JOURNAL OF INFORMATION SCIENCE AND ENGINEERING 36, 495-512 (2020)
DOI: 10.6688/JISE.202005_36(3).0002

A Novel Debugging Technique Based on Lightweight Crash Report Considering Security

DONGMIN JANG[1], SUJUNE LEE[1], YOOWON JANG[2], HOHYEON JEONG[1] AND EUNSEOK LEE[3]
[1]*Department of Electrical and Computer Engineering*
[2]*Department of Software Platform*
[3]*School of Software*
Sungkyunkwan University
Suwon, 16419 Korea
E-mail: {jangdm; hoakw; jangyoowon; jeonghh89; leees}@skku.edu

Crashes can occur due to code defects while using released software. This is mainly caused by various errors ranging from simple errors to unhandled exceptions. When a crash occurs, a crash report is generated and transmitted to the developer to debug the code. Debugging for tracing and correcting them is a very important task in terms of improving the dependability of the software. The problem is that the crash report contains too much information. So that it is difficult to focus on the core information related to the crash. To make matters worse, in the security-critical situation, such as the case of the defense-related sites to which this paper is targeted, important exception information of client for debugging is not properly provided. In this paper, to solve the above problem, we propose a novel technique to automatically generate high-quality lightweight crash report with high security by collecting exception and memory information useful for error tracing without violating user's personal information in the execution environment. Furthermore, we propose a precise error tracing technique by linking the crash report with the source code of the development environment. To validate the proposed technique, we applied it to prominent open source projects, such as security, registry, and so on using the MS Windows platform. And we compared the results with WinDbg, the most powerful tool available for the same purpose. As a result, our proposed technique improves security by excluding five critical information that threatens security while maintaining error tracing accuracy of existing research. In addition, the amount of information needed for error tracing is reduced by 72%, making it easier for developers to resolve errors. Finally, the automation of crash report generation and error tracing improves error tracing efficiency by reducing the time required for error analysis by 78%.

Keywords: software debugging, error tracing, crash report auto-generation, secure crash report, memory dump

1. INTRODUCTION

The functions of the software provided to users are becoming increasingly complex. As a result, the size of software and the cost of development have increased over the past several decades. In this situation, the most expensive part of the software life cycle is debugging, which accounts for about 50% of the total development cost [1, 2]. Since the cost of debugging increases in proportion to the life cycle of the software, the most cost part is the debugging performed during the maintenance phase after the software is released [3, 4]. Fig. 1 shows the process of debugging the software is released. If an error occurs while running the software released in the execution environment, a crash report

Received April 23, 2019; revised July 5 & September 1, 2019; accepted September 3, 2019.
Communicated by Chang-Shing Lee.

is generated and a crash report is sent to the developer. After receiving the crash report generated in the execution environment, the developer analyzes the crash report in the development environment, to trace and correct the location of the error. Then, when all the tests pass, the developer has a recursive structure to release the software again. The workflow in Fig. 1 is a very important task in improve the dependability of software. Software crash is often caused by uncomplicated problems such as unhandled exceptions in simple errors [5]. However, if a novice developer who does not have enough knowledge of debugging is error tracing, it takes more time to solve even it is a simple problem, which reduces the efficiency of error tracing. In addition, experienced developers are involved if the software requires complex debugging. However, even experienced developers may find it difficult to debug and take a long time to work [6]. In this paper, we discuss three main causes (error report generation, analysis, and error tracing) of debugging difficulties, and propose an error tracing method that facilitates debugging even for novice developers who are not skilled.

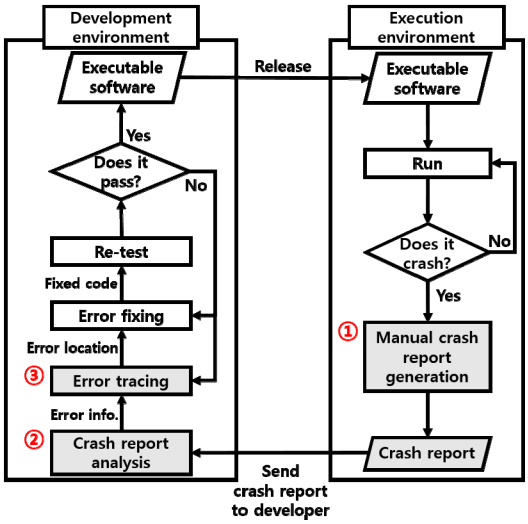


Fig. 1. Debugging process of the released software.

First, when an error occurs, the existing crash report generation step generates a dump file contain information, such as system information, memory dump, and crash dump from the operating system [7-9]. At this time, all or part of the generated raw dump file is attached to the crash report [8, 10-13]. In this case, the developer can confirm various items of information about the execution environment. However, there is a disadvantage that developers cannot easily debug problems because they need to analyze the unprocessed information of a crash report. Another problem is that if the user deactivates the ability to send crash reports generated by the operating system, the developer will not receive the error information and will not be able to trace the error [8]. For example, a dump file of security-critical software used by a defense security agency or government agencies that has security implications contains a lot of important information, so it difficult to keep trace of the error because the dump file is not provided to

the developer.

Second, the existing crash report analysis step processes the information of the crash report generated in the execution environment. Thus, the developer can analyze the contents of the dump file more easily [14], and receive the information necessary for error tracing [15, 16]. However, information that is not fully processed can contain meaningless data in the error tracing phase. In addition, there is a possibility that other module information list, and personal information used by the user among the attached data may be included, which is also weak in terms of security [17]. Therefore, it may take a long time to process the information, due to the dump data collected together with the unnecessary data for error tracing. Also, there is a lot of data to analyze including the processed error information (or crash information), which requires a lot of time and effort by the developer.

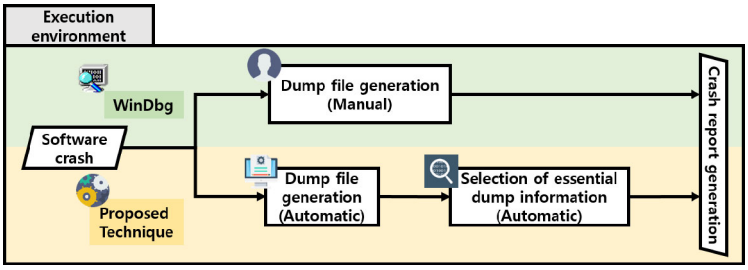


Fig. 2. (a) Crash report generation in the execution environment.

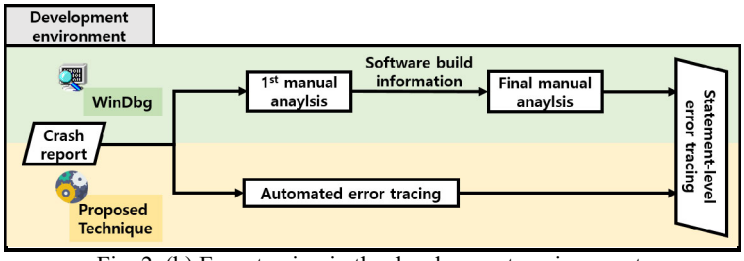


Fig. 2. (b) Error tracing in the development environment.

Third, the error tracing step of the existing error tracing techniques [15, 16] uses error information and source code to trace errors in files or functions. However, since the tracing scale is large, additional time and effort is required by the developer to find the accurate error location. For the most popular error tracing tool in Windows environments WinDbg [14], statement-level error tracing is performed using the program database information [18], dump file, and source code used in the debugging mode of the Integrated Development Environment (IDE). However, in order to trace errors on a statement-level basis, the developer must do two manual analyzes as shown in Fig. 2. If the software crashes in the execution environment as shown in Fig. 2 (a), the user must generate the dump file and manually send the dump file to the developer to trace the error. When a user sends a dump file to a developer, a file level error tracing is needed to first determine what version of the software has crashed. As shown in Fig. 2 (b), the developer extracts the build information by input the dump file received from the user and the

symbol file path. This allows you to check the source code path in the development environment while checking the version of the software. Secondly, the developer can trace statement-level errors by placing a dump file, symbol file path, and source file path as input to WinDbg. Developer can trace errors in this way, but user cannot error trace them unless you provide a dump file for security problem or other reasons in execution environment. In addition, even if developer receive a dump file from the execution environment, two manual analysis tasks require additional debugging time and effort [19].

In order to overcome these limitations, we propose the automatic generation and analysis of crash reports and automated error tracing for efficient debugging. The proposed automatic crash report generation step collects from the dump data only the executable file unique number, relative virtual address information, stack information, and executable environment information, which are the minimum information required for a statement-level error tracing. By processing the collected data, it is possible to generate a lightweight crash report that does not contain personal information (Section 3.2). In addition, the lightweight crash report delivered to the developer can be analyzed to identify software that require error tracing automatically (Section 3.3), and to analyze the program database of the software to automatically trace statement-level errors (Section 3.4).

The contribution of this paper is as follows:

- The efficiency of crash report generation is increased by the collection of information that is essential for error tracing as a lightweight crash report unlike existing crash reports.
- Crash report can be used with security-sensitive software as it improves security while eliminating user private information and module information other than errors.
- When a software crash occurs, the software architecture of the client-server structure allows the developer to always receive a crash report and tracing the error to the statement-level.
- Debugging costs can be reduced by automating error trace information collection, crash report generation, and statement-level error tracing.

To verify the validity of this research, we apply the proposed technology to trendy open source projects [20-29] by each category. The experimental result show crash reports automatically generated by the proposed technology have only about 28% of the information of existing reports, reducing the information to be analyzed. Also, the time required for error tracing is reduced by about 78%. In addition, solves the problem that crash reports are not attached to developers due to security problems in software (*e.g.*, defense security software, government related software, *etc.*). The above solution eliminates security-related information (other module information list, and personal information, *etc.*) in the execution environment, thereby maintaining the accuracy of error tracing and improving the security of error report. As a result, we confirmed the effectiveness of debugging efficiency improvement by automatically generating lightweight crash reports with enhanced security and automating statement-level error tracing.

The rest of the paper is organized as follows. Section 2 reviews related studies on error tracing techniques. Section 3 describes the proposed techniques. Section 4 presents the experimental results for the proposed technique. Section 5 discusses threats to validity. Finally, Section 6 concludes the paper.

2. RELATED WORK

In general, the starting point for debugging released software is when software that runs in the execution environment experiences a crash, such as an unhandled exception. In order to solve the above-mentioned situation that deteriorates the usability of the software, research [15, 16] and tools [14, 30] have been provided for performing error tracing tasks. These techniques [14-16, 30] basically assume debugging after the developer receives the crash report with the dump file attached. A dump file is a file that stores the system status at a specific point in time provided by the operating system, and includes various status information, such as system information, CPU status, memory dump, and process information.

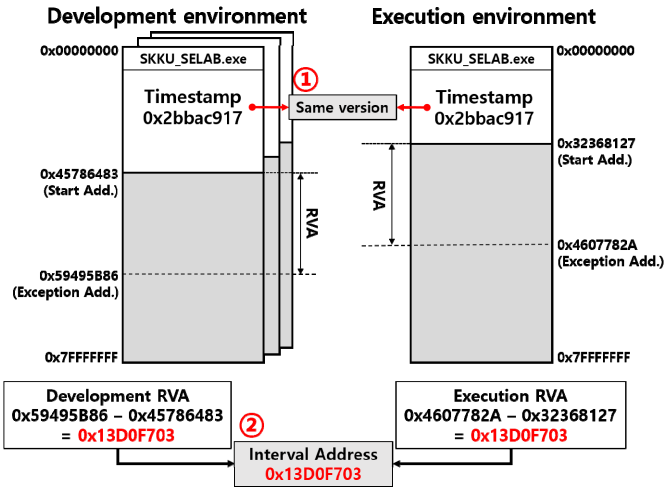


Fig. 3. Crash point specific to relative virtual address (RVA) by environment.

RETracer [15] is a binary-level backward taint analysis service that is suitable for large-scale crash reporting. The input to the RETracer extracts the binaries of the dump file information, the stack memory of individual processes, the crash thread, and the memory dump of a crash. Then, a backward data-flow graph is generated based on the crash point of the stack, and a bad value node, such as a damaged pointer, is selected. The service analyzes the address value of the selected node, and traces the error by function. In the case of CrashLocator [16], the crash stack information is used to trace the error. The software analyzes the call graph and control flow to obtain crash stack information, and calculates the suspicion score of approximate function through backward slicing. The calculated suspicion score is ranked, and transmitted to the developer. Therefore, the developer can check the top n functions to find the error. These studies [15, 16] reduce the debugging cost by extracting meaningful information from the dump file for error tracing. However, because the unit of error tracing is a file or function unit, the developer must perform additional error tracing to determine the exact location of the error. Because of this, the time and effort employed by the developer is high.

WinDbg [14] is a support tool for postmortem analysis debugging [31] provided by

Microsoft. This tool is the most popular used and powerful tool for debugging in the Windows environment. It can analyze the dump data through various commands by inputting the dump file created in the Windows environment. Dump analysis includes source code debugging, memory dump file, crash dump, breakpoint setting, and call stack [32, 33]. In addition, the information is processed, and provides output in a text format that can be understood by the developer. However, the analysis output is huge, and requires a lot of relevant experience and knowledge for developers to trace errors. Moreover, so as to trace statement-level errors in the source code, the developer must first extract the information (executable file or dynamic library file) of the module that crashed in the crash dump and satisfy both the conditions. The satisfying conditions are: 1) there is program database file path information in the extracted module information, and 2) there is a section in which the Relative Virtual Address (RVA) and source code line information are mapped in the program database file in the corresponding path. If both of the above conditions are satisfied, it is possible to trace the source code statement-level error through a specific point of crash. Fig. 3 shows a schematic of the technology [34]. The schematic shows that it is possible to trace source code statement-level errors through a crash point (Section 3.2). CrashRpt [30] is an open source-based error reporting library and program. The tool generates a crash report containing a screenshot, a dump file, and a hexa-type error log at the time of the crash, and delivers it to the developer.

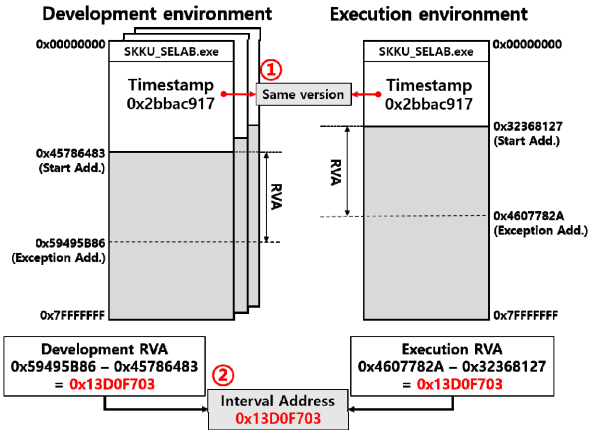


Fig. 3. Crash point specific to relative virtual address (RVA) by environment.

Although it is possible to reduce the debugging cost by using the above tools, WinDbg [14] has many limitations for statement-level error tracing, and when the constraints are not satisfied, developers must manually trace the error. CrashRpt [30] only supports the generation and delivery of a crash report, so the cost of error tracing is large. Commonly both tools communicate dump data to the developer, which may contain personal information.

As shown in Table 1, the above studies [15, 16] and tools [14, 30] have significant impact on the generation, analysis, and error tracing phases of crash reports. However, CrashLocator and CrashRpt perform real time error monitoring to detect errors in the

execution environment. Therefore, personal information and information of all the running processes may be collected and it is vulnerable to security. In particular, CrashRpt is more vulnerable to security because it snapshots the desktop of the execution environment. When the software crash is detected, most tools generate a dump file. RETracer and WinDbg require the user to manually create a dump file. But CrashLocator is excluded from the comparison because it starts error tracing based on the assumption that there is a crash report. The output data is different for each technology, and the data that a developer intuitively understands and debugs the errors is a crash report. A crash report is the best output data because it contains comprehensive debugging information as well as software crash information. Crash report of CrashRpt is in binary-level, so developers need to convert it to natural language, and WinDbg needs to analyze 82 different infor mation (to filter unnecessary information for debugging). Comparing the error tracing level, RETracer, CrashLocator, and CrashRpt are function-level error tracing. WinDbg cannot do statement-level error tracing without further analysis as shown in Fig. 2 (b).

Table 1. Comparison of existing error tracing techniques and proposed error tracing technique.

	RETracer	Crash Locator	CrashRpt	WinDbg	Proposed Technique
1) Error monitoring	X	O	O	X	X
2) Data collection	Dump file generation (Manual)	N/A	Dump file generation (Auto)	Dump file generation (Manual)	Dump file generation (Auto)
3) Method of error analysis	Backward taint analysis	Ranking suspicious functions	Capture and Replay	Dump file analysis	Selection of essential dump file
4) Output	Tainted function paths	Error ranking list	Dump file Snapshot Crash report (Binary-level)	* Crash report (#82 info.)	* Crash report (#23 info.)
5) Error tracing level	Function	Function	Function	Limited statement	Un-limited statement
6) Security consideration	X	X	X	X	O

* Detailed information is given in Table 2.

The above related technologies have a limitation that security cannot be assured due to the risk of leakage of personal information due to error monitoring and dump file information. And additional debugging cost is required for statement-level error tracing.

3. METHODOLOGY

This section describes how to automatically generate crash reports by collecting only the necessary information for error tracing, and how to trace errors based on the generated crash report.

3.1 Overview

Fig. 4 shows a flowchart of the proposed method and the detailed process of crash report automatic generation, analysis and statement-level error tracing in this paper.

First, if a crash occurs while running the released software in the execution environment, the automatic crash report generation step (Section 3.2) is performed. At this stage, the software performs (1) dump file creation; (2) collection of essential information for error tracing; and (3) crash report auto generation, and sends the crash report to the development environment. After completing the above steps, the crash report analysis phase (Section 3.3) is performed in the development environment. This step (1) analyzes the content of the crash report received in the execution environment; (2) finds the project path that matches the released software based on the analysis result; and (3) finds and analyzes the program database file of the project. Finally, the error tracing step (Section 3.4) is performed. In this step, two input values are used: (1) the interval address of the crash point included in the crash report; and (2) the source code-RVA mapping information extracted in the line section of the program database file. Therefore, the proposed method does not require the developer to ask the user for a dump file, unlike the logic of WinDbg (Fig. 2), which is a popular error tracing tool. In addition, security is considered by generating a crash report with only essential information for error tracing, except for security sensitive information such as user personal information. Also, the efficiency of the debugging work is improved by the developer automating the crash report generation, and the error trace of the statement-level.

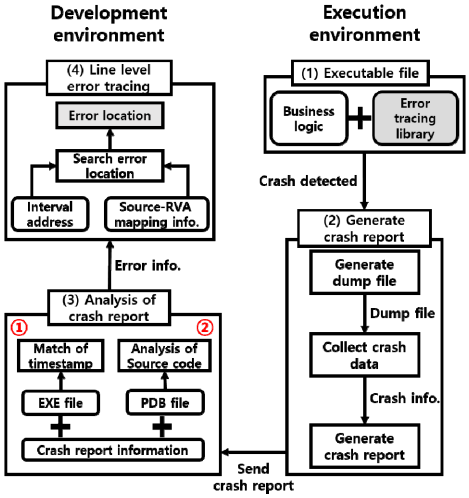


Fig. 4. Overview of crash report generation, analysis, and error tracing methods.

3.2 Crash Report Auto Generation

The goal of the automatic crash report generation step is to collect only the information that is necessary for statement-level error tracing in the execution environment, generating a crash report, and then delivering it to the developer.

3.2.1 Collection of crash information

Table 2 shows 82 information obtained from the existing dump file [35], and 23 error tracing collection of information the crash report proposed in this paper. The proposed crash report collects four types of information from the dump file. The collected information items are system information, module information, thread information, and exception information.

Table 2. Comparison between legacy (WinDbg) and the proposed crash report collection information.

System Info. (#4)	OSBUILD	OSNAME	OSPLATFORM TYPE
	BUILDOSVER STR		
Module Info. (#8)	BUCKET ID	BUCKET ID MOD TIMEDATESTAMP	BUCKET ID OFFSET
	FAILURE BUCKET ID	FAILURE EXCEPTION CODE	FAILURE FUNCTION NAME
	FAILURE MODULE NAME	MODULE NAME	
Thread Info. (#4)	FAULTING IP	FOLLOWUP IP	STACK TEXT
	THREAD ATTRIBUTES		
Exception Info. (#7)	DEFAULT BUCKET ID	ERROR CODE	EXCEPTION CODE
	EXCEPTION CODE STR	EXCEPTION RECORD	EXCEPTION ADDRESS
	PROCESS NAME		
Excluded Info. (#59)	ANALYSIS SESSION TIME	ANALYSIS SESSION HOST	ANALYSIS SESSION ELAPSED TIME
	ANALYSIS SOURCE	ANALYSIS VERSION	BUCKET ID FUNCTION STR
	BUCKET ID IMAGE STR	BUCKET ID MOD CHECKSUM	BUCKET ID MODULE STR
	BUCKET ID MODVER STR	BUCKET ID PREFIX STR	BUGCHECK STR
	BUILD VERSION STRING	BUILDDATESTAMP STR	BUILDLAB STR
	CONTEXT	DEBUG FLR IMAGE TIMESTAMP	DUMP CLASS
	DUMP FLAGS	DUMP QUALIFIER	DUMP TYPE
	EXCEPTION PARAMETER1	EXCEPTION PARAMETER2	FAILURE ID HASH
	FAILURE ID HASH STRING	FAILURE IMAGE NAME	FAILURE PROBLEM CLASS
	FAILURE SYMBOL NAME	FAULT INSTR CODE	FOLLOWUP
	FOLLOWUP NAME	IMAGE NAME	LAST CONTROL TRANSFER
	* MODLIST SHA1 HASH	* MODLIST WITH TSCHKSUM HASH	NUMBER PARAMETERS
	OS LOCALE	OS REVISION	OSBUILD TIMESTAMP
	OSEDITION	OSSERVICEPACK	PRIMARY PROBLEM CLASS
	PROBLEM CLASSES	PRODUCT TYPE	SERVICEPACK NUMBER
	STACK COMMAND	SUITE MASK	SYMBOL NAME
	SYMBOL STACK INDEX	TARGET TIME	* THREAD SHA1 HASH MOD
	* THREAD SHA1 HASH MOD FUNC	* THREAD SHA1 HASH MOD FUNC OFFSET	USER LCID
	WATSON BKT MO-DOFFSET	WATSON BKT MOD-STAMP	WATSON BKT MODULE
* Security-sensitive information	WATSON BKT PROCSTAMP	WRITE ADDRESS	

The system information collects the operating system type, operating system version, processor level, and processor architecture information. This system information enables the developer to understand the environmental information of the execution environment in order to reproduce the error. Also, if the development environment is different from the execution environment, it is difficult for developers to trace errors. Therefore, basic information about the system is essential. The module information collects the name of the module, which is an executable file or dynamic library, the physical address and size of the module loaded into the physical memory, and the unique number of the module. This module information provides overall information about the module in error. When a version is upgraded due to the bug fixing of adding additional function in the module, a version-specific module is stored in the development environment. Therefore, it is necessary to provide information for exploring the same module among the various modules in the development environment and the module in which the error occurs in the execution environment. The thread information collects the call stack information by using the thread ID, back trace method, and thread attribute in which the exception occurs. This information can be traced back to the execution history by specifying the point at which the module crash occurred as the start point. In addition, it confirms the behavior and flow performed by the user of the execution environment. Finally, the exception information collects faulty physical memory start/end addresses, faulty statement addresses, interval addresses, error codes, and error types (*e.g.*, null pointer references, divide by zero, *etc.*). This information identifies what kind of exception occurred in the module. Through the error code and error name, the developer can intuitively identify what error occurred. In addition, the location of the error is traced to the statement-level through the interval between the start/end address of the error module and the physical address where the error occurred.

Therefore, the crash report proposed in this paper excludes 59 information from the existing crash report (WinDbg). There are two reasons why the information in the crash report is excluded. The first reason is the removal of redundant information in the crash report. In the crash report information output by WinDbg in Table 2, MODULE NAME, BUCKET ID IMAGE STR, BUCKET ID MODULE STR, FAILURE IMAGE NAME, IMAGE NAME, and WATSON BKT MODULE all show the same module name information (*e.g.*, SKKU_SELAB.exe). It is based on actual data consisting of executable or dynamic library file. Thus, our crash report only collected MODULE NAME, and we excluded other redundant data sets like this. The second reason is security-sensitive information. The five kinds of security-sensitive information are the list of modules not related to the error, or the information in which the personal information in the execution environment is exposed in a hash format. As a result, it eliminates redundant data and security-critical information, and automatically generates a lightweight crash report that contains 23 information that is essential for the error tracing.

3.2.2 Essential information required for error tracing

Fig. 3 shows that the essential information required for statement-level error tracing is the RVA and timestamp. The RVA is a relative virtual address representation of the data or operation information required for program execution based on the module start address 0×0, regardless of where it is loaded into memory. With this characteristic, the

error point can be found regardless of the execution environment, by taking the identifying difference between the error occurrence address of the physical memory, and the module start address. For the address difference information to be meaningful, it is necessary to analyze the version based on the same module, and it can be searched using the time stamp information. After extracting the minimum information required for error tracing from the dump file using such program operation characteristics, the contents are included in the crash report, then transmitted to the developer.

3.3 Crash Report Analysis for Error Tracing

The crash report analysis phase analyzes the error based on the information contained in the crash report received from the execution environment. The developer can use the timestamp information from various items of information in the crash report to trace the project path of the software that crashed. In addition, the goal is to extract the statement-level error information using the difference between the start address of the software, and the address where the problem occurred.

3.3.1 Exploration of the same module based on the timestamp

To use the address difference information of the proposed crash report meaningfully, we search for the same module of the execution environment module and the development environment with the time stamp information. The exploration method explores the project paths of released software by using the module name and time stamp included in the crash report. This method can analyze the header of files with the same extension as an error module, and trace file-by-file errors by finding the path of the executable file or dynamic library file having the same timestamp.

3.3.2 Analysis of RVA-line mapping information based on program database

Once the file with the error is found by timestamp, it analyzes the program database [18] to utilize the address difference information. The program database file can be created in the same path as the module through option setting in debugging mode, when compiling the released module. This file is generated to support debugging during development, and includes information, such as the path information of resources required for operating the module, RVA mapping information per line of source code, and symbol table. At this time, the RVA mapping information for each line of the file is analyzed, and the RVA mapping information for each source code line and the address difference information extracted from the conflict report are transmitted to the error tracing step.

3.4 Statement-Level Error Tracing

In the statement-level error tracing step, our goal is to specify the error location in line units through the RVA mapping information of the source code line received in the crash report analysis step and the address difference information in the error module. Fig. 3 shows that basically, when using the same address difference value in the same module, it is possible to find an exact error point, irrespective of the execution environment.

The RVA mapping information obtained from the analysis of the program database is composed of a list of functions. Fig. 5 shows that in each list, RVA-line mapping in-

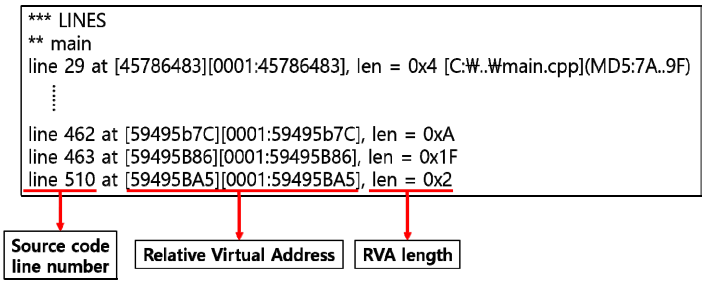


Fig. 5. Relative Virtual Address (RVA) and source code line mapping information.

formation is recorded. Among the recorded information, the mapping RVA information and the source code line information may not be exactly the same, and the accurate source code line is traced using the mapped RVA and RVA length information. Finally, if the source code line of the range including the address difference value is found, the error statement information is derived together with the information included in the crash report.

4. EXPERIMENTAL RESULTS

This section validates the proposed error tracing technique using WinDbg [14], which is popular in the Microsoft Windows platform environment, to verify the validity with the C/C++ language open source projects [20-29] based on the Native Win32 API. The experimental results are evaluated through the following three research questions:

- RQ1: Is the quality of the proposed crash report improved in comparison to the existing crash report?
- RQ2: Has the proposed error tracing technique reduced the error tracing cost compared to the previous studies?
- RQ3: How effective is the proposed crash report in terms of security?

The experiment method is to inject errors and library-type suggestion techniques that cause frequent crashes [5], such as null pointer dereference or divide by zero, in the open source project on the development environment PC, and distribute them to the execution environment PC. When a crash occurs due to an error injected by the released software, the project collects and analyzes the memory dump file, and checks the validity of the statement-level error trace through the crash report. All our experiments are based on Windows OS NT/Server version and 32/64bit architecture. The experimental environment is equipped with Intel Core i5-7300HQ CPU@2.50GHz and 8GB RAM. The development environment server has Intel Core i5-7600 CPU@3.50GHz and 16GB RAM.

4.1 RQ1: Is the proposed crash report high quality?

Basically, the high-quality crash report should contain information to quickly resolve software crashes, and the developer should receive a crash report every time an error occurs. Table 2 shows that the comparison target WinDbg [14] tool can obtain 82

error tracing information, and the developer can analyze detailed information through various item of error information. However, some pieces of information are meaningless to the actual error tracing, developer readability is degraded, and the error analysis time increases. On the other hand, the proposed crash report uses only 23 error information, including time stamps and RVA address values, which are essential for error. Therefore, we create high-quality crash reports that can trace errors in statement-level by using only the core information needed for the error tracing, rather than using a low-quality crash report that contains information that is not structurally existing, or meaningless to error tracing.

In addition, WinDbg must insert the dump file, symbol file path, and source code path to get accurate error tracing information. However, when the developer receives a dump file from the execution environment, it does not immediately know the timestamp information of the software that caused the crash. Therefore, we first extract the timestamp information using the dump file and symbol file path information. Complete error tracing information can be obtained only by inserting an error source code path using the extracted timestamp information. In this way, developers need three input files to obtain accurate crash information using WinDbg. When a developer receives only a dump file, it needs to perform two operations to ensure accurate error tracing, and cannot analyze the error if the dump file is not received in the execution environment. On the other hand, the proposed crash report can always check whether the error has occurred by sending the crash report to the developer in the execution environment whenever an error occurs in the released software. Also, the proposed crash report generates a high-quality report that can be debugged quickly without any additional work of the developer, unlike WinDbg.

4.2 RQ2: How effective is the proposed technique in error tracing?

Table 2 shows that the existing error tracing tool, WinDbg [14] analyzes the error results of about 82 memory dump analyses, including system environment information of the execution environment. In addition, WinDbg should use the source file path, the symbol file path, and the dump file as input values, in order to obtain accurate analysis results. If any of this information is excluded, the developer needs further analysis effort. However, it is difficult for the developer to see the results at once, and error tracing costs a lot of analysis. On the other hand, the proposed technique can obtain the same error tracing accuracy with only 23 analysis results, which is about a 72% decrease in the existing research. Therefore, the size of the crash report has been reduced, and the readability is also improved, because the developer only reads the core contents necessary for error tracing.

Table 3 shows the experimental results. The total analysis time showing the error line position was reduced by about 78%, compared to existing tools. There are two main reasons why the proposed error tracing technique takes less time to trace errors. First, the input value type of WinDbg is the input data of the mini dump file itself, so the data to be analyzed is vast. However, the error tracing method proposed in this paper uses a lightweight crash report, so the size of the data to be analyzed is small, and the analysis time can be reduced. The second existing tool is either unable to trace the error line in the program database file, or it takes a long time to trace down to the statement-level.

Table 3. Comparison of error tracing time between WinDbg and the proposed technology.

Program	Tool	Error tracing time (s)
Process Hacker	Proposed Tech.	3.821
	WinDbg	17.545
Navicat keygen	Proposed Tech.	5.219
	WinDbg	21.915
idenLib	Proposed Tech.	3.158
	WinDbg	14.744
KernelModeMonitor	Proposed Tech.	2.469
	WinDbg	10.688
Sandbox Detection	Proposed Tech.	2.691
	WinDbg	11.455
TS Security Editor	Proposed Tech.	1.105
	WinDbg	8.612
RSVWR	Proposed Tech.	1.691
	WinDbg	8.894
File I/O	Proposed Tech.	2.006
	WinDbg	9.885
Cdr2pdfviewer	Proposed Tech.	1.592
	WinDbg	8.104
Mynote	Proposed Tech.	1.936
	WinDbg	9.577
Average efficiency (%)		78.84%

However, in this study, since the trace path is set in advance as the input value, it takes less time to find the error in the program database file.

4.3 RQ3: Has the proposed crash report improved security?

Existing crash reports can contain sensitive personal information, because they are sent with a list of all modules running in the execution environment and memory information. Therefore, if a development environment PC is hacked by a malicious hacker, a crash report containing user or execution environment information may be leaked, which may cause a serious security problem. For example, leaks of a crash report provided by security-sensitive defense companies and governments could cause serious problems.

However, the crash report proposed in this paper does not use the user personal information, because it only collects system environment information, associated information about the executed module, and exception information. Therefore, as in the existing research, the developer can ask the user for the dump file containing their personal information, or not directly analyze the dump file by the developer, and can trace the error module and statement-level error through the auto generated crash report. As a result, the security of the user's personal information is improved, while keeping the accuracy of error tracing the same as for the existing research.

5. THREATS TO VALIDITY

An external threat to validity is that our implementation and evaluation were fo-

cused on the Windows platform running on 32/64bit architectures. However, our error tracing design is generally available regardless of operating system. Therefore, we expect to extend the proposed technology to work with other operating systems and architectures such as Linux.

An internal threat is a validity that our proposed technology is related to security. Our proposed technology focuses on defense security software or security-related software. Their commonalities do not provide information about errors. Therefore, the developers do not receive useful information for debugging, including dump file. To mitigate this threat, we analyze security critical information and information essential for error tracing from the execution environment. It then provides developers with a lightweight crash report that can focus on the error.

6. CONCLUSIONS

In this paper, the high-quality lightweight crash report is automatically generated that provides only information essential for error tracing in the event of an error in released software. The generated crash report has improved security by excluding private information of the execution environment and module information other than errors. In addition, we propose an error tracing automation technique based on the generated crash report. The proposed technology is inserted and released into the developed software as a library. When a software crash occurs in the execution environment, it is recognized, and the lightweight crash report is automatically generated by collecting only the minimum data required for error tracing. The developer can use the Timestamp information included in the lightweight crash report to check the location and version of the module where the error occurred, and use the RVA information to trace the software's statement-level error. In order to supplement existing crash reports, which include information not related to error tracing (module information not related to errors, user personal information), the crash report is automatically generated by collecting only the minimal information (system information, module information, thread information, exception information) from the error trace. Also, by analyzing the crash report and automating the error tracing technique at the statement-level, we obtained error tracing results at the same level as the previous studies. Experiments have reduced the time required for crash report analysis and error tracing by 78%, compared to traditional error tracing tools, and increased the security of personal information leakage, by collecting and sending only the data required for error tracing to developers. Therefore, we confirmed that the proposed method can be used in research institutes and companies that use security-sensitive software as well as the crash report generation method and error tracing method proposed in this paper. In future research, we will create metrics that can be used to evaluate the practicality of this study for commercial software rather than open source, and compare it with commercial tools.

ACKNOWLEDGMENTS

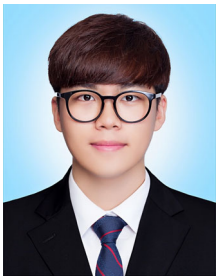
We thank the anonymous reviewers for their helpful feedback. I thank Sujune Lee, Yoowon Jang, and Hohyeon Jeong for help and support in implement proposed tech-

nique. This research supported in part by the Next-Generation Information Computing Development Program (2017M3C4A7068179), and the Basic Science Research Program (2016R1D1A1B03934610, 2019R1A2C2006411) through the National Research Foundation of Korea (NRF) grant funded by the Korean Government (MSIT).

REFERENCES

1. T. Britton, L. Jeng, G. Carver, P. Cheak, and T. Katzenellenbogen, "Reversible debugging software," Judge Business School, University of Cambridge, 2013.
2. P. Ohmann and B. Liblit, "CSIclipse: presenting crash analysis data to developers," in *Proceedings of the on Eclipse Technology eXchange*, 2015, pp. 7-12.
3. Undo Software, "Increasing software development productivity with reversible debugging," Technical Report No. 1, white paper, 2014.
4. I. Alazzam and K. Nahar, "Combined source code approach for test case prioritization," in *Proceedings of International Conference on Information Science and System*, 2018, pp. 12-15.
5. Juliet Test Suite for C/C++, https://samate.nist.gov/SRD/testsuite.php.
6. M. Beller, N. Spruit, and A. Zaidman, "How developers debug," *PeerJ Preprints*, Vol. 5, 2017, Art. No. e2743v1.
7. Microsoft, "Overview of memory dump file options for Windows," https://support.microsoft.com/en-us/help/254649/overview-of-memory-dump-file-options-for-windows.
8. K. Glerum, K. Kinshumann, S. Greenberg, G. Aul, V. Orgovan, G. Nichols, D. Grant, G. Loihle, and G. Hunt, "Debugging in the (very) large: ten years of implementation and experience," in *Proceedings of ACM SIGOPS 22nd Symposium on Operating Systems Principles*, 2009, pp. 103-116.
9. J. Seo, S. Lee, and T. Shon, "A study on memory dump analysis based on digital forensic tools," *Peer-to-Peer Networking and Applications*, Vol. 8, 2013, pp. 694-703.
10. J. Clause and A. Orso, "A technique for enabling and supporting debugging of field failures," in *Proceedings of the 29th International Conference on Software Engineering*, 2007, pp. 261-270.
11. Apple Crash Reporter, https://developer.apple.com/library/archive/technotes/tn2004/tn2123.html.
12. Microsoft Windows Error Reporting, https://docs.microsoft.com/en-us/windows/desktop/wer/about-wer.
13. Mozilla Crash Reports, https://crash-stats.mozilla.com.
14. Microsoft WinDbg, https://docs.microsoft.com/en-us/windows-hardware/drivers/debugger/debugger-download-tools.
15. W. Cui, M. Peinado, S. K. Cha, Y. Fratantonio, and V. P. Kemerlis, "RETracer: triaging crashes by reverse execution from partial memory dumps," in *Proceedings of the 38th International Conference on Software Engineering*, 2016, pp. 820-831.
16. R. Wu, H. Zhang, S.-C. Cheung, and S. Kim, "Crashlocator: Locating crashing faults based on crash stacks," in *Proceedings of International Symposium on Software Testing and Analysis*, 2014, pp. 204-214.

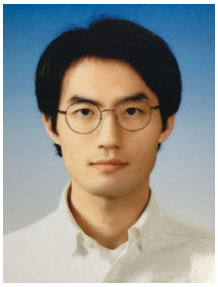
17. P. Broadwell, M. Harren, and N. Sastry, "Scrash: a system for generating secure crash information," in *Proceedings of the 12th USENIX Security Symposium*, Vol. 12, 2003, pp. 273-284.
18. Programdatabase file, https://docs.microsoft.com/en-us/cpp/build/reference/pdb-use-program-database?view=vs-2017.
19. A. Ganapathi and D. Patterson, "Crash data collection: a Windows case study," in *Proceedings of International Conference on Dependable Systems and Networks*, 2005, pp. 280-285.
20. Processhacker, https://github.com/processhacker/processhacker.
21. Navicat-keygen, https://github.com/DoubleLabyrinth/navicat-keygen.
22. idenLib, https://github.com/secrary/idenLib.
23. KernelModeMonitor, https://github.com/alex9191/KernelModeMonitor.
24. Sandbox-Detection, https://github.com/MojtabaTajik/Sandbox-Detection.
25. TS-Security-Editor, https://github.com/aurel26/TS-Security-Editor.
26. ReadStringsFromRegistry, https://github.com/GiovanniDicanio/ReadStringsFromRegistry.
27. FileI/O, https://github.com/bokiex/File-I-O.
28. Cdr2pdfviewer, https://github.com/stxh/cdr2pdfviewer.
29. Mynote, https://github.com/richb255/mynote.
30. CrashRpt, https://sourceforge.net/projects/crashrpt.
31. J. Xu, D. Mu, X. Xing, P. Liu, P. Chen, and B. Mao, "POMP: postmortem program analysis with hardware-enhanced post-crash artifacts," in *Proceedings of the 26th USENIX Conference on Security Symposium*, 2017, pp. 17-32.
32. R. Wu, M. Wen, S. Cheung, and H. Zhang, "ChangeLocator: locate crash-inducing changes based on crash reports," *Empirical Software Engineering*, Vol. 23, 2018, pp. 2866-2900.
33. J. Xu, D. Mu, P. Chen, X. Xing, P. Wang, and P. Liu, "Credal: Towards locating a memory corruption vulnerability with your core dump," in *Proceedings of ACM SIGSAC Conference on Computer and Communications Security*, 2016, pp. 529-540.
34. D. Devi and S. Nandi, "PE file features in detection of packed executables," *International Journal of Computer Theory and Engineering*, Vol. 4, 2012, pp. 476-478.
35. S. Grekhov, J. Jeong, and M. Levin, "On reducing of core dump file size," in *Proceedings of IEEE EuroCon*, 2009, pp. 1288-1292.



Dongmin Jang (張東珉) received the B.S. degree in Computer Engineering from Seowon University, Korea, in 2018. He is currently an M.S. candidate in the Department of Electrical and Computer Engineering at Sungkyunkwan University, Korea. His research interests include software testing, and test automation.

Sujune Lee (李壽峻) received the B.S. degree in Electronic Engineering from Kyonggi University, Korea, in 2016, M.S. degree in Electrical and Computer Engineering at Sungkyunkwan University, Korea, in 2019. He is currently a Researcher at Korea Electronics Technology Institute (KETI), Seongnam, Korea. His research interests include fault localization and software testing.

Yoowon Jang (張由源) received the B.S. degree in Software from Sungkyunkwan University, Korea, in 2017, M.S. degree in Software Platform at Sungkyunkwan University, Korea, in 2019. He is currently a Researcher at Fasoo, Seoul, Korea. His research interests include automated program repair and genetic programming.



Hohyeon Jeong (鄭昊鉉) received the B.S. and M.S. degrees in Electrical and Computer Engineering from Ajou University, Korea, in 2012 and 2014, respectively. He is currently Ph.D. candidate in Electrical and Computer Engineering at Sungkyunkwan University, Korea. His research interests include self-adaptive software systems and test automation.

Eunseok Lee (李殷碩) received his Ph.D. and M.S. degrees in Information Engineering from Tohoku University, Japan, in 1992 and 1988, respectively, and a B.S. degree in Electronic Engineering from Sungkyunkwan University, Korea, in 1985. He was an Assistant Professor in the Department of Information Engineering of Tohoku University, Japan. He was a Research Scientist in the Information and Electronics Laboratory of Mitsubishi Electric Corporation, Japan, from 1992 to 1994. He is currently a Professor with the Department of Computer Engineering, Sungkyunkwan University. His current research topics include automated software testing, debugging, and automatic program generation and repair.