2009

# Towards An Automated, Black-Box Method For Reversing Web Applications

Georgios Argyros
*Department of Informatics and Telecommunications, University of Athens*, std06042@di.uoa.gr

Konstantinos Papapanagiotou
*Department of Informatics and Telecommunications, University of Athens*, conpap@di.uoa.gr

# TOWARDS AN AUTOMATED, BLACK-BOX METHOD FOR REVERSING WEB APPLICATIONS

Argyros, Georgios, Department of Informatics and Telecommunications, University of Athens, Panepistimiopolis, Ilisia, 15784, Greece, std06042@di.uoa.gr
Papapanagiotou, Konstantinos, Department of Informatics and Telecommunications, University of Athens, Panepistimiopolis, Ilisia, 15784, Greece, conpap@di.uoa.gr

## Abstract

Analyzing web applications in order to discover possible security vulnerabilities is a complex and challenging procedure that may often produce an increased number of false positives and false negatives. This is mainly due to the fact that most modern websites use dynamic content that may affect the output that web applications produce. In this paper we discuss novel fuzzing techniques that can be used towards providing an automated black-box reversing method for web applications. These techniques focus on detecting changes in dynamic content that can produce false positives. Our goal is to identify different execution paths that an application may follow. Such information on the structure of a web application can provide insight for additional vulnerabilities that would lie undetected if traditional methods for analysis were used.

***Keywords:*** *Web Application Security, Black-Box Testing, Vulnerability Analysis, Fuzzing*

## 1 INTRODUCTION

Web applications can be found nowadays in websites whose main purpose varies from entertainment or social networking to business and professional services. The ease of installation and deployment makes the development and use of web applications very attractive to organizations and web designers. At the same time, due to their nature they suffer from several security issues which are often exploited to result in information leaks. Inevitably, a lot of focus is being drawn to techniques for testing web applications' security.

In general, we can distinguish two types of software testing in terms of vulnerability analysis: black-box and white-box testing. Black-box testing refers to cases where the software's source code is not available, in contrast to white-box testing which is frequently referred to as source code analysis. In the majority of cases web application testing and analysis demands the use of black-box techniques, as no access to the applications source code is given.

While recent developments in the sectors of program verification and analysis (Drewry, 2007) resulted in great improvements on the way testing is performed on programs with access either to the source code (Cadar C., 2006) or the binary (Godefroid, 2008), web application testing remains in a relatively primitive state. Acquiring the binary executable files of a closed source program in order to check for vulnerabilities is often a simple matter. Contrarily, although web applications appear often in various websites, the executables and source code for these applications are usually unavailable to end users. This fact puts penetration testers and application security analysts in the awkward position of having to test a program without access either to the source code or to the binary. As a result the web application testing procedure is usually a time consuming task which mainly consists of checking program outputs that are produced from specially crafted inputs against a known database of vulnerability signatures.

In order to assist in automating this process, a variety of commercial and open source products have been developed, such as Watchfire's Appscan (IBM, 2008), WebInspect (SPI Dynamics, 2008) from SPI dynamics and WebScarab (OWASP, 2008) from OWASP. These programs are known as web application scanners.

Although these products are able to detect the most evident vulnerabilities they do not attempt to analyze the application's structure and thus, they offer poor code coverage and may miss various more complex issues (Petukhov, 2008).

Despite the fact that the implementations of web application scanners vary, these tools are build on some common general principles. Their structure can be divided in two components: the web crawler and the fuzzer (Lanzi, 2007). The web crawler is responsible for discovering all the scripts of the application. Then for each script the web crawler identifies data entry points (DEP), such as variables, cookies and form fields. along with any default values that these may hold. Subsequently, the fuzzer replaces each default value, if available, with a value that may trigger a vulnerability. Then, each response is scanned for signatures that indicate a known vulnerability.

Fuzz testing is a form of black-box testing technique widely used to detect security vulnerabilities in modern software by submitting attack strings and monitoring the program's responses. Even though this approach automates and facilitates the web application testing procedure to a great extent there are some drawbacks that may produce false positives or false negatives. In detail, when scanning large applications and particularly applications with large databases, the input values that will be found for each variable usually have a very large range. However, most of them will correspond to the same execution path being followed by the program. This will result in sending a large number of useless requests that actually correspond to a single execution path. Secondly, some vulnerabilities require crafted input values to be passed into more than one DEP. Trying all possible combinations to all DEPs identified for a particular script, would make the fuzzing process inefficient from a time perspective.

In this paper, we address these issues by proposing automated black-box reversing, a technique for automatically reverse engineering web applications by using fuzzing techniques that analyze the responses to various requests. This novel approach is used to discover changes in the program’s execution flow by analyzing the output that is produced in response to crafted inputs. In detail, we define the problem of automated black-box reversing as: given a set of inputs for a program, we need to determine how many of these inputs follow a different execution path, without any access to the source code or the binary. Our goal is to determine with the greatest possible accuracy the different execution paths that a web application may follow. This insight in the structure of a web application can reveal additional vulnerabilities that would lied undetected if traditional methods of application analysis were used.

The most important challenge in successfully reverse engineering a web application is handling dynamic content. The majority of modern web pages are mainly dynamic; their content and structure may change periodically or according to user input. Additionally we need a way to detect and analyze the conditional branches and loop constructs of the application. The following sections will describe in detail how we intend to deal with these problems.

The proposed, black-box technique is practically based solely on the web application's responses. As a result, we expect that it will probably miss any changes that do not produce any differences in the output. However, this is a black-box approach, meaning that the only available information for the web application that is being examined is the output data that is produced. Consequently, analyzing execution path changes that do not result in any change in the application’s output is doubtful.

The structure of this paper is as follows: in chapter 2 we examine dynamic content, the main reason that produces false positives in web application analysis. We distinguish two different categories of dynamic content and discuss methods for detecting changes in such content. In the next chapter we describe algorithms that can be used to determine the different execution paths within a web application. Finally, we conclude with some remarks and directions for future work.

## 2 DYNAMIC CONTENT

We define dynamic content as parts of a web page that can change dynamically either as a result of changing user input or parameters (e.g. time, date, location). These parts may represent strictly content of a web page, or even elements of its structure. Practically anything that a user may notice as a change in a web page may be characterized as dynamic content, in contrast to static content that remains the same, regardless of user input.

Black-box application analysis is solely based on the various results that different inputs produce when inserted in an application. Consequently, dynamic content in web pages is of great importance when it comes to web application analysis (Artzi, 2008). Differences identified in output that has been produced from a dynamic page may result in false positives if identified as a result of change in execution flow, rather than a change in the content only. Successful identification of such dynamic content is essential for efficient black-box analysis as it will eliminate false positives and will lead to the discovery of additional vulnerabilities. In this section we categorize dynamic content according to the frequency that changes occur and the impact these changes may have on application analysis. Furthermore, we discuss possible methods that can be used to identify dynamic content.

### 2.1 Types and impact of dynamic content

The vast majority of modern web applications are designed not to be static. As a result testing these applications for vulnerabilities can be challenging mainly for two reasons:

- Two identical requests may produce different output without necessarily having followed different execution paths.
- Two different requests may produce identical output, possibly following different execution paths.

In addition, several applications' content changes frequently and sometimes regularly, with data being added or removed, as for example in web applications that are used to broadcast news or weather information. Black-box reversing is based on analyzing responses that may also contain such dynamic content. In order to more efficiently determine the application's structure it is important that dynamic content is properly detected, identified and isolated or even discarded before further analyzing the provided response.

Here, we distinguish two different categories of dynamic content, according to how frequently alterations are observed and, consequently, according to the effect that these alterations can have on black-box application analysis:

- "Minor" dynamic content includes data, attributes or metadata that may be different between two identical requests, or requests that are executed in a short period of time. Such data may be originated from small fields (e.g. fields indicating date, time) or data containers. Usually changes that indicate dynamic content as "minor" are rather frequent and should not be a result of execution path change.
- "Major" dynamic content includes data whose alterations may indicate important changes in the application's execution. Such changes are usually observed less frequently and affect a large amount of variables. These alterations cannot be identified as "minor" as they may often indicate that a different execution path was followed.

In several occasions minor changes may be characterized as major if they have a greater impact on the application data. For instance, a short change in a time variable is considered a "minor" change in most cases. However a change from "23:59" to "0:00" can be indicated as a "major" change as it affects other variables too, such as the "day of week variable".

## 2.2 Detecting and categorizing changes in dynamic content

An automated black-box web application vulnerability scanner may be configured to first identify and discard “minor” dynamic content, and then decide on the importance of “major” dynamic content. This may be implemented for example by conducting several similar or identical requests in a short period of time, in order to determine “minor” changes in dynamic content. Repeating the same requests after a while can point out more important, “major” changes in application data.

Changes that occur in “minor” dynamic content are often regarded as false positives in regard with identifying different execution paths. In detail, identical requests that produce small changes in the output may lead an automated tool or even an analyst to believe that a different execution path was followed. Such minor changes occur frequently and may be due to external factors, such as changes in time, date or input of new content. As a result it is not practical, even for automated tools, to continuously monitor and discard such changes.

However an efficient and rapid method that identifies and discards minor dynamic content is essential for black-box reversing web applications. Even though classic text diff-ing algorithms may be appealing for detecting minor changes between consecutive outputs, they are proved to be inefficient in the context of web application vulnerability scanning (Jung, 2008). This is mainly due to the fact that text differencing algorithms are unable to detect any form of structure in the text. Furthermore, they usually perform a line by line analysis in which a large amount of data that is static has to be discarded. Apparently, a more intelligent method is required in order to properly identify changes in application outputs.

Other suggested solutions include comparing the linear ASCII sum of characters in the response (Hotchkies, 2004). While such techniques may successfully detect the presence of dynamic content they would also discard any form of structure in the produced output and thus, make further analysis inaccurate. We are currently researching some approaches used in the field of program analysis and clone code detection in order to maximize the accuracy of dynamic content detection.

After having eliminated minor dynamic content, it is essential to identify false positives that may occur as changes in major dynamic content. During the application analysis changes in major dynamic content can occur that may be falsely identified as being the result of a change in the execution path. This can be due to the fact that significant amounts of data which are normally infrequently altered have been changed. Normally, discovering a false positive like this would discard all previous findings and the analysis should be started over, taking into account the new information that came up. The execution paths that were previously identified may have been false positives as only changes in major dynamic content have occurred.

In order to detect such infrequent changes that do not indicate a change in the execution path we introduce the concept of guard requests. In regular time intervals during the web application reversing process, each of the discovered execution paths are verified using specific, predetermined inputs. Each time the same, predetermined input is submitted, the same output should be produced, as a specific execution path is followed. This output that corresponds to the predesigned input is recorded and saved for each execution path that has been detected, in order to be verified during future checks. When during such a guard request a change in the output is detected, this will most likely indicate that only a significant amount of major dynamic content has changed, and not the actual execution path.

When such a false positive occurs, the new output has to be recorded in order to replace the previous one. The reversing procedure will have to fall back to the previous guard and discard any execution paths that were found until then, as possible false positives. This is because the change in major dynamic content that was detected may have also triggered other false positives that were not detected until then. As a result we need to fall back to the point in the analysis that this change had not taken place, that is to the previous guard. Thus, only a small part of the analysis will have to be repeated, depending on the period that is set

for performing guard requests. Periodic guard requests may introduce an overhead to the reversing analysis but also provide an efficient mechanism that can effectively reduce false positives.

## 3 DETERMINING EXECUTION FLOW CHANGES

In this section we examine possible methods that can be used to determine changes in the execution path that is being followed by a web application.

### 3.1 Selecting candidate inputs

Generally in black-box reversing, execution path changes can be determined by monitoring different program responses triggered by specially crafted inputs (Sparks, 2007). In this direction many suggestions have been made for selecting inputs that can lead to useful observations and results (Cadar, 2006). Usually these inputs are based to known inputs that have been detected during normal use of web applications.

In addition to detecting execution path changes caused by submitting known input, that is input values that can be found within the application's context, specially crafted inputs could also be used to detect corner side paths where bugs may reside. This section describes some possible options for such inputs.

Obviously, in black-box application analysis, it is highly unlikely to randomly find inputs that match equality conditions in the application's runtime. Moreover, such a brute-force technique will oppose a heavy overhead in the analysis with questionable benefit. Nevertheless, we have identified several values that may produce interesting output results and thus should be tested:

- Submission of very large or very small, and even negative values has a good chance to match inequality conditions.
- Changing variables with values set to known boolean constants such as 0 and 1 to the opposite value is likely to force the program to follow a different execution path.
- String variables can be changed to hold an empty string ("") as web applications usually take special actions for such cases.
- Common variable names and values can be submitted (for example "debug=1"), which can be either selected from a public list, or determined manually by the tester.

Furthermore, this technique could be used in a two-stage fuzzing. During the initial fuzzing stage, selected attack inputs are tested in order to detect if the application takes different execution paths. Afterwards, in a second stage, the analysis can move deeper in the application's code, using additional attack inputs.

### 3.2 Identifying different execution paths

We identify an execution flow change by inspecting both the HTTP header and the output produced by the application.

Generally, changes in HTTP header are a reliable way of judging whether the path has changed as the fields of the header usually change using specific programming functions. Examples in the PHP programming language include the header() function which changes the fields of the HTTP header, and the setcookie() function used to change browser cookies. We accept that changes detected in the HTTP header field are evidence that the submitted input caused a change in the execution path that was followed.

As we have previously described, determining changes based on the application's output is more challenging due to the dynamic content of the websites. Additionally, output that is generated within loops may vary from request to request. In particular, data coming from a database can cause many changes in the application's output without changing the execution path followed. In order to detect database content we use two heuristics:

- Value range: Usually web application's variables that are part of an SQL query are found to have a very large range of values, each of them corresponding to a different record in the database. Thus, if a variable is found to have a large range of values, it is with high probability a part of an SQL query.
- Printing in loop constructs: It is very common that web applications are printing the results from an SQL query within loops, by one record at a time. Thus, we need to detect when an application executes such loops. We further examine this issue of loop analysis in the following section.

Once all dynamic content, minor or major, has been successfully discarded from a response, then we can examine whether the specific response is different from the rest of the output that has been collected so far. Subsequently, if significant differences are found, they can be analyzed in order to determine if they are the result of a loop. Finally, if the loop analysis does not detect any loop then the new response is classified as a different execution path.

### 3.3 Loop analysis

As with the execution flow analysis that we described in the previous section, loop analysis can only be performed when every iteration of the loop construct produces an output. In such case, pattern recognition algorithms can be used in order to further analyze a loop construct. More specifically, we need to determine the output that was produced in different iterations. Apparently, such data will have many similarities in its structure or even content. Thus, a properly designed pattern matching algorithm can be used to detect such similarities in a web application's output. For example, when we identify that a database record is printed, we can then try to match its pattern against the remaining output. If repetitiveness is observed then a loop has been identified. As many algorithms for pattern matching have been proposed (Kontogiannis et al., 1996) the selection of an appropriate method and the study of its effectiveness are left for future work.

## 4 CONCLUSIONS – FUTURE WORK

In this paper we have discussed the issue of automated web application black-box reversing. In this context our main goal is to identify the different execution paths that a web application may follow with the biggest possible accuracy that a black-box analysis may provide. This will enable us to find vulnerabilities and weak points that could not be identified with other traditional methods of web application analysis. Towards this goal we discussed some open challenges and issues and also proposed some theoretical solutions that are based on fuzzing algorithms and can tackle the problems that we identified. More specifically, we indicated the importance of efficiently identifying dynamic content in the output of web applications. We provided some theoretical means to detect dynamic content according to a categorization that we suggested. Furthermore, we proposed algorithms for detecting different execution paths within an application towards a fully automated black-box reversing of web applications.

Although our focus is mainly on web applications, black-box reversing could also be useful to analyzing other applications whose source code or even binary are unavailable. This challenge is encountered frequently during the analysis of remote procedure calls (RPC) or any other not publicly available network application. Moreover the analysis of such programs that operate in network level is theoretically simpler because of the lack of dynamic content which is often encountered in web applications.

Currently, we are implementing the proposed algorithms in order to observe their effectiveness through real life testing. We intend to test them in publicly available and open source web applications that can provide us with input on the accuracy of the discovered execution paths. Various techniques for identifying minor changes in dynamic content are also being evaluated. Finally, different algorithms for pattern matching can be examined in order to determine the most efficient to be used in loop analysis. A properly designed pattern matching algorithm may successfully identify loop constructs and thus provide valuable insight in black-box web application analysis and reversing.

## References


Artzi S., Kieżun A., Dolby J., Tip F., Dig D., Paradkar A., Ernst M. D. (2008). Finding bugs in dynamic web applications, In ISSTA 2008, Proceedings of the 2008 International Symposium on Software Testing and Analysis, pp. 261-272, Seattle, WA, USA.

Cadar C., Ganesh V., Pawlowski P.M., Dill D. L., Engler D.R. (2006). EXE: Automatically Generating Inputs of Death, In Proc. CCS'06, Alexander, VA, USA.

Drewry W., Ormandy T. (2007). Flayer: Exposing Applications Internals, In Proc. First USENIXWorkshop on Offensive Technologies (WOOT '07), USA.

Godefroid P., Levin M. Y., Molnar D. (2008).Automated Whitebox Fuzz Testing, In Proc. 16th Annual Network & Distributed System Security Symposium, San Diego, CA, USA.

Hotchkies C. (2004). Blind SQL Injection Automated Techniques, BlackHat Briefings USA.

IBM (2008). Rational AppScan Developer Edition V7.8, Retrieved May 1, 2009, from http://www.ibm.com/developerworks/downloads/r/appscan/

Jung, J., Sheth, A., Greenstein, B., Wetherall, D., Maganis, G., and Kohno, T. (2008). Privacy oracle: a system for finding application leaks with black box differential testing. In CCS '08: Proceedings of the 15th ACM conference on Computer and communications security, pages 279-288, New York, NY, USA.

Kontogiannis K.A., Demori R., Merlo E., Galler M., Bernstein M. (1996). Pattern Matching for Clone and Concept Detection, Journal of Automated Software Engineering, Kluwer Academic Publishers, Vol. 3. pp.77-108.

Lanzi A., Martignoni L., Monga M., Paleari R. (2007). A Smart Fuzzer for x86 Executables. In Proceedings of the Third international Workshop on Software Engineering For Secure Systems, International Conference on Software Engineering. IEEE Computer Society, Washington, DC, USA.

Martin M., Lam M. S. (2008). Automatic Generation of XSS and SQL Injection Attacks with Goal-Directed Model Checking. 17th USENIX Security Symposium, USA.

OWASP (2008). The OWASP WebScarab Project, Retrieved April 22, 2009, from http://www.owasp.org/index.php/Category:OWASP_WebScarab_Project

Petukhov A., Kozlov D. (2008). Detecting Security Vulnerabilities in Web Applications Using Dynamic Analysis with Penetration Testing, In Proc. OWASP Application Security Conference 2008, Ghent, Belgium.

Sparks, S., Embleton, S., Cunningham, R., and Zou, C. (2007). Automated Vulnerability Analysis: Leveraging Control Flow for Evolutionary Input Crafting. In Computer Security Applications Conference, 2007. ACSAC 2007. Twenty-Third Annual, pages 477-486.

SPI Dynamics (). WebInspect 8.0, Retrieved April 10, 2009, from https://download.spidynamics.com/webinspect/default.htm