SPI DYNAMICS

Start Secure. Stay Secure.™

Hybrid Analysis

An Approach to Testing Web Application Security

By SPI Labs

Table of Contents

Hybrid Analysis

Abstract

Testing Web applications for security defects is now considered a necessary part of the development process. However, none of the traditional methods of automated security testing provides comprehensive security coverage and accurate results for Web applications. While source code analysis is capable of finding insecure programming practices that have potentially rendered the code vulnerable to malicious attacks, it can be limited by the types of languages that have been utilized in crafting the Web application and can only find potential vulnerabilities rather than actionable results. While black box testing techniques are beneficial because they eliminate language dependency and the need for parsing the source or binary code into an analyzable form, they are also limited by the fact that do not have access to the source code, and if unable to "guess" where some pages or files are located, can provide a false sense of security by producing numerous “false negatives”. Only an approach that combines the strengths of both source code analysis and black box testing can be used to produce secure Web applications. This hybrid analysis approach can provide broad code coverage, identify all points of input to an application, track data as it moves through an application, and then validate the vulnerabilities it does find, ultimately resulting in more accurate results. Developers should look toward hybrid analysis tools that combine the depth of source code analysis with the accuracy of black box testing to help them secure code more easily and confidently.

Introduction

As reliance on the Web as a conduit for business processes and as a method for conducting day to day transactions has grown tremendously in the past few years, there has also been a corresponding growth in the number of attacks upon Web applications. After an initial delay in responding to this threat, developers are now aware of the need for their Web applications to operate securely, and are taking measures to secure their applications before their products are released. A variety of tools have been developed whose sole purpose is to help developers secure their Web application code. Generally, secure development tools fall into two categories: those that perform source code analysis to find common security coding mistakes, and those used to implement black box testing, which gather information about an application via the data that is returned from it much as an actual user (or attacker) would. The tools that are available to support secure development endeavors, however, are still works in progress. While each method of testing has inherent strengths, each method also has significant weaknesses. The aim of this paper is to detail the benefits and limitations of each approach, and describe why an application that utilizes a methodology of hybrid analysis consisting of both approaches is the most effective method of creating secure Web applications in development.

Source Code Analysis

Source code analysis is an approach for verifying application code syntactically and semantically for known vulnerable programming constructs without actually having to execute the code. The main focus of source code

analysis is to ensure that all inputs to an application are verified and that it utilizes its API in a secure manner. The focus of an automated source code security analysis is essentially to find bad programming practices that have potentially rendered the code vulnerable to malicious attacks. Source code analysis can be seen as white box testing or a proactive approach where the application is tested and secured regardless of whether or not it can be practically exploited.

Source code analysis is very effective at enforcing secure coding standards and improving overall code quality and application performance. Some of the security checks that source code analysis is effective at finding are listed below:

- Detecting input data points
- Identifying the use of insecure methods
- Verifying application programming interface (API) calls
- Checking memory allocation
- Identifying misconfiguration
- Identifying critical parameters
- Tracing information flow

The primary requirement of source code analysis is to parse the code correctly to understand the semantics of the code segment. Source code analysis is traditionally performed by the developer during the compile phase utilizing the parsing and analysis steps that are already taking place. It is

also important to verify application output data points. Consider a scenario where the same database is shared between two applications. If one application is successfully attacked and an attacker manages to insert malicious data in the database, the second application retrieves that data and gets exploited even though all its inputs were verified.

Depending on the nature of the programming language used during development, tools can be divided into two categories: compiled languages, and interpreted languages.

Compiled Languages

Compiler Augmentation

Stackguard, stackshield, RATS, and ITS4 are some of the common source code analysis tools for detecting buffer overflows and insecure library function uses in C source code. PMD and FindBugs™ are some of the tools that can be utilized for Java code analysis. Some tools provide support for C# alone, but few tools provide support for all three languages of the .NET Framework (C#, J# and VB).

Finding and fixing potential bugs before an application is executed helps the developer to keep the system inherently secure. It is convenient to define security checks with access to high level code while compiling. Consider an example where you would like to define a rule to verify every SQL query

before it is executed. A simple source code analysis check (as opposed to trying to exploit every parameter and see if it results in database error) can identify all of the relevant SQL statements and wrap them with appropriate validation:

Find: `SQLExecute(SQLqueryString)`

Replace With : `if(Regexobj.Match(SQLqueryString))`

`SQLExecute(SQLqueryString)`

The following graphic represents the typical phases of the compilation process.

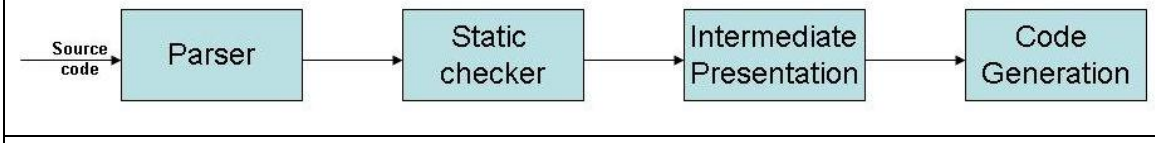

Figure 1 Phases of Compiler

The program code is syntactically and semantically verified in the source code analyzer and represented with an intermediate format. With source code analysis, it is possible to enforce secure programming rules such as naming conventions, variable declaration, and proper usage of library functions during the static checker step.

Limitations of Source Code Analysis

At first glance, source code analysis would seem to be the most suitable approach to enforce secure programming practices. Still, though, researchers have been unable to accurately find and remediate all the vulnerabilities from code statically. Some of the main limiting factors are tracking dynamic binding, dynamically resolved execution paths or interpreting scripting languages. Dynamic binding is a common implementation style especially in object oriented languages where the abstract method or interface methods can have multiple implementations due to polymorphism. Another reason is that a variety of languages are often used to code applications. Source code needs to be parsed before it can be checked and security checks need to be represented with what is available from the parser and compiler tools. This not only delays the process of security support for newer languages, but also makes identifying possible exploits in that language an issue.

A majority of current source code security checks fall into the category of whether a particular method is called or not. Or, whenever a variable is declared, enforcing the type specific range and performing value validation on them, and so forth. The actual vulnerability detection is either left to be discovered during runtime or otherwise not done with accuracy. However, if traditional compiler intelligence is combined with a security analysis to perform interprocedural analysis of the data, a lot more can be achieved. A precise flow of input and all the computation it may affect, be it a SQL query execution or HTTP response construction that can eventually cause SQL

injection or Cross Site Scripting, can only be achieved by incorporating interprocedural data and control flow analysis. Unless these techniques are included, any analysis of how data flows into the system will be incomplete.

Most scripting languages such as PHP, JavaScript, VBScript, Perl and Python are interpreted languages and are loosely data typed. In a loosely typed language the data type is not enforced at the time of compilation, meaning that a defined variable can contain varying types of data at runtime such as an integer or string. This makes it even harder to enforce type and range specific security validation on them. PHP can even accept an array for a scalar value at runtime and cause the application to generate an exception which may not be handled well, causing the application to behave in an unexpected and vulnerable way. An effective method to test this category is still an open research issue because script is integrated in the HTML page and semantics are interpreted depending on the dynamic state at the time that statement is executed.

Black Box Testing

Black box testing analyzes the application from the actual view of the user (and also a potential hacker), and not the developer of the application. The method of gathering information about the application is via the user interface or through requests made directly to the Web server. Runtime and dynamic analysis can be thought of as forms of black box testing. One of the main advantages of black box testing is that it eliminates language

dependency and the need for parsing the source or binary code into an analyzable form. It is also unbiased since the tester can bd independent of the developer or designer of the application. Furthermore, black box testing can also identify vulnerabilities in a third-party component or database code that source code analysis would not uncover since it does not have access to the third-party component's source code.

Wherever there is an opportunity to inject malicious user input, the black box testing application injects it and analyzes the response to take further action. A successful attack gives direct access to the target system but an unsuccessful attack can generate an unhandled exception that may also give an attacker a potential vulnerability to exploit. Many times an unfiltered error message exposes critical information regarding the server and how the application is configured, giving the user access to server code and helping them to inject malicious data in a more focused manner. Guessing every possible input combination is unrealistic, though. It is resource intensive and in reality is ultimately impractical. Consider the Web application form with 15 input fields on it (very common when you are applying for college admissions, or a mortgage application). All the fields are validated but one. The attack is still possible but only if you enter correct data in 14 fields and malicious data in the 15th. There is a fair chance that the vulnerability would go undetected even after multiple attempts. However, a source code analysis tool could prevent that because it finds and wraps every input coming into the Web application into a validation filter.

Dynamic Analysis

Also known as automated penetration or fuzz testing, dynamic analysis occurs when a security tool actively attacks the running application based on thousands of known vulnerabilities and attack patterns. A dynamic analysis tool executes thousands of attacks on the application in a matter of minutes, just as a hacker would over days or weeks. Some of the potential security issues that dynamic analysis is adept at locating follow:

- Exception handling
- Parameter manipulation
- Buffer overflows
- Cookie manipulation
- Session state management
- Proper validation of input

Security defects that are found by fuzz testing techniques are often highly dangerous and readily exploitable vulnerabilities that could be leveraged by actual attackers to inflict critical damage upon an application.

The danger of taking only the dynamic analysis approach is that it can be less thorough than source code analysis because it does not have access to or detailed knowledge of the application source code. Dynamic analysis tools crawl an application like a Web spider to discover all of its pages and files and then uses this site map to direct automated attack attempts. If the tool is unable to "guess" where some pages or files are located, or is blocked by

complex authentication or session management, then it would not be able to effectively attack and assess the security of those hidden resources. The developer can then end up with a false sense of security.

Runtime Analysis

Runtime analysis takes the results of the application execution and analyzes them to find unusual or unexpected responses. These analysis tools simulate a vulnerable input scenario and test the application behavior against it. Some of the Web application vulnerabilities that runtime analysis is effective in detecting follow:

- Command injection
- Server misconfiguration
- Certificate analysis
- Cross-Site scripting
- HTTP compliance
- Encryption strength

As mentioned earlier, script languages or dynamically generated contents have limitations with source code analysis. A runtime approach can unfold what is not known statically.

Limitations of Black Box Testing

The biggest challenge in black box testing comes from predicting the input space for the application. The majority of black box Web application tools send malformed requests or inject malicious data into input fields and then analyze the response. Let's consider an example of SQL Injection. Many companies allow Internet access to archives of their press releases. A URL for accessing the company's fifth press release might look like this:

```
http://www.thecompany.com/pressRelease.jsp?pressReleaseID=5
```

The SQL statement the Web application would use to retrieve the press release might look like this (the client-supplied input is underlined):

```
SELECT title, description, releaseDate, body FROM pressReleases
WHERE pressReleaseID = 5
```

The database server responds by returning the data for the fifth press release. The Web application will then format the press release data into an HTML page and send the response to the client. To determine if the application is vulnerable to SQL injection, an attacker would likely try injecting an extra true condition into the WHERE clause. For example, if you request this URL...

```
http://www.thecompany.com/pressRelease.jsp?pressReleaseID=5 AND
1=1
```

and if the database server executes the following query…

```
SELECT title, description, releaseDate, body FROM pressReleases
WHERE pressReleaseID = 5 AND 1=1
```

…and if this query also returns the same press release, then the application is susceptible to SQL injection. Part of the user's input is interpreted as SQL code. Unless the tool enters the correct input, the vulnerability would be missed.

Hybrid Analysis

It is easy to see the importance of both source code analysis and black box testing when building secure Web applications. It is also clear that neither method by itself offers a complete solution. Source code analysis makes sure that every possible execution path is tested. However, any dynamic change limits how far it can go. Dynamic and runtime analysis make sure that dynamic decisions are not exploitable. However, neither can ensure full code coverage. Black box testing used in conjunction with source code analysis produces much more accurate results than either approach in isolation.

Think of a cross-site scripting vulnerability whereby an attacker is able to embed malicious code into an application and trick a user into executing the code on their own machine. A source code analysis product might be able to identify the potential of a cross-site scripting vulnerability by finding un-validated inputs or poor session handling – if the particular language and compiler is supported. Consider the login page which would be displayed as a result of this URL:

```
http://host/login.php?error=Invalid%20User%20Name:%20Bob
```

The resulting page would display something like this:

Invalid User Name: Bob

If the value of the parameter 'error' was altered to contain JavaScript, and if the input was not validated, numerous malicious tasks could be executed because the user input is dynamically included in the page, as evidenced by the redisplay of the User Name value. Many source code analyzers would recognize that as a potential vulnerability. But what if the user input was stored in a database and then retrieved for processing later? Or what if that input was displayed on a different page than the one on which it was submitted? A source code analysis of that code would likely miss both of those problems. The converse of properly validating input can also be problematic for source code analyzers. There are many possible ways to code for validation of input. For example, developers can set up rules to define accepted characters, or set up operations to convert all input into their equivalent HTML entities. Source code analysis would have a hard time accounting for all of these methods across all the languages which could be utilized.

Source code analysis can certainly find possible avenues for cross-site scripting, but it has a much harder job validating that an actual vulnerability exists. Efforts can be misdirected or wasted when developers spend time fixing a potential vulnerability that in reality is not even exploitable in the

application. A hybrid analysis tool, which will know about the cross-site scripting possibility from an analysis of the source code, will target this potential vulnerability during the dynamic analysis phase to accurately verify whether the page is exploitable by attempting to hack it. For a simple example, a black box testing tool might submit the following script to the login form in the example listed earlier:

```
<script>alert<'XSS'></script>
```

By submitting executable script, the tool can determine that the potential vulnerability identified during the source code analysis is an actual, exploitable vulnerability. However, only a combination of source code analysis and black box testing would find this vulnerability if the information was stored in a database and processed later, or otherwise executed on a different page. Hybrid analysis would first find the potential vulnerability, and then verify it by tracking the submitted data throughout the application.

Conclusion

Source code analysis tools and black box testing alone are only a partial solution when developing secure web applications. Therefore, we would suggest that a hybrid approach would be the best path to take when testing a Web application for potential security issues. A hybrid analysis approach combines black box testing with source code analysis to reduce false positives and find more vulnerabilities during development. The source code

Hybrid Analysis

analysis phase discovers the application's attack surface and identifies all potential vulnerabilities. The black box testing phase uses the intelligence gleaned from the source code analysis to execute a series of attacks using automated hacking techniques that eliminate false positives and yield the actual exploitable security vulnerabilities in the application. Only an approach that combines the strengths of both source code analysis and black box testing can be used to produce secure Web applications. Developers should look toward hybrid analysis tools that combine the depth of source code analysis with the accuracy of black box testing to help them secure code more easily and confidently.

About SPI Labs

SPI Labs is the dedicated application security research and testing team of SPI Dynamics. Composed of some of the industry's top security experts, SPI Labs is specifically focused on researching security vulnerabilities at the Web application layer. The SPI Labs mission is to provide objective research to the security community and give organizations concerned with their security practices a method of detecting, remediating, and preventing attacks upon the Web application layer.

SPI Labs' industry leading security expertise is evidenced via continuous support of a combination of assessment methodologies used to produce the most accurate web application vulnerability assessments available. This direct research is utilized to provide daily updates to SPI Dynamics' suite of security assessment and testing software products. These updates include new intelligent engines capable of dynamically assessing web applications for security vulnerabilities by crafting highly accurate attacks unique to each application and situation, and daily additions to the world's largest database of more than 5,000 application layer vulnerability detection signatures and agents. SPI Labs engineers comply with the standards proposed by the Internet Engineering Task Force (IETF) for responsible security vulnerability disclosure. Information regarding SPI Labs policies and procedures for disclosure are outlined on the SPI Dynamics Web site at: http://www.spidynamics.com/spilabs.html.

About SPI Dynamics Incorporated

Start Secure. Stay Secure.

Security Assurance Throughout the Application Lifecycle.

SPI Dynamics' suite of Web application security products help organizations build and maintain secure Web applications, preventing attacks that would otherwise go undetected by today's traditional corporate Internet security measures. The company's products enable all phases of the software development lifecycle to collaborate in order to build, test and deploy secure Web applications. In addition, the security assurance provided by these products help Fortune 500 companies and organizations in regulated industries — including financial services, health care and government — protect their sensitive data and comply with legal mandates and regulations regarding privacy and information security. Founded in 2000 by security specialists, SPI Dynamics is privately held with headquarters in Atlanta, Georgia. For more information, visit www.spidynamics.com or call (678) 781-4800.

Contact Information

SPI Dynamics
115 Perimeter Center Place
Suite 1100
Atlanta, GA 30346

Telephone: (678) 781-4800
Fax: (678) 781-4850
Email: info@spidynamics.com
Web: www.spidynamics.com