Structured Data and Source Code Analysis for Financial Fraud Investigations

Joe Sremack
Financial and Enterprise Data Analytics
FTI Consulting
Washington, DC USA
joseph.sremack@fticonsulting.com

***Abstract*— Financial fraud investigations are becoming increasingly complex. The volume of data continues to increase along with the volume and complexity of underlying source code logic. As the volume and complexity increase, so too does the importance of identifying techniques for reducing the data to manageable sizes and identifying fraudulent activity as quickly as possible. This paper presents how to ensure that all data was properly collected and a methodology for reducing the complexity of such investigations by identifying similarities and differences between the source code and structured data.**

Keywords-structured data analysis, source code review, fraud investigation.

I. INTRODUCTION

Structured data and proprietary source code are two of the most critical sources of information for large-scale financial fraud investigations. Proprietary source code is used to execute in-house investment strategies for investment banks, hedge funds, and other financial institutions. In the financial setting, source code review yields information about how the organization carried out its operations. The second source of information, structured data, are an organized form of data in which connected data are stored in a discrete, atomic form. Structured data are continuously generated during the course of business, and most business events and transactions create structured data that chronicle the organization's history – most notably the financial transactions. The most common form of structured data is data stored in databases. Together, structured data analysis and source code review can reveal how a business operated in a way that is not possible with only one method.

Financial fraud investigations introduce unknowns about the quality and completeness of both the source code and structured data that were produced. Since most transactions are generated from automated events from the source code, analyzing both in junction with each other is critical for not only validating the completeness of both, but also how the organization truly operated. Falsifying both the transactions and the source code together so that there are no discrepancies is infeasible for virtually any type of fraud. The complexity of synchronizing the source code and the structured data, while carrying out the fraud and creating falsified financial reports, is beyond the capabilities of even the best fraud operatives.

The complexity of synchronizing source code and structured data is what makes analyzing both together so critical. Several failed attempts at uncovering financial frauds have demonstrated that merely analyzing the transactions is not enough, the most famous of which is the Bernard Madoff Ponzi scheme scandal [1]. Analyzing structured data alone does not necessarily tell you how the data entered the data repository or what data was excluded, modified, or code-generated. Likewise, analyzing the source code alone does not provide sufficient evidence, since the source code does not necessarily contain information on what steps were actually run and when, nor the extent of the fraud. The source code would most likely have parameters and input data passed into it, and the data could be altered outside of the source code environment. Combining source code review with the structured data analysis identifies data points and values that could not be generated from a normal, non-fraudulent course of business, such as account values and financial charges.

Analyzing source code in a fraud investigation setting is a complex and time-consuming process. A fraud investigation typically hinges on identifying key anomalous transactions or data patterns that diverge from normal business operations and then identifying how the fraud was conducted within the transactions, source code, and business processes. Most key employees have either been laid off or fired when a financial firm is accused of fraud. As such, information about the source code and locating key documentation is difficult or impossible. The source code may be poorly documented, which requires identifying which files and sections of code need to be reviewed, and volume may be in the tens of thousands of files and millions of lines of code. Reviewing every line of code would not be realistic, regardless of the number of analysts. The culling process of reducing the amount of code that needs to be reviewed requires a precise process that reduces the volume to a manageable size for review but does not exclude key information.

This paper discusses the methodology for performing source code review and structured data analysis that have been applied in several financial fraud investigations. Elements of this methodology have been employed on several large-scale financial fraud investigations. The second section covers the general observations an investigator looks for during the course of this type of investigation. The next section covers the basics of collecting and validating both sources of information. The fourth section discusses data element mapping and function call mapping, which is followed by a section on data value mapping. The concluding section summarizes the process.

II. Current Research

Current research in source code analysis and classification have yielded several techniques that work well under ideal conditions, but those techniques are not always well-suited to real-world fraud investigations. Major research has been conducted on creating dependence graphs and semantic analysis [2][3]. Most of the topical topics and challenges related to source code analysis are based on those outlined in the seminal paper “Reverse Engineering: A Roadmap.” Several techniques, such as island parsing and lake parsing, are better suited for the constraints of a large-scale fraud investigation [5]. Likewise, the field of structured data reverse engineering has produced techniques and a roadmap for analysts [6][7]; however, the need to understand the relationship between the source code and the structured data has not been addressed for practical, complex investigations.

A modern approach to reverse engineering source code is the use of Unified Modeling Language (UML) tools to automatically identify and document source code language constructs, call maps, program behavior, and architecture [8][9]. A common standard that is based on UML has developed called the Knowledge Discovery Metamodel (KDM). The model is based on an Object Management Group Standard that has become an ISO standard in 2012 [10]. Practical objections to UML-based reverse engineering approaches, including KDM, limit the usefulness of using such an approach when time limitations exist and prior knowledge of the relationships of the source code is required [11]. For purposes of a fraud investigation, the total set of source code need not be analyzed, and the analysis should assist with limiting the amount of information that needs to be analyzed. Moreover, the time required for setting up a KDM or other UML-related documentation process with an unknown set of source code can be more time-consuming than operating without any such tool.

Fraud detection and reverse engineering research is a growing field because of the proliferation of cases of fraud and the increasingly complex manner in which they are conducted. The majority of current financial fraud detection literature is based on the analysis of financial transactions using anomaly detection data mining approaches (e.g., Bayesian belief networks, neural networks, and cluster analysis) [12][13][14][15]. The assumption throughout the majority of the literature is that data is pre-cleansed and do not incorporate information about the systems that generated the data.

III. Analysis Motivations

There are several purposes for performing this type of analysis. One purpose is to identify how the source code and structured data relate to one another and whether they both tell the same story about the business operations that were performed. The relationship has many dimensions and attributes and depends on the layout of the structured data and the function(s) of the source code. The relationship depends on how the source code affects the structured data and whether the structured data fully adheres to the rules in the source code. For example, one set of source code can modify all sales transaction fields except for customer service inquiries, so the relationship is defined on those fields based on how the structured data adheres to the rules in the source code. The structured data may vary from the source code rules, and that difference shows where other means for modifying the data exist.

A second purpose is to identify key data points in the structured data. Some structured data contains cryptic field names, obscure data values, and a large volume of objects. Analyzing which data points the source code affects can help with defining what certain fields contain. The source code will show where the outputs of the code are stored within the structured data and which inputs and operations are performed on the data before they are stored. Several examples of these are database inserts, deletes, and updates and the generation of output files that are later stored in the database. The process of identifying which inputs and functions generate which outputs in the structured data can also be performed to pare down the number of fields for analysis. The key fields can be identified by locating the outputs from the source code that either have the input sources or the actions that are critical for the investigation. Likewise, some fields may simply be unimportant for the investigation; identifying the unimportant fields that can be ignored in future analyses is valuable for reducing the complexity of the investigation.

Third, the structured data can be compared to the data modification rules in the source code to detect whether any of the data were modified outside of the source code. In the case of a Ponzi scheme, stock trades may be entered by a non-fraudulent program, while a fraudulent process will later correct those trades to represent that different trades were executed. This is critical in fraud investigations because of the possibility of data modifications that occur outside of normal business processes. For example, a rogue program or manual data manipulation can be used to perpetrate the fraud, and the data modified by a rogue process can be difficult to detect otherwise. Several other possibilities, such as a different version of the code having been run or the code was later modified to appear to be non-fraudulent, can exist, but the comparison of source code rules to structured data will detect these differences regardless.

The fourth purpose is to reduce the number of lines of source file code that need to be analyzed. A common problem in any analysis is reducing the volume of data to a manageable size without compromising the completeness of the relevant analysis. The relevant sections in the source code can be more quickly located by analyzing where the key data fields are manipulated and how. A casual chain can be created from those sections to create a network of related sections, and any isolated sections of code can be more easily categorized or deemed non-relevant.

This paper highlights the key types of analysis techniques that have been applied in practical situations with success. As such, these techniques form an agile toolbox for quickly and effectively analyzing large volumes of source code and structured data in the absence of adequate documentation for financial fraud investigations. While the techniques listed are known in the areas of data and source code reverse engineering, their usefulness and practicality have neither been documented in current research circles, nor presented in a manner in which the combined source code and structured data can reside in a single analysis repository.

IV. Data Collection

The initial steps of any investigation are to collect all data and documentation and verify that the collection is complete. The first step is to survey the data and test that all objects – such as schemas and views – exist in the copied data. Next, comparisons of control totals from the source system to the copied data are performed to provide extra assurance that no data were lost or corrupted. For example, the summation of several numeric columns and the counts of distinct text values in several columns across every table between the source system and copied data are compared and verified. Finally, all available documentation about the source system are collected. The documentation should cover data information (e.g., Entity-Relationship Diagrams, Data Dictionaries, and data value definitions) and business purposes and use (e.g., list of system users and business requirements) [16].

Like structured data analysis, source code review does not begin until after validating the data collection and gathering all supporting documentation. Unlike with structured data analysis, however, ensuring that all source code has been collected is much more complex. Determining if the complete set of source code is available is a critical step that requires reviewing additional documentation, and this step sometimes requires that the source code in question be compiled.

Compiling source code is a dependable method for validating each individual program, for if it does not compile, an issue is known. Compiling another institution's source code is rarely a simple operation, though. Many obstacles make compiling unfeasible, such as compiler settings, compiler versions, and the availability of third-party and custom library files. As a result of these obstacles, compiling the source code is not always possible.

Source code documentation is critical for quickly understanding how the program operates, what functions it serves, the entry points to the source code, and which individual files constitute the complete set of source code. Comments embedded in the source code are valuable, but they rarely provide any insight into the business purpose of the source code or the function call order of the source code. Comments alone also do not offer a reliable method for determining versioning, business purpose, or testing. The following types of documentation are some of the standard documentation that should be collected:

- Compiler logs;
- Source code change history;
- Use case testing documentation;
- Configuration management documentation, and
- Business requirements documentation.

Not all companies fully document all of their source code. Interviews or depositions of programmers and key business owners can greatly assist with understanding how the source code functions and how to analyze the structured data [17]. These interviews, however, cannot always be conducted due to employees refusing to be interviewed or an inability to otherwise question them. These difficulties make the analysis more difficult and are further motivation for why source code and structured data should be analyzed together.

V. Analysis Techniques

The analysis of source code and structured data is an iterative process of conducting a series of related analyses that assist with the investigation. This section details three analysis methods that have been successfully employed on several large-scale financial fraud investigations. The methods are: data element mapping, function call mapping, and data value analysis. Together, the analysis techniques in this section provide a template for streamlining the analysis process and identifying key relationships between the structured data and the source code. As with any analysis, the methods needed for each investigation vary depending on requirements and available information. Moreover, the techniques in this section do not constitute a full methodology for any type of investigation. Standard analysis techniques for the structured data and source code should still be conducted (e.g., structured data surveying and searching the source code for particular functions, such as file printing).

A. Data Element Mapping

Data element mapping is a process for searching the set of source code by the field and object names in the structured data. The process is often referred to as "crawling code," whereby a code base is scanned for particular values or operations [18]. The information gained from this process is a listing of source code files and line numbers where there are search term matches on particular data field and object names. Data element mapping is typically conducted before the other two steps, for it offers both a survey of the source code and data elements, as well as a set of potential entry points for the analysis.

The first step is to collect the set of structured data field and object names that are to be searched. All potentially relevant names should be included, such as:

- The database name;
- Schema names;
- Known database object owners;
- Table names;
- XML tags, and
- Table field names.

Common names that would result in too many false positive keyword search matches should be excluded from the search set, including "date," "ID," "comment," and "owner."

Next, the searching is performed against the source code. The source code should be indexed in a document repository, which is an application that allows for keyword searching and regular expression matching, and the keywords should be searched, with the resulting output containing a list of all keyword matches, the file and line where the match occurred, and the matching line of text from the source code.

The output is then reviewed to compile a list of key source code files. The results should be quantified by matches per keyword, and any keyword with a number of results that are anomalously too high or too low should be reviewed further to ensure that they had proper matches. Too many matches can be explained by a keyword that is often used in the source code language or is an alias that has meaning beyond the structured data object name. Too few matches can sometimes be the result of the field or object name not being explicitly called. Further analysis of the other fields in its table or related objects is required to determine if an alias or function is manipulating that field or object instead. For example a stored procedure within a database may be called instead. Any outliers that are identified that are truly anomalies should be removed from the result set.

The resulting set should be quantified once more to generate a count of search matches per source code file. The source code files with the most search matches are to be the starting points for the investigation. These are the files that generate the most transactional activity and should, at a minimum, be reviewed for the structured data operations. In addition, any source code files that had search term matches for the most critical object or field names are critical sources of information that should be analyzed in depth.

B. Function Call Mapping

The second technique is creating a mapping of how the various elements within the source code relate to one another, which is called function call mapping. The purpose is to identify additional sections of the source code that relate to other critical sections. Function call links from the critical sections of source code from data element mapping or other processes are used to locate additional code that may not have been initially deemed to be relevant. The function call links can be carried out to quickly gain a better understanding of how the source code sections relate to one another and which sections are either isolated or truly not relevant to the investigation.

Performing data element mapping alone may not sufficiently identify all relevant sections of source code. Most enterprise-level applications are designed with layers of abstraction for code reuse and ease of development. This is true for object-oriented programming languages, functional languages, and other modern language types. Relying on a method such as data element mapping will result in the investigator missing the ancillary sections of code that perform data operations. For example, the source code may have a main section of code that explicitly operates on several critical structured data fields, but those records are updated in a secondary portion of the source code that had no search matches in data element mapping.

Function call mapping is performed by identifying all possible mechanisms by which sections of source code can call or reference other sections of the source code and then searching for those keywords and referencing the results. The process is akin to data element mapping in that a set of keywords is created, the source code is searched based on those keywords, and the results are categorized and analyzed.

The first step, identifying all mechanisms for calling or referencing other sections of code, is conducted by creating a list of potential keywords specific to the programming language(s) in the source code and pattern indexes for how to identifying the reference. The investigator creates a list of keywords and regular expressions that will return the command that performed the calling or referencing and the section of code that was called. A common example in the C language is a function. The syntax will remain relatively consistent with parentheses after the function name and possible parameters within the parentheses. There are, however, additional mechanisms for calling or referencing additional sections of code. The source code could be stored as a .h "header" file that is referenced with the #include command. In addition, other code could be compiled and called as an external program via the ShellExecute() command. Regular expressions to store the calling command and referenced section of call need to be generated for any of the identified mechanisms.

The second step is to run the regular expressions from the previous step against the entire set of source code. The output should store the following:

- Calling/referencing mechanism;
- Called/referenced section of source code;
- Source code file where the match occurred, and
- Line of code where the match occurred.

The output from the second step next needs to be analyzed to identify key relationships and remove anomalies. Similar to data mapping, the first step is to quantify which source code objects are referenced most frequently. Examine the results to ensure those results are not false positives due to an incorrect regular expressions or a keyword that is used in other ways in the source code. The same is done for source code files that had too few or no search matches.

The findings from these steps can be shown in various ways. The simplest is simply identifying any sections of source code that are related critical sections of code. The critical code can be source code already known to be critical, results from the data element mapping, or source code that received a large number of search matches in the function call mapping phase. In addition, the findings can be depicted in graphic form to document the process flow for the source code. For example, Figure 1 was generated for a fraud investigation to show which AS/400 libraries either called or were called by the main source code library. Performing this analysis allowed the investigators to generate a map of all possible entry points for the enterprise investment trading platform

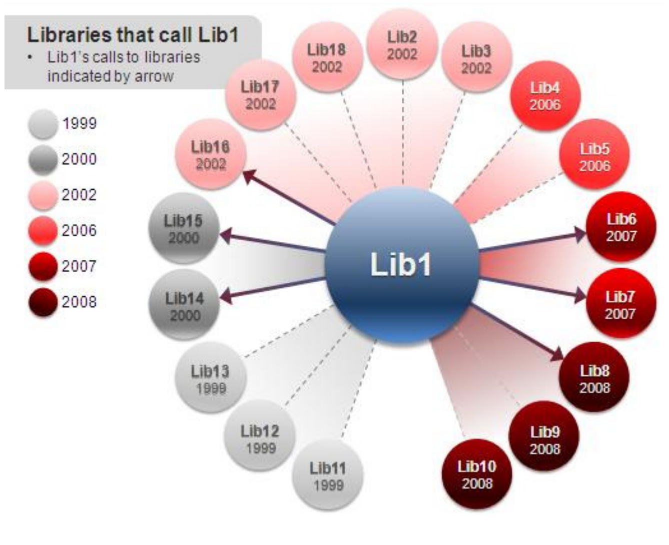

Figure 1. Function call mapping output.

Function call mapping is an iterative process that is typically run multiple times to refine the results. The critical sections of source code may contain additional mechanisms for calling or referencing other sections of source code, which requires the creational of additional regular expressions and searching the source code again. The amount of source code can affect the results, and in order to reduce the volume of search matches, investigators can limit their search population to key source code.

C. *Data Value Analysis*

The source code can be utilized to validate the data stored in the structured data. Most data in structured data repositories arrive via an external source and remained unchanged or were entered through automated logic. Data value analysis is performed to analyze for the latter. Under normal circumstances, structured data that are entered or updated through automated logic should be limited to only the types of values possible in the source code.

Acceptable value analysis is the process by which the constraints from the source code for particular fields are documented and then compared to the values in the structured data. In a fraud investigation, detecting anomalies is a key component for identifying areas in which fraud may have occurred. This analysis is typically performed only against key fields that are altered in the source code. Analyzing non-key fields is typically too time consuming. The process is performed by identifying how the data is altered in the source using the results from the data element mapping result set and then constructing a list of all unique values or patterns from the structured data set. A comparison will either show conformity or some anomaly. Anomalies may be caused by manual updates or data that were never altered by the source code. Both of these conditions require further analysis, as either case can be the result of fraud.

Unaltered field analysis is performed by analyzing all fields from the structured data that did not appear in the data element mapping process. Some fields may legitimately never be altered by the source code; however, if these fields appear in key tables, there is a possibility that that field is being manually updated. This analysis is performed by comparing the full field list for key tables against the data element mapping results for those tables. Any fields that do not appear in the data element mapping result set should be analyzed further.

Date value analysis, the third form of data value analysis, is a multi-step analysis based on either acceptable value analysis or external event analysis aimed at creating a chronology of the structured data and when the collected source code may have been operational. The first step is to identify the date range of records in key tables that have “unacceptable” values if there are any date values stored in the table or a linked, related table. These types of date fields are commonly stored in the table or in an audit table that stores a history of transactions. Since source code versions change over time, having a baseline date range is critical for knowing when the process for storing data in a particular table began. Next, identify any information about the source code version history from embedded comments or related documentation. The investigator should analyze the data for acceptable values from the structured data and find the latest unacceptable value. That result is then compared to the source code change history documentation to identify any possible discrepancies.

External events can be analyzed in conjunction with the date value analysis by examining for higher volume of certain transactions or new transaction patterns. Most financial companies have fixed transaction patterns based on holidays, normal trading hours, and regulatory and legal events. The investigator should create a chronological list of important regulatory and legal events and normal trading dates. The normal trading dates should be compared to transaction volume from the structured data. In addition, that analysis should be compared to the acceptable value analysis to determine if the source code was modified in accordance with the regulatory and legal events. In many fraud cases, the source code and underlying data change dramatically when there is a regulatory or legal threat.

VI. CONCLUSION

Financial fraud investigations are becoming increasingly complex and require practical, agile approaches to reduce the volume to a manageable size. The volume of data continues to increase, as does the volume of underlying source code logic and the variance and interrelationships of the source code. As the volume and complexity increase, so too does the importance of identifying techniques for reducing the data to manageable sizes and identifying fraudulent activity as quickly as possible. Practical limitations make common theoretical approaches, such as semantic analysis, unfeasible and require techniques that do not depend on *a priori* knowledge of the source code and data.

Current research the related fields of data mining, source code reverse engineering, and fraud investigations provide useful tools and techniques that are appropriate for certain applications. Static analysis and UML-based approaches, when employed properly, can yield great results and provide useful insights in an automated fashion. Likewise, statistical data mining techniques allow an analyst to survey large volumes of data and understand the meaning and interrelationships of the data. The difficulty is that many of those techniques do not always have practical value when so many data variables are unknown, documentation is not available, and time constraints exist. Practical techniques where the meaning of the data and source code meet offer an alternative in those cases.

This paper presented the main practical techniques for ensuring that all data were properly collected and three critical methods for reducing the complexity of financial fraud investigations by identifying similarities and differences between the source code and structured data. Proper data collection is the vital first step for ensuring that all information is available for further analysis. By properly collecting and validating the data, one can be confident that the full investigation can commence. Data element mapping is a semi-automated method for reducing the volume of source code that needs to be analyzed when looking for a relationship between the source code and structured data. Function call mapping is a rapid method for identifying relationships between source code files and entry points in the application. Function call mapping is made more powerful by limiting the results to only related source code that have at least one section of code deemed critical to the investigation. Finally, data value analysis is a series of techniques to validate the contents of the structured data and identify potential anomalies.

The nature of financial fraud and the technologies used to conduct them continue to change, and as such, so too will fraud investigations change. Technological advancements and changing business practices, such as cloud computing and offshore data processing, introduce new complexity that will require advanced techniques for identifying critical information. So long as investigators remember to focus on identifying key data relationships and identify anomalies, advanced data analysis and visualization tools and techniques will allow investigators to be able to distill large volumes of information into their critical components and unravel the fraud.

REFERENCES

[1] U.S. Securities and Exchange Commission Office of Investigations, "Investigation of failure of the SEC to uncover Bernard Madoff's Ponzi scheme," August 31, 2009. J. Clerk Maxwell, A Treatise on Electricity and Magnetism, 3rd ed., vol. 2. Oxford: Clarendon, 1892, pp. 68–73.

[2] G. Canfora and M. DiPenta, "New Frontiers of Reverse Engineering," Future of Software Engineering, Future of Software Engineering (FOSE), 2007, pp. 326-341.

[3] A. Yazdanshenas and L. Moonen, "Crossing the Boundaries while Analyzing Heterogeneous Component-Based Software Systems," 27th IEEE International Conference on Software Maintenance, 2011.

[4] H. Mueller, J. Jahnke, D. Smith, M. Storey , S. Tilley, and K. Wong, "Reverse Engineering: A Roadmap," ICSE – Future of SE Track, 2000, pp. 47-60.

[5] L. Moonen, "Generating Robust Parsers using Island Grammars," Proceedings of the Working Conference on Reverse Engineering, 2001, pp. 13-22.

[6] W. Premerlani, M. Blaha, "An Approach for Reverse Engineering of Relational Databases," Communications of the ACM, Volume 37, Issue 5, May 1994, pp. 42-50.

[7] N. Mian, T. Hussain, "Database Reverse Engineering Tools," Proceedings of the 7th WSEAS International Conference on Software Engineering, February 2008, pp. 2006-2011.

[8] S. Rugaber and K. Stirewalt, "Model-Driven Reverse Engineering," IEEE Software Volume 21, 2004, pp. 45-53.

[9] F. Barbier, S. Eveillard, K. Youbi, O. Guitton, A. Perrier, E. Cariou, "Model Driven Reverse Engineering of COBOL," *Information System Transformations: Architecture Driven*

Modernization Case Studies, Mogran Kaufmann, 2010, pp. 283-299.

[10] ISO/IEC 19506:2012, “Information Technology – Object Management Group Architecture-Driven Modernization (ADM) – Knowledge Discovery Meta-Model (KDM), 2012.

[11] R. Kollmann, P. Selonen, E. Stroulia, “A Study on the Current State of the Art in Tool-Supported UML-based Static Reverse Engineering,” Proceedings of the Ninth Working Conference on Reverse Engineering, 2002, pp. 22-32.

[12] I. Jacobson, “Ivar Jacobson on UML, MDA, and the Future of Methodologies,” UML Forum FAQ,

[13] A. Sharma, P. Panigrahi, “A Review of Financial Accounting Fraud Detection based on Data Mining Techniques,” IJCA Journal, Volume 1, 2012, pp. 37-47.

[14] K. Fanning, K. Cogger, R. Srivastava, “Detection of Management Fraud: A Neureal Network Approach,” International Journal of Intelligent Systems in Accounting, Finance, and Management, Volume 4, June 1995, pp. 113-126.

[15] S. Wang, “A Comprehensive Survey of Data Mining-Based Accounting – Fraud Detection Research,” International Conference on Intelligent Computation Technology and Automation, Vollume 1, 2010, pp. 50-53.

[16] E. Kirkos, C. Spathis, Y. Manolopoulos, “Data Mining Techniques for the Detection of Fraudulent Financial Statements,” Expert Systems with Applications, Volume 32, 2007, pp. 995-1003.

[17] J. Sremack, "The collection of large-scale structured data systems," Digital Evidence Magazine, January/February 2012.

[18] L. Hollaar, "Requesting and examining computer source code," in Expert Evidence Report, Vol. 4, No. 9, May 10, 2004, pp. 238-241.

[19] OWASP Foundation, "OWASP code review guide," Version 1.1, 2008, pp 49-50.