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
[2]Tianjin 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].

The rest of the paper is organized as follows. Related work is discussed in section 2, main introduction of EA-CRT is illustrated in section 3. The Disassemble and decompile module is separately elaborated in section 4 and 5. The model based error orientation is showed in section 6. Finally experimental statistics are presented and direction of future work is proposed in section 7 and 8.

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

* This work is supported by National Science Foundation of China 61303214 and China State Grid Corp science and technology project KJ15-1-32

comprehensive analysis of the graphs.

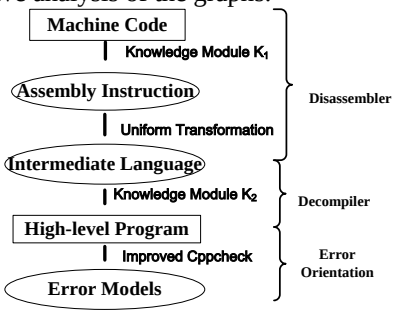

Fig 1 Structure chart of EA-CRT

A. Disassembler

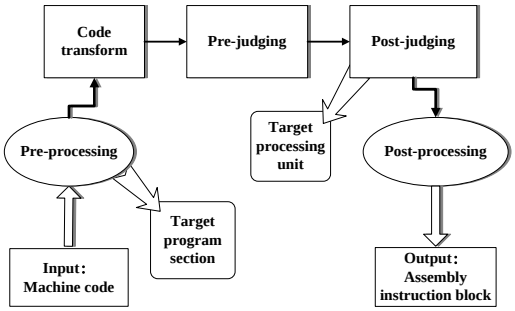

Fig 2 Disassembler

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 K_1. 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 K_1. Here define incorrect instructions as instructions which deviate from syntactic rules based on K_1.

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 K_1 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.

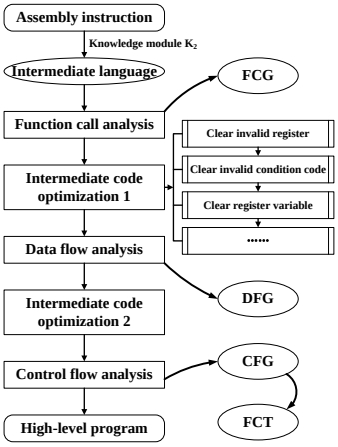

Fig 3 Decompiler

Consequently K_2 keeps unchanged during decompiling procedure. The overall process of decompilation is revealed in figure 3. Once intermediate language is formed, target instruction blocks become syntactically unambiguous. For each subroutine, a basic block is built. The entire intermediate representation is transformed into a *function call graph* (FCG).

A variable is treated as invalid when it has not been used after definition. Often invalid variables are meaningless and can be swept out. *Intermediate code optimization 1* is performed to clear invalid register, invalid condition code, register variables, redundant and transition instructions. Expressions with more than two operands are imported to initially shape high-level concept.

To ensure coherence of global data and variable, decompiler collects information about register and condition code throughout target instructions and spread it out across basic blocks. This multi-block information gathering procedure is titled as *data flow analysis*. A data flow graph (DFG) is formed and it contains information described above. With DFG in hand, global data and variable information are applied into intermediate code, for instance, displacing variable mnemonic, etc. That is exactly the objective of *Intermediate code optimization 2*.

FCG contains details about function calling relationship, structural information not involved. After data flow analysis, target intermediate code is so semantically distinct that control flow information appears effortless to obtain. *Control flow analysis* converts intermediate code into a structured diagram called control flow graph (CFG) via proper structured algorithm.

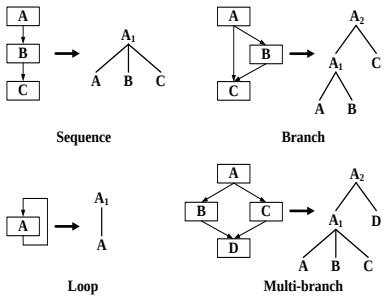

Fig 4 CFG to FCT templates

CFG is processed to obtain function control tree (FCT) based on rules shown in figure 4. Finally in FCT, leaf nodes

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 m

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 H8/38024 MCU smart meter device. This program takes the power-down testing and battery testing action through the interrupt handler function periodically. The cycle time is 0.0016384s. Once the interrupt has been raised, the program will use an A/D converter to detect the battery voltage for constantly. If the measured voltage value has been less than the reference value for more than twice, the program will set the variable *_PowerDown* to 1, then waiting for the power-down process.

In this program, the assignment statement *HF_ADSR_ADSF=1* will set the bit-7 *ADSF* of an 8-bit read/write register *ADSR* , which can start and stop A/D, to 1. Then the A/D converters start A/D conversions to measure the battery voltage. However, if the assignment statement above never appears in the program, the smart meter device will not have ability to detect the changes of voltage. The improved Cppcheck can check the values of register variables like *HF_ADSR_ADSF* and extract the incorrect statements into regular expressions. The users will know the specific location of the questionable code through the regular expressions.

The PRCE regular expression is presented below in XML format.

```
<?xml version="1.0"?>
<rule version="1">
    <pattern>HF_ADSR_ADSF=0;GuB_SysTmp=0;while\(HF_AD
SR_ADSF\){GuB_SysTmp--;nop\(\);if\(GuB_SysTmp<1\)break;}
    </pattern>
    <message>
        <id>powerdownCheck</id>
        <severity>warning</severity>
        <summary>powerdown check.It's invalid to let adsf set to
    0 when you check the voltage.
        </summary>
    </message>
</rule>
```

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 1 Decompiling result

Type	Pe(%)
Renesas M16C	90.04
Renesas 78k/0	90.61
Intel 8051	75.86

The smart meters' black-and-white screen fault is used for examining the correctness of our method. The test is repeated for 12 times. The first 8 tests run for checking the state of register variables of the program, and the last 4 tests run for checking logical errors of the program. Column 2 presents the fault models results gained by EA-CRT. Column 3 shows the real fault models. Column 4 indicates the correct models of the results. Column 5 is calculated by dividing 'correct' with 'result'.

Table 2 Error Analysis result

Test No.	Result	Real	Correct	Rate(%)
1	1	1	1	100%
2	3	1	1	100%
3	2	1	0	0%
4	2	2	2	100%
5	1	2	1	50%
6	2	2	2	100%
7	3	1	1	100%
8	0	1	0	0%
9	1	1	1	100%
10	2	1	1	100%
11	2	2	2	100%
12	1	1	1	100%
\	20	16	14	87%

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

[1] *Related definition of Reverse Engineering, Wikipedia* [EB/OL]. http://zh.wikipedia.org/wiki/%E9%80%86%E5%90%91%E5%B7%A5%E7%A8%8B 2013-9-4.

[2] *The advent of 64GB iNAND embedded flash drives* [EB/OL]. http://www.dzsc.com/data/html/2011-9-3/97488.html 2013-9-4.

[3] C. Kruegel, W. K. Robertson, F. Valeur, et al. Static Disassembly of Obfuscated Binaries [C]. *In: USENIX Security Symposium*. 2004, 13: 18-18.

[4] N. Krishnamoorthy, S. Debray, K. Fligg. Static detection of disassembly errors [C]. *In: Reverse Engineering, 2009. WCRE'09. 16th Working Conference on*. IEEE, 2009: 259-268.

[5] B. Schwarz, S. Debray, G Andrews. Disassembly of executable code revisited [C]. *In: Reverse Engineering, 2002. Proceedings. Ninth Working Conference on*. IEEE, 2002: 45-54.

[6] S. Nanda, W Li, L. C. Lam, et al. BIRD: Binary interpretation using runtime disassembly [C]. *In: Proceedings of the International Symposium on Code Generation and Optimization*. IEEE Computer Society, 2006: 358-370.

[7] Song X, Wang P, Zhou S., Software Testing Technology Overview of the Smart Meter, [J]Electrical Measurement & Instrumentation. 2014: 11[51]，18~22

[8] Qi Weiwei, Study on the method of full performance detection of intelligent power meter, Telecom World , 2014: 10,134-135

[9] Alabdulkerim L., Lukszo Z., Information Security Assurance in Critical Infrastructures: Smart metering case [c]. 2008 First International Conference on Infrastructure Systems and Services: Building Networks for a Brighter Future (INFRA), 2008:1-6

[10] Ye L, Xia S,. Shen L, Design of Smart Energy Meter Communication Protocol and Function Consistency Testing. [J] Electrical Measurement & Instrumentation. 2010:8[47]18-25L.

[11] JIANG, B. ZHOU, et al. Study of Structural Analysis Algorithm for Disassembled Code [J]. *Journal of Chinese Computer Systems*, 2007, 6: 019.

[12] J. LIU, X.WANG, et al. A method to process disassembly Endianness for firm-code: China, CN201210489305.4 [P]. 2013-3-13.

[13] Liu J, Wang X，et al, Analysis of key disassembly problems based on embedded smart meter, [J] journal of Computer Applications，2014,5(2):555-559

[14] Liu J, Wang X，et al, Smart meter software function test model based on disassembly technique, [J]journal of Computer Applications，2014, 34（12）:3507-3510.