the Open Source Code. The experiment shows that the proposed method is 87% effective for the fault model of black and white.

Keywords—Code Reverse Engineering; EA-CRT; model based Test

I. INTRODUCTION

As an important branch of reverse engineering, code reverse engineering [1] has long been a hot issue in the software engineering field. It targets binary code of ultimate program and takes conversion from machine code to high-level language as a core mission. Furthermore, two kinds of key technologies—disassemble and decompile are involved.

Error analysis is the challenge research of computer science. Error Analysis for code of embedded system is even more important and harder. Another challenging difficulty is that for the reason of patent, most of time, there is no source code of the high level program, but only the machine code in the MCU. Different MCU of the embedded system has the different instruction set. There are more complicated reasons for error analysis for embedded system.

The open source code CppCheck in a tool of static software analysis, which can analyze not the syntax error but the high level error, comparing with the other static software analysis tool. CppCheck has very good expansibility, using the open regular expression library PCRE. PCRE implements the detection of defects in lexical analysis phase, and then according to the value of the linener object Token to achieve the scalability of the defect mode [6].

This paper presents an implementation scheme of EA-CRT. Taking the software of the smart meters as an example. The experimental results also show superiority of our method with code reverse technology.

II. RELATED WORK

Disassembly research concentrates on algorithm improvements at home and abroad, such as static disassembly combined with machine learning [3], error detection via control flow [4], mixed disassembly method [5], disassembly based on speculation [6], etc.

For the research of the reliability of the software for the smart meter, Zhejiang Electric Power Research Institute [7] utilize cross test strategy to test the software. Qi weiwei of Huangshan power supply company Measurement Center of state grid [8] focus on testing the full performance of the smart meter, especially the function test, not the software. Alabdulkerim L [9] test the smart meter mainly on the (1) recognizing the software identification number; (2) the correctness of the algorithm and function; (3) the security of the software and data; (4) the security of the attributes; (5) the recognition of the faults and long protection. Yeli [10] of Hubei Electric Power Test Research Institute utilize assess the smart meter by combination of operating mode and communication protocol.

This paper presents an implementation scheme of EA-CRT. Taking the software of the smart meters as an example. The experimental results also show superiority of our method with code reverse technology.

III. EA-CRT

Figure 1 presents main infrastructure of this procedure. The assembly instruction block is generated as the output file and it varies according to the structure of EA-CRT. Disassembler takes machine code section as input, and knowledge module contains configuration information about embedded OS and related instruction set. [11-13]

Decompiler analyzes syntax notations and semantic structure of target block. Code structure analysis technology [11,12] is adopted to generate control flow graph (CFG), data flow graph (DFG) and function call graph (FCG). In conclusion, high-level program segment is organized by
A. Disassembler

As shown in figure 2, this disassembler regards machine code section as input, treats assembly instruction block as output and performs following steps:

Pre-processing of machine code section includes machine code endian adjustment [13], instruction and data blocks separation and target section normalization. Afterwards a target program section which conforms to the next input format is generated.

Code transformation function translates 01 strings into assembly instructions on the basis of knowledge module K₁. Owing to embedded memory capacity limit, often it’s not possible to complete translation once. Thus, target program section is segmented to proper size and transformed into assembly instruction blocks a couple of times.

Pre-judging of assembly instruction blocks mainly discards syntactically incorrect instructions according to K₁. Here define incorrect instructions as instructions which deviate from syntactic rules based on K₁.

Post-judging function summarize semantics of instruction block generated in last step to obtain ideal target processing unit that matches best with input machine code section. Alternative targets stem from K₁ and TOPSIS decision algorithm [14] is adopted here.

Finally, Post-processing module is implemented to optimize current assembly instruction block.

Compared with previous ones, this disassembler is able to acquire hardware configuration information that accords with machine code as well as output assembly instruction block”.

B. Decompiler

The decompiler translates assembly instruction block into uniform intermediate language by substitute regulation shown in table 1.
are transformed into basic block contents and others are translated into corresponding control structure information. Furthermore, high-level library function information is added to complement program structure details. As a result, high-level program is generated recursively starting from the root node of FCT.

C. Model based error orientation

1. The analysis of black-and-white screen fault

The black-and-white screen faults happened in smart meter devices in spot are analyzed concretely below.

1. Some smart meter devices’ MCU will be out of control state if the Vcpu are between 1.0V and 1.7V during power down.

2. If Vcpu are below 4.2V during power down, MCU will start to store power-down data and then change into a low-energy mode. If Vcpu are in 3.1V< Vcpu<4.2V, the system will use the capacitors to power the smart meter, then change to use the battery to power if Vcpu<3.1V. The smart meter needs higher current (about 7mA) to run the power-down processing system at full speed. If the battery doesn’t have enough energy, the battery voltage will drop to below 2.2V, then the MCU has the chance to be out of control.

All the two faults above may be caused by the unreasonable design of smart meter software. When Vcpu is in the low-voltage state, if the program can’t judge the Vcpu in time, the system will not take the power-down process in bounded time, which can cause the loss of data and the black-and-white screen fault. What’s more, the infinite loop during checking Vcpu or the failure of storing electric energy data can cause the black-and-white screen fault.

2. The method of extracting faults in

The authors worked to improve the Cppcheck software and add some functional modules, which can check the smart meter program through the configuration of command words stored in the specific ports or registers of MCU. The improved Cppcheck can check which functional module of the program can cause the black-and-white screen fault and extract the target into a PCRE regular expression, which can be used by users to position and recheck the target code.

For example, there is an embedded program used on a Renesas M16C, 78k/0 and Intel 8051. Summary outcome is shown in table 3. The “Pe” column presents accurate rate of decompiling results in contrast to actual application program.

<table>
<thead>
<tr>
<th>Device</th>
<th>Power Down Fault Rate (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Renesas M16C 90.04</td>
<td></td>
</tr>
<tr>
<td>Renesas 78k/0 90.61</td>
<td></td>
</tr>
<tr>
<td>Intel 8051 75.86</td>
<td></td>
</tr>
</tbody>
</table>

IV. EXPERIMENTAL RESULTS

The implementation of EA-CRT is tested on smart meter devices. The source code section is originally extracted in ammeter internal chips. Sample machine code is provided by our cooperation partner Zhejiang Electric Power Institute and Tianjin Electric Power Institute.

The test is applied in three types of processing unit—Renesas M16C, 78k/0 and Intel 8051. Summary outcome is shown in table 3. The “Pe” column presents accurate rate of decompiling results in contrast to actual application program.

<table>
<thead>
<tr>
<th>Test No.</th>
<th>Result</th>
<th>Real</th>
<th>Correct</th>
<th>Rate(%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>100%</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
<td>1</td>
<td>1</td>
<td>100%</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>1</td>
<td>0</td>
<td>0%</td>
</tr>
<tr>
<td>4</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>100%</td>
</tr>
<tr>
<td>5</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>50%</td>
</tr>
<tr>
<td>6</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>100%</td>
</tr>
<tr>
<td>7</td>
<td>3</td>
<td>1</td>
<td>1</td>
<td>100%</td>
</tr>
<tr>
<td>8</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0%</td>
</tr>
<tr>
<td>9</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>100%</td>
</tr>
<tr>
<td>10</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>100%</td>
</tr>
<tr>
<td>11</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>100%</td>
</tr>
<tr>
<td>12</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>100%</td>
</tr>
</tbody>
</table>

The results are listed as followings: there are total 16 real fault models within 12 tests. The EA-CRT has gained 20 fault models result from all tests. 14 are correct fault models. The
correct rate is 87%. The EA-CRT can locate most fault models of embedded code.

V. CONCLUSIONS
This paper presents the EA-CRT—an error analysis method with embedded code reverse technology. Our implementation of EA-CRT coordinates knowledge module with disassembler, decompiler and model based error orientation. Experimental data upon smart meter—a typical embedded device has confirmed the validity of our implementation to meet free decompiling demand. The model can be expanded to the other perspectives of the embedded systems well.

References