GIAC
CERTIFICATIONS

Global Information Assurance Certification Paper

Copyright SANS Institute
Author Retains Full Rights

This paper is taken from the GIAC directory of certified professionals. Reposting is not permited without express written permission.

Interested in learning more?

Check out the list of upcoming events offering
"Implementing and Auditing CIS Controls (Security 566)"
at http://www.giac.org/registration/gccc

Increase the value of static code analysis by enhancing its rule set.

GIAC (GCCC) Gold Certification

Author: Michael Matthee, michael.h.matthee@protonmail.com
Advisor: David Hoelzer

Accepted: November 2017

Abstract

Static analysis tool vendors are debating whether to allow their customers a rule-set tailored to their environment. There is no empirical evidence to support each argument or counter-argument. Veracode does not accept custom rules and argues that lock-down is in their customer's best interest. Checkmarx enables their customer to customize a rule-set under very special license agreements, while open-source tools such as SonarQube allow for complete customization. Putting vendor concerns and priorities aside, should the enterprise add a tailored rule-set – by adding rules that enforce its secure coding standards – too? More importantly, does a tailored rule-set increase the value of static code analysis to the business? In this study, four different static analysis tools – Veracode, IBM AppScan, Burp Proxy Scanner and SonarQube – scan a JavaScript application. After showing the limitations of the default rule-set for each scanner, the research study adds rules that cover the distinct design and coding standards of the sample application. It is not possible to add a custom rule-set to every scanner. For that reason, the experiment adds the tailored rule-set to the SonarQube platform and combines the results of the two scanning tools: the one tool enforces security standards while the other finds common flaws in the code. While prior research shows that combining the strengths of multiple code analysis tools deliver better results in general, this research study proves that a tailored rule-set improves the outcome even more. The research undertaking recommends practical steps to increase the coverage of automated static analysis and maximize its value to the enterprise.

© 2018 The SANS Institute, Author Retains Full Rights

© 2018 The SANS Institute Author retains full rights.

© 2018 The SANS Institute, Author Retains Full Rights

1. Introduction

Malicious actors, the drivers that motivate them and how they compromise a business, change continuously (Center for Internet Security, 2016). Amidst this changing threat landscape, the Center of Information Security (CIS) maintains a set of technical controls that defend the enterprise from attack. CIS updates these controls as new security incidents and breaches emerge from around the globe. Security controls help a company to reach and can - at times - exceed compliance requirements such as PCI, ISO, HIPAA, ITIL, and NIST. Above all, the Center for Information Security (2017) claims that the first five critical controls alone will stop approximately 85% of attacks in the real world.

One CIS technical security control deals with securing application software and code review is an essential piece (Center for Internet Security, 2016). Developers could peer review the code manually; however, the idea of the CIS framework is that an enterprise will do this automatically and on a continuous basis (Center for Information Security, 2017). A static analysis tool (or SAST) can achieve this ideal. A SAST tool analyzes source code, bytecode, and binaries in a non-running state to find potential security vulnerabilities within a code-base. Common SAST tools include Veracode, IBM AppScan, Burp Static Scanner, Checkmarx, and SonarQube.

However, a SAST scan cannot discover all the security flaws within a code-base. According to Houser (2014), over 40% of software vulnerabilities originate from a system’s chosen architecture and design. It is also difficult to find flaws within a system’s design and architecture by parsing code only (Chess & West, 2007). Houser (2014) recommends a manual approach to finding flaws in software architecture.

Houser (2014) groups architectural risk analysis according to job function – a function that may be specific to the enterprise. Houser's method then assigns a CWE (Common Weakness Enumeration) risk factor to every job role within the architecture.

© 2018 The SANS Institute Author retains full rights.

Houser's risk analysis method can highlight the following potential flaws in an enterprise architecture:

1. Vulnerable business rules that in turn may lead to improper input validation CWE-20,
2. Weak access controls with ill-defined authorities and responsibilities that may lead to a violation of least privilege CWE-272 or information exposure CWE-200,
3. Incompatible definitions of data, which can lead to the exposure of sensitive data through data queries CWE-202.

Houser (2014) explains how the distinct inner workings of an enterprise and its architecture can lead to security vulnerabilities in software. Security architecture, enterprise architecture and secure coding standards all play a vital role in securing application software – many of them being specific to the enterprise.

To what extent can a static code analysis tool find flaws in a system's architecture? Chess & West (2007) reason that analyzing the risk of system architecture can only be a manual task. However, around the same time, Dalci et al. (2006) proved that custom rules within a SAST tool could embrace security standards unique to an enterprise. Finally, in 2008, Shawky et al. (2008) used a SAST tool to discover the distinct patterns of software design and enterprise architecture in a code-base specifically. Today, several static code analysis tools claim to be industry-specific.

CodeSonar from GrammaTech (GrammaTech CodeSonar®, 2017) is one example of a static code analysis tool that can do this. GrammaTech claims that its static analysis tool ensures that companies follow DO-178C (aerospace), IEC 61508 (industry), ISO 26262 (automotive), and IEC 62304 (medical) standards. For example, the DO-178C compliance document from the Radio Technical Commission for Aeronautics (RTCA, 2011) says that source code must match both the data flow and control flow of its software architecture. To satisfy the DO-178C standard, CodeSonar will parse the source

© 2018 The SANS Institute Author retains full rights.

© 2018 The SANS Institute, Author Retains Full Rights

© 2018 The SANS Institute Author Retains Full Rights

code and then tell the customer whether the program adheres to the intended architecture and standard or not (GrammaTech CodeSonar®, 2017). CodeSonar can also enforce design standards – and if not by default, then at least through customization (GrammaTech CodeSonar®, 2017). Customization plays an important role when laying down architecture and design constraints for a code-base, especially when the needs become enterprise or project specific.

SAST tools can, therefore, find vulnerabilities within a system's design or architecture. However, the potential precision and accuracy of a SAST tool is still an obstacle in the vendor industry. The lack of accuracy and precision in SAST tools arise from a mathematical problem called the halting problem (Hicks, 2016), i.e., one cannot always determine if a computer program and its data input will either halt or run forever. Likewise, when automating code review, an analyzer cannot simulate and calculate all possible outcomes of a program. A scanner does not know what will happen when some arbitrary data is input to the system; there are too many possibilities to calculate. Examples of coding structures that will create such an undecidable outcome include bounds checking for array access, a SQL query constructed from untrusted user input and dereferencing a pointer after releasing it (Hicks, 2016). A static analysis tool cannot report definitively on these coding structures as the code may be vulnerable or secure depending on the state of the program and its data input. An alternative is to enforce a safe coding style, for instance: only allowing prepared SQL statements in the code-base and not allowing developers to build dynamic SQL queries based on user input. Nevertheless, it is impossible to make a perfect static analysis tool due to the halting problem.

In addition to the halting problem, tool vendors are also forced to balance scalability and precision within their product (Hicks, 2016). For a SAST tool to be accurate, it must scrutinize the source code meticulously. During analysis, a SAST tool will consume more resources on a program that is cluttered and difficult to understand

© 2018 The SANS Institute Author retains full rights.

© 2018 The SANS Institute, Author Retains Full Rights

than a code-base that is clean and well-structured (Hicks, 2016). A messy code-base increases the workload for an automated code review to complete. And yet, customers expect their SAST tool to run through vast amounts of code within a reasonable amount of time. SAST tool vendors cannot assume that the code-base is clean. So, to speed up the analysis process, precision and accuracy must give way. However, by tailoring a rule-set to the enterprise, reasonable assumptions about the structure, architecture, and framework of the code-base can be made, including:

1. the language of the software program, e.g., JavaScript
2. the technology framework(s) that the software is using, e.g., AngularJS, NodeJS or Coffee
3. the enterprise may require specific coding guidelines, standards, and design/architecture frameworks, e.g., only allow prepared SQL statements within the source code.

Adopting such criteria for evaluating the code-base may – or may not – increase the speed and precision of the SAST tool. Could custom rule-sets attain both: greater tool precision along with an increase in language and framework coverage?

2. Arguments against customization

One tool vendor that does not allow its clients to customize the default rule-set is Veracode (2014). Veracode prides itself in claiming that it has the lowest number of false positives in the industry. It claims to achieve this by offering a well-balanced rule-set within its platform.

After questioning Veracode about its reasons for not allowing custom rules in its platform, one of its principal solutions architects had the following to say:

© 2018 The SANS Institute Author retains full rights.

"*Allowing developers to write rules can lead to an artificially low false positive rate by providing the ability to suppress findings one at a time, or by writing custom rules. The first approach is labor intensive; the second approach is error-prone (and possibly abuse-prone) and requires developer training or often consulting work from the vendor and must be repeated for each application.*" (Principal Solutions Architect, personal communication, 29 June 2017).

Veracode does not allow anyone to change or adapt its default rule-set – even if this happens within an isolated environment. Veracode believes that its rule engine is too advanced for its customers to enhance and consume responsibly – a client may inadvertently cause damage to herself.

Reducing false positives is a problem inherent in static code analysis tools, but so are false negatives (Chess & West, 2007). According to Dalci & Steven (2006), tool vendors favor general rule-sets – i.e., rules that will minimize error across a vast spectrum of companies and software applications. Tool vendors prefer to scale their engines to a mass audience over being perfect for a select few (Chess & West, 2007). Veracode (2017) claims to have a low false positive rate of only 5% and argues that its competitors lag with a rate of 32% by comparison. Knowing the false positive rate of a tool is useful, but understanding its false negative rate is important too. A false negative rate predicts how many flaws a tool may miss during its scanning exercise. Unfortunately, customers notice false positives more often than their false negative counterparts as a scan will incorrectly report them as security flaws. By comparison, a customer does not receive a list of vulnerabilities that a tool does not detect – you do not know what you cannot see – and neither does Veracode.

3. Arguments in favor of customization

Reaching a vast customer base is a vital piece of marketing, but offering the best rule-set to the distinct characteristics of each client has value too. Not only can custom

© 2018 The SANS Institute Author retains full rights.

© 2018 The SANS Institute Author Retains Full Rights

© 2018 The SANS Institute, Author Retains Full Rights

rule-sets expand the scope and role of static analysis, but custom rules can also increase the precision and accuracy of the tool's results. Over the years, research has favored two techniques to reduce the number of false negatives produced by SAST tools: ranking of security warnings and the use of machine learning models (Zhao, 2016). A third alternative is to tailor custom static analysis rules to the enterprise or project at hand (Dalci & Steven, 2006). Ideally, every company keeps its own set of security standards alongside a wealth of threat and incident data that is unique to its line of business. While tool vendors are unlikely to tailor their rules to suit a client, it could be worthwhile for each enterprise to do so individually.

4. Related research

Recent undertakings attempt to offer benchmarks on the accuracy and precision of static analysis tools in the marketplace. Livshits (2017) from Stanford supports a set of vulnerable Java applications to evaluate the scope and accuracy of static analysis tools and their rule-sets. The SAMATE (2017) initiative at NIST and the Build Security In (2017) program at the Department of Homeland Security have similar aspirations. But, there is no widely accepted yardstick for measuring static analysis tools and its rule-sets (Chess & West, 2007), so research studies extrapolate their results.

From time to time, the NSA uses SAMATE test suites to evaluate SAST tools that are available in the marketplace (National Security Agency Center for Assured Software, 2011; NSA, 2012). Returning to the CIS critical security controls, a perfect SAST tool will automate code review entirely and will not require manual inspection of the code-base at all. Ideally, the SAST tool will find all vulnerabilities – or true positives – within the code-base. NSA calls this metric the *recall rate* which measures how well a SAST tool can detect every known security flaw within a code-base.

The recall rate for five SAST tools analyzed by the NSA during a 2011 study shows that SAST tools do not perform equally well (Center for Assured Software

© 2018 The SANS Institute Author retains full rights.

© 2018 The SANS Institute, Author Retains Full Rights

National Security Agency, 2011). Depending on the language or technology at play, it may be better to combine one or two SAST tools. Data for Table 1 below originates from this NSA research study. It shows the recall rate while pairing the results from five different SAST tools. The NSA study runs SAST scans on a test suite written in Java called Juliet.

Table 1: Results of an NSA study on combining SAST tools to get better recall rates (Center for Assured Software National Security Agency, 2011).

	Tool 1	Tool 2	Tool 3	Tool 4	Tool 5
Tool 1	0.53	0.59	0.77	0.73	0.59
Tool 2	0.59	0.11	0.70	0.52	0.25
Tool 3	0.77	0.70	0.67	0.91	0.80
Tool 4	0.73	0.52	0.91	0.50	0.62
Tool 5	0.59	0.25	0.80	0.62	0.22

The NSA study analyzes the results of five SAST tools; the product names are anonymous and are therefore listed as Tools 1 to 5 within the table. The recall rate in this study range between zero and one with a larger number being better. Blocks colored in grey within Table 1 represent the recall rate for a single SAST tool. Table entries in blue confirm that the pairing of SAST tools improves the final recall rate – i.e., a combined toolset finds more security flaws within the code-base.

The NSA conducts these research studies on an infrequent basis, and they consistently arrive at similar conclusions (NSA, 2012):

1. Different tools have different strengths, particularly when analyzing programming code of a specific language type.

© 2018 The SANS Institute Author retains full rights.

2. Combining tools deliver better results as they can complement one another.

According to the TIOBE (2017) index, the popularity of programming languages fluctuates over time. To what extent can SAST tools complement one another when the target technology and programming language are still emerging and new? What happens when a SAST scanner does not support a new development framework?

5. Research study

The tool vendor must evolve its rule engine to support new languages, frameworks, and technologies over time – and many of these changes can be specific to the business. Adding a rule-set and tailoring it to the secure coding standard, design, architecture and technology of an enterprise, could enhance the results of an overall code scanning exercise.

One technology that undergoes many changes is JavaScript. According to TIOBE (2017), JavaScript has been increasing in popularity over the past few years; more projects are adopting JavaScript today as their primary programming language. JavaScript is a weakly typed language which makes it malleable and hard to scan accurately (Liang, 2014). JavaScript is also dynamic, and it can alter its behavior at runtime – a dangerous characteristic to an unsuspecting victim. Over the years, JavaScript frameworks have appeared to make the language safer and more structured during implementation (TIOBE, 2017). Today, trending frameworks include AngularJS and NodeJS; both are based on a variation of JavaScript called TypeScript.

During this research study, a vulnerable application was built using AngluarJS and NodeJS and then scanned using four different static analysis tools: Veracode, IBM AppScan, Burp Proxy Scanner and SonarQube. The vulnerable application extends an existing NodeJS application from OWASP called NodeGoat (Karande, OWASP, 2017).

© 2018 The SANS Institute Author retains full rights.

© 2018 The SANS Institute, Author Retains Full Rights

NodeGoat forms the bulk of the sample program, and the author increases the attack surface by adding vulnerabilities to a new screen using AngularJS. Many of the AngularJS flaws are subtle, and the SAST tool needs to have a deep understanding of the AngularJS framework to detect them.

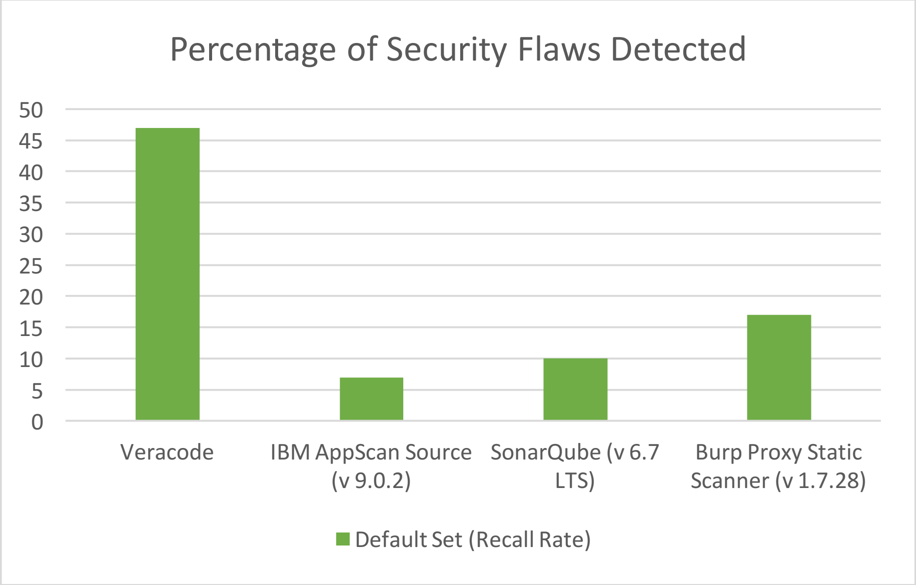


Figure 1: Percentage of flaws found within a vulnerable JavaScript application by each scanning tool.

Figure 1 shows the percentage of flaws found by each scanning tool (see Appendix A for detailed results of this scanning exercise). Of all the SAST tools, Veracode finds the most security flaws in the sample application. IBM AppScan Source performs poorly; however, version 9.0.2 is over two years old, and its support for JavaScript technologies may have improved since. This lack of support reaffirms the need to continuously update and run the latest version of a SAST tool. SAST tools do not support every language and framework that a project may use. The extent to which a scanner understands a programming language may vary too. Table 2 below, lists popular programming languages in the industry and records to what degree each tool supports it.

© 2018 The SANS Institute Author retains full rights.

Table 2: A selection of popular languages and tool support for each (SonarQube, 2017; IBM, 2015; Veracode, 2017). The color red denotes no tool support; yellow indicates some tool support while green marks complete, extensive or unknown version support for a programming language.

	Veracode	IBM AppScan	SonarQube
Java	JRE 1.4 – 1.8	Supported –versions are NA	Good support
Scala	Up to v2.13 & compiled with JavaC 1.6 – 1.8	No support	No support
.NET C#	VS .NET 2003, 2005, 2008, 2010, 2013, 2015, 2017 / Mono 4.x	.NET 2.0, 3.0, 3.5, 4.0, 4.5	Good support. .NET versions not clear.
ASP.NET with C# or VB.NET	VS .NET 2003, 2005, 2008, 2010, 2013, 2015, 2017 – .NET 1.x, 2.0, 3.x, 4.x or core 1.1 for C# only	.NET 2.0, 3.0, 3.5, 4.0, 4.5	Good support. .NET versions not clear.
VB.NET	VS .NET 2003, 2005, 2008, 2010, 2013, 2015, 2017 – .NET 1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.6	.NET 2.0, 3.0, 3.5, 4.0, 4.5	Good support. .NET versions not clear.
JavaScript and TypeScript	Support AngularJS and NodeJS. No support for CoffeeScript nor Dart	No specific support for JS frameworks listed.	No specific support for JS frameworks listed.

© 2018 The SANS Institute Author retains full rights.

© 2018 The SANS Institute, Author Retains Full Rights

		Empirical results show poor support.	Empirical results show poor support.
PHP	5.2 – 5.6	4.x to 5.3	Only ten rules

Table 2 evaluates a sample of programming languages and does not reflect a tool’s complete support profile. However, gaps in the support for these programming languages exist, reducing the ability of SAST tools to capture every flaw that a code-base may have. While analyzing JavaScript code, Veracode does not support CoffeeScript nor does it examine Dart. Both JavaScript technologies, CoffeeScript (2017) and Dart (2017), have been in existence for longer than six years – a long time for SAST tools to add support for them within their rule engines. SAST tool vendors zoom in on their client’s greatest need; this typically involves the more popular language constructs and frameworks in Java. But, new technologies, frameworks, and architectures need new SAST rules to accompany them. An extensive list of supported languages and frameworks for each tool is available online (SonarQube, 2017; IBM, 2015; Veracode, 2017).

Running multiple scanning tools increases the total coverage and likelihood of finding coding flaws (Center for Assured Software National Security Agency, 2011) – the strengths of one tool cover the weak areas of another. This research study benchmarks such tool combinations against a custom rule-set that assumes a set of enterprise security standards.

Figure 2 shows the scan results for various tool and rule-set combinations against the vulnerable JavaScript application. Combining the results of each scanner with that of Burp Proxy improves the outcome, shown as blue bars in Figure 2. Combining the strengths of all four tools (shown as yellow bars in Figure 2) advances the recall rate significantly. However, adding a rule-set that applies a security standard to the project,

© 2018 The SANS Institute, Author Retains Full Rights

© 2018 The SANS Institute Author retains full rights.

improves the result of each scanner drastically too (shown as dark green bars); and in the case of Veracode, it exceeds all other combinations of toolsets. In this study, the tailored rule-set finds another 19% of security flaws when added to the joint strength of four SAST tools – the investment is small but the reward is big. This increase affirms the need for a custom secure coding standard and how such a rule-set deters security flaws that may otherwise occur.

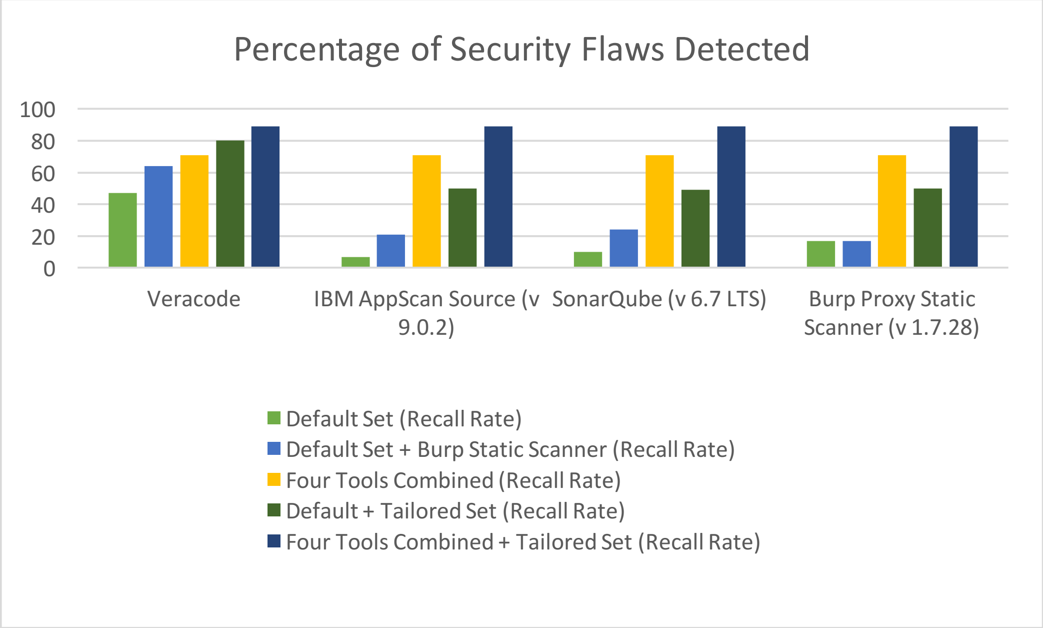


Figure 2: Scan results before and after adding a tailored rule-set.

This research experiment adds a tailored rule-set using the extension framework of SonarQube to find violations of the secure coding standard in the source-code. The experiment adds coding standards that forbid error-prone constructs such as *sce.trustAsHtml* of AngularJS and *window.location* within JavaScript.

A code snippet that violates a secure coding standard does not imply that an exploit or vulnerability exists for each finding; neither is it considered a false positive. However, it does highlight areas in the source code that violate the coding, design, and

© 2018 The SANS Institute Author retains full rights.

© 2018 The SANS Institute, Author Retains Full Rights

architecture standards of its owner. Security standards cover the areas that Veracode misses during its analysis of the vulnerable JavaScript application.

5.1. Adding custom rules

Even though this research proves that an augmented rule-set can improve the value of automated code review in the enterprise, some static analysis tool vendors - including Veracode - do not allow customers to add more rules directly to their engines. In this research study, rules are added to the SonarQube tool to enforce the custom security code standard.

Two options exist while adding custom rules to the SonarQube platform: either extend an existing plugin with more scan rules or develop an entirely new plugin. Both approaches need the API from SonarQube (2017).

To extend the existing JavaScript rule-set, SonarQube needs the SonarJS 3.0 dependency along with the Java SDK in its classpath. After installing Java and adding the SonarJS dependency create a new Java plugin and an entry point, or main class, by implementing the *Plugin* interface as depicted below:

```
public class JavaScriptAngularExtensionRulesPlugin implements Plugin {
        @Override
        public void define(Context context) {
           context.addExtension(JavaScriptAngularExtensionRulesDefinition.class);
        }
```

List this main class within the manifest file of the final plugin artifact so that the SonarQube platform can register it to its list of available extensions.

The next step is to create a repository that holds the custom rules. Creating a new and custom repository enables the scanning platform to apply rules selectively across projects. Tailored rules may increase the accuracy and cover more flaws in one project, but it might not be a good fit for another. Excessive rules can add unnecessary noise and slow down an analysis process. Rule repositories enable one to apply rules selectively,

© 2018 The SANS Institute Author retains full rights.

where proper. Create a new rule repository and give it a name along with a key as shown below.

```
public class JavaScriptAngularExtensionRulesDefinition extends CustomJavaScriptRulesDefinition {
        @Override
          public String repositoryName() {
            return "Extension repository for AngularJS rules ";
          }
```

The key acts as a primary key within the SonarQube platform to distinguish it from other rule repositories. The rule repository class must implement the *checkClasses()* method to return an array of custom rules to add. Using the *$sce.trustAsHtml(…)* Angular method may result in a subtle and complex flaw that a scanner may miss during its scan of the code-base. If the enterprise were to forbid the use of this AngularJS method, then a custom SonarQube rule can enforce this during software development cycles. The code snippet below is the signature of this rule.

```
@Rule(
        key = “A1",
        priority = Priority.MAJOR,
        name = “$sce.trustAsHtml function is forbidden“,
        tags = {“security”}
        )
        @SqaleSubCharacteristic(RulesDefinition.SubCharacteristics.DATA_RELIABILITY)
        @SqaleConstantRemediation("5min")
public class ForbiddenAngularFunctionCheck extends DoubleDispatchVisitorCheck {
```

Logic within the *visitCallExpression* method detect the use of the forbidden method and will flag it when used by developers.

```
@Override
public void visitCallExpression(CallExpressionTree tree) {
        ExpressionTree callee = tree.callee();
        if (callee.is(Kind.IDENTIFIER_REFERENCE) &&
FORBIDDEN_FUNCTIONS.contains(((IdentifierTree) callee).name())) {
```

© 2018 The SANS Institute Author retains full rights.

© 2018 The SANS Institute, Author Retains Full Rights

```
        addIssue(tree, "Do not use the Angular $sce.trustAsHtml function.");
    }
```

By running a custom SonarQube platform in tandem with Veracode, one reaches both a higher recall rate and coverage for enterprise and technology-specific rules. An open-source and free tool, such as SonarQube, can augment and enhance the scanning exercise. A further research study can quantify how the combination of a tailored tool (such as SonarQube) with a reliable and commercial tool (such as Veracode) increases the recall rate for an entire portfolio of projects within an enterprise.

Adding a coding standard within the SonarQube scan and pairing that with a standard Veracode scan improves the result to find many more flaws in the vulnerable JavaScript application. As discussed in Section 4, combining the strengths of two or more SAST tools delivers better results than using a single scanner. However, by adding enterprise-specific rules to the one scanner (e.g., SonarQube) and running that alongside a second scanner (e.g., Veracode), the results exceed those of the Center for Assured Software National Security Agency research study (2011).

5.2. The alternative

During this research study, Veracode was reporting false negatives – i.e., it did not detect all known flaws within the vulnerable JavaScript application. The pre-scanner of Veracode did not link all the application resources correctly causing the tool to miss nine false negatives from the outset. However, credit is due to Veracode consultants who are very helpful and supportive to its customers and particularly during this research study.

Veracode takes its customer base seriously and does listen when there is a need to support an enterprise-specific framework, a new technology framework - such as AngularJS - or a new programming language. Veracode did perform an analysis of the false negatives after the research study was complete, but such assessments take time. Adding framework support can also be a costly and time-consuming exercise. The

© 2018 The SANS Institute Author retains full rights.

© 2018 The SANS Institute, Author Retains Full Rights

© 2018 The SANS Institute Author Retains Full Rights

alternative is to add a second SAST tool in the toolset and tailor its rule-set to cover coding and design standards of the owning enterprise – adding support to scanners such as Veracode in the situations where they fail. As Veracode did not detect 53% of the vulnerabilities in this sample study, it may be prudent to do so.

6. Conclusion

Adding code, design, and architecture standards to an automated code review increases the value of code scanning for the enterprise; SAST tools find more security flaws in the source-code without increasing the false positive rate. Many of these standards are specific to the industry or business, but it requires a scanner to hold a tailored rule-set.

Although existing commercial tools have an expansive set of rules, some tool vendors do not allow customers to add custom rules, e.g., Veracode (2014). Veracode wants to protect the integrity and credibility of its rule-set and not subject it to any wrongdoing (Principal Solutions Architect, personal communication, 29 June 2017; Senior Application Security Consultant, 20 November 2017).

No static code analysis tool is perfect. Most vulnerabilities arise from the specific architecture, design, and business logic of the underlying software system (Houser, 2014; Chess & West, 2007). SAST tools can find flaws in software architecture and design (GrammaTech CodeSonar®, 2017), but prominent scanners like Veracode (2014) do not support this capability. Vendors like Veracode need to balance a low false positive rate along with speed, performance, and quality of results for a broad audience (Hicks, 2016). However, a tailored rule-set can make reasonable assumptions about the structure, architecture, and framework of the source-code to be in line with specific enterprise or industry standards.

© 2018 The SANS Institute Author retains full rights.

© 2018 The SANS Institute, Author Retains Full Rights

Another concern is that SAST tools do not support every language and framework in the software industry (IBM, 2015; Veracode, 2017; SonarQube, 2017). Research studies by the Center for Assured Software National Security Agency (2011) conclude that running multiple SAST tools together increases the recall rate – the strengths of one tool covers the weaknesses of another. However, the four SAST tools used during this research study do not support prominent programming languages in the industry like Go, CoffeeScript, and Dart (SonarQube, 2017; IBM, 2015; Veracode, 2017). Furthermore, many organizations develop frameworks and domain-specific languages that are unknown to the industry at large. Tailoring a rule-set to new or unsupported programming languages extends the reach of code analyzers.

In this research study, four SAST scanners analyze a vulnerable JavaScript application with carefully crafted security flaws. Of the four tools, Veracode delivers the best results, but not a perfect score. Although Veracode has good customer support, it takes time and effort to align a rule-set to the specific needs of an enterprise – and unfortunately in most cases, impossible (Principal Solutions Architect, personal communication, 29 June 2017; Senior Application Security Consultant, 20 November 2017). Moreover, the default rule-set of the tools do not score high enough. In this study, Veracode missed more than fifty percent of the coding flaws – meaning that many flaws may go unnoticed to the unwary customer. Running a second scanner in conjunction with Veracode, one that enforces a custom secure coding standard for the enterprise, is a better alternative.

Research studies by the Center for Assured Software National Security Agency (2011) show that running two or more SAST tools together deliver better results than running only one. This research study confirms that thesis by running four different SAST tools together; showing that the strengths of one tool cover the weaknesses of another. However, this study further extends the research of the NSA by running a second scanner that holds the standards of the owning enterprise in mind. This research study

© 2018 The SANS Institute Author retains full rights.

© 2018 The SANS Institute, Author Retains Full Rights

concludes that adding enterprise-specific rules to the one scanner (e.g., SonarQube) and running that alongside the default wiring of a second scanner (e.g., Veracode), the results exceed those of the Center for Assured Software National Security Agency research study (2011). Running the default rule-set of a static code analyzer is not enough; automated code review using SAST must also include an enterprise security standard. Doing so maximizes the business value of static code analysis and further automates the successful implementation of the CIS critical security control for application security.

© 2018 The SANS Institute Author retains full rights.

© 2018 The SANS Institute, Author Retains Full Rights

References

Chess, B., & West, J. (2007). Introduction to static analysis. In *Secure programming with static analysis.* Upper Saddle River, NJ, United States: Addison-Wesley.

Zhao, P. (2016). Case studies of a machine learning process for improving the accuracy of static analysis tools. Retrieved from University of Waterloo: https://uwspace.uwaterloo.ca/handle/10012/11004

Phegade, R., Jain, R., Randhir, A., & Kadav, P. (2016). Removing web application vulnerabilities with static analysis. *International Research Journal of Engineering and Technology*(3(11)), 1488-1490.

Carnegie Mellon University. (2017, May 22). *Secure coding standards.* Retrieved May 22, 2017, from http://www.cert.org/secure-coding/research/se

OWASP. (n.d.). *OWASP.* Retrieved May 22, 2017, from https://www.owasp.org/

Houser, W. (2014, December 01). *Static Analysis is not enough: The Role of Architecture and Design in Software Assurance.* Retrieved June 14, 2017, from https://www.nist.gov/publications/static-analysis-not-enough-role-architecture-and-design-

Abd-El-Hafiz, S. K., Shawky, D. M., & El-Sedeek, A. (2008). Recovery of object-oriented design patterns using static and dynamic analyses. *International Journal Of Computers & Applications*(30(3)), 220-233.

GrammaTech CodeSonar®. (2017, June 14). Retrieved from http://www.verifysoft.com/en_FDA_standards.html

Livshits, B. (2017, June 14). *Stanford SecuriBench*. Retrieved from https://suif.stanford.edu/~livshits/securibench/

SAMATE. (2017, June 14). *SAMATE - Software Assurance Metrics And Tool Evaluation*. Retrieved from http://samate.nist.gov/

US CERT. (2017, June 14). *Build Security In*. Retrieved from https://www.us-cert.gov/bsi

Michael Matthee,
michael.h.matthee@protonmail.com

© 2018 The SANS Institute, Author Retains Full Rights

SonarQube. (2017, June 14). *Security-related rules - SonarQube Documentation*. Retrieved from https://docs.sonarqube.org/display/SONAR/Security-related

Veracode. (2014, June 6). *Secure Agile Q&A: Scale, Continuous Integration and Policies*. Retrieved from https://www.veracode.com/blog/2014/06/secure-agile-qa-scale-continuous-integration-and-policies

Center for Internet Security. (2016, August 31). *CIS Controls*. Retrieved June 14, 2017, from https://www.cisecurity.org/controls/

Center for Information Security. (2017, July 21). *Mapping and Compliance*. Retrieved July 23, 2017, from https://www.cisecurity.org/cybersecurity-tools/mapping-compliance/

Dalci, E., & Steven, J. (2006). A framework for creating custom rules for static analysis tools. *Proc. Static Analysis Summit* (pp. 49-54). Gaithersburg: NIST. Retrieved from https://samate.nist.gov/docs/NIST_Special_Publication_500-262.pdf

Hicks, M. (2016, August 15). Program Analysis. *Software Security*. College Park, Maryland, United States: University of Maryland.

Karande, C. (2017, October 14). *OWASP_Node_js_Goat_Project*. Retrieved from owasp: https://www.owasp.org/index.php/Projects/OWASP_Node_js_Goat_Project

NSA. (2012, March 29). *SATE4.* Retrieved November 20, 2017, from samate.nist.gov: https://samate.nist.gov/docs/SATE4/SATE%20IV%206%20Stick%20to%20Facts%20II%20Erno.pdf

National Security Agency Center for Assured Software. (2011, June 26). *BH_US_11_WillisBritton_Analyzing_Static_Analysis_Tools_WP.pdf.* Retrieved November 20, 2017, from media.blackhat.com:

© 2018 The SANS Institute Author retains full rights.

https://media.blackhat.com/bh-us-11/Willis/BH_US_11_WillisBritton_Analyzing_Static_Analysis_Tools_WP.pdf

Center for Assured Software National Security Agency. (2011). *CAS Static Analysis Tool Study - Methodology.* Fort George G. Meade: NSA.

TIOBE. (2017, November 21). *tiobe*. Retrieved from tiobe: https://www.tiobe.com/tiobe-index/

Liang, Y. E. (2014). *JavaScript Security.* Birmingham, UK: Packt Publishing.

Karande, C. (2017, November 14). *OWASP*. Retrieved November 22, 2017, from NodeGoat: https://github.com/OWASP/NodeGoat

CoffeeScript. (2017, November 22). Retrieved from http://coffeescript.org/

Dart. (2017, November 22). Retrieved from https://www.dartlang.org/

SonarQube. (2017, November 22). *Extension Guide*. Retrieved from docs.sonarqube.org: https://docs.sonarqube.org/display/DEV/Extension+Guide

Veracode. (2017, October). Veracode Compilation Guide.

IBM. (2015). Installation and Administration Guide. *IBM Security AppScan Source Version 9.0.2*.

© 2018 The SANS Institute Author Retains Full Rights

© 2018 The SANS Institute Author retains full rights.

Appendix A
Detailed Results of Sample Application

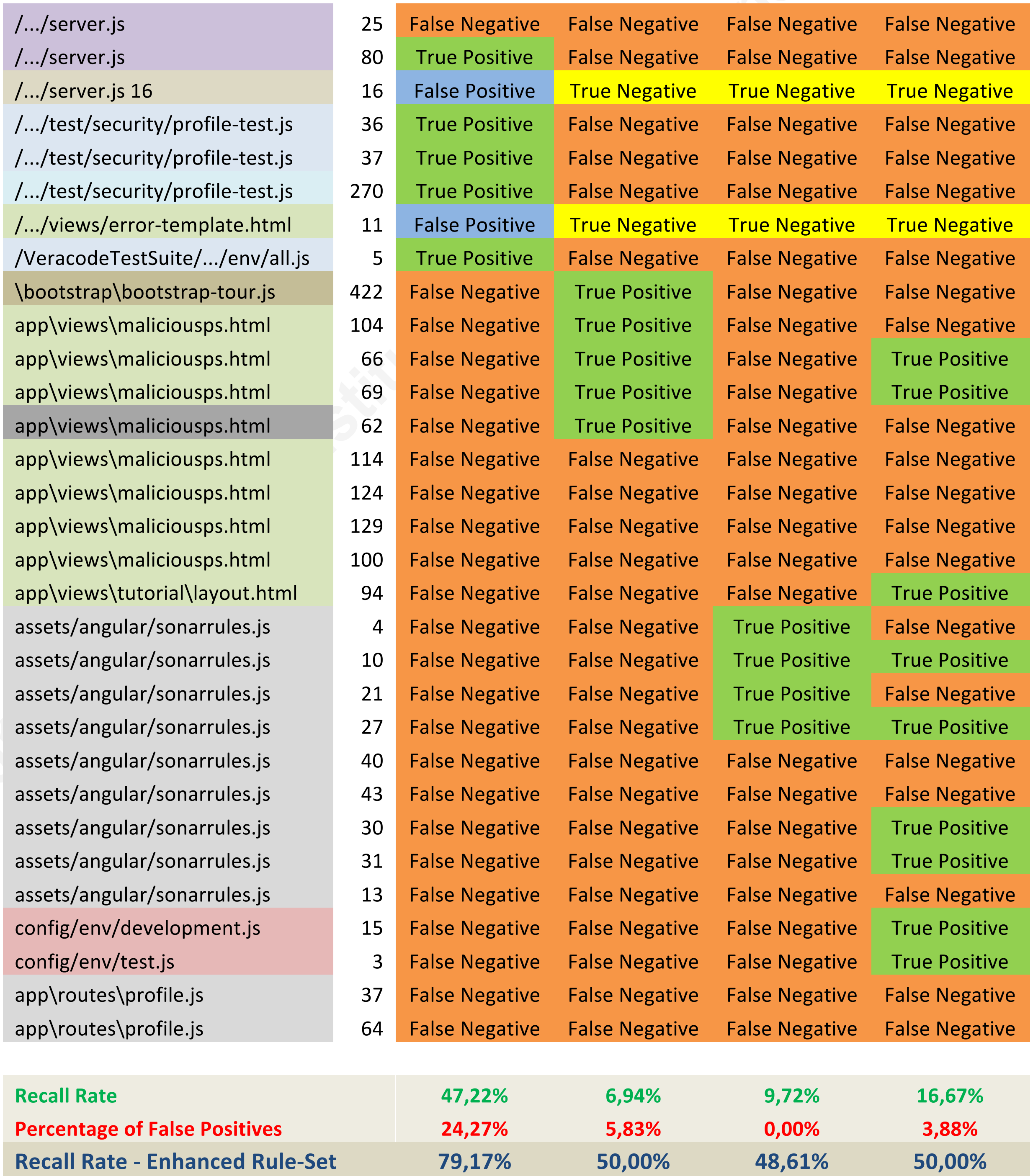
Vulnerability	Line	Veracode	IBM AppScan	SonarQube	Burp Proxy
/.../app/data/allocations-dao.js	79	False Negative	False Negative	False Negative	False Negative
/.../app/data/allocations-dao.js	30	False Positive	True Negative	True Negative	True Negative
/.../app/data/profile-dao.js	71	False Positive	True Negative	True Negative	True Negative
/.../app/routes/contributions.js	24	True Positive	False Negative	True Positive	False Negative
/.../app/routes/contributions.js	25	True Positive	False Negative	True Positive	False Negative
/.../app/routes/contributions.js	26	True Positive	False Negative	True Positive	False Negative
/.../app/routes/index.js	60	False Negative	False Negative	False Negative	False Negative
/.../app/routes/index.js	61	False Negative	False Negative	False Negative	False Negative
/.../app/routes/index.js	82	True Positive	False Negative	False Negative	False Negative
/.../app/routes/session.js	60	False Positive	True Negative	True Negative	True Negative
/.../app/routes/session.js	138	False Positive	True Negative	True Negative	True Negative
/.../app/routes/session.js	189	False Positive	True Negative	True Negative	True Negative
/.../app/routes/session.js	48	True Negative	False Positive	True Negative	True Negative
/.../app/routes/session.js	65	True Negative	False Positive	True Negative	True Negative
/.../app/routes/session.js	73	True Negative	False Positive	True Negative	True Negative
/.../app/routes/session.js	106	True Negative	False Positive	True Negative	True Negative
/.../app/views/allocations.html	15	False Positive	True Negative	True Negative	True Negative
/.../app/views/allocations.html	22	False Negative	False Negative	False Negative	False Negative
/.../app/views/allocations.html	35	True Positive	False Negative	False Negative	False Negative
/.../app/views/allocations.html	39	True Positive	False Negative	False Negative	False Negative
/.../app/views/allocations.html	42	True Positive	False Negative	False Negative	False Negative
/.../app/views/allocations.html	45	True Positive	False Negative	False Negative	False Negative
/.../app/views/benefits.html	20	False Positive	True Negative	True Negative	True Negative
/.../app/views/benefits.html	50	True Positive	False Negative	False Negative	False Negative
/.../app/views/benefits.html	51	True Positive	False Negative	False Negative	False Negative
/.../app/views/benefits.html	52	True Positive	False Negative	False Negative	False Negative
/.../app/views/benefits.html	56	True Positive	False Negative	False Negative	False Negative
/.../app/views/benefits.html	57	True Positive	False Negative	False Negative	False Negative
/.../app/views/contributions.html	20	False Positive	True Negative	True Negative	True Negative
/.../app/views/layout.html	56	True Positive	False Negative	False Negative	False Negative
/.../app/views/layout.html	75	True Positive	False Negative	False Negative	False Negative
/.../app/views/layout.html	126	True Negative	False Positive	True Negative	True Negative
/.../app/views/login.html	101	False Positive	True Negative	True Negative	True Negative
/.../app/views/login.html	110	False Positive	True Negative	True Negative	True Negative

© 2018 The SANS Institute

Author retains full rights.

/.../app/views/login.html	115	False Positive	True Negative	True Negative	False Positive
/.../app/views/login.html	117	False Positive	True Negative	True Negative	False Positive
/.../app/views/login.html (auto)	115	False Negative	False Negative	False Negative	True Positive
/.../app/views/maliciousps.html	100	False Negative	False Negative	False Negative	False Negative
/.../app/views/maliciousps.html	114	False Negative	False Negative	False Negative	False Negative
/.../app/views/profile.html	24	False Positive	True Negative	True Negative	True Negative
/.../app/views/profile.html	41	True Positive	False Negative	False Negative	False Negative
/.../app/views/profile.html	45	True Positive	False Negative	False Negative	False Negative
/.../app/views/profile.html	49	True Positive	False Negative	False Negative	False Negative
/.../app/views/profile.html	53	True Positive	False Negative	False Negative	False Negative
/.../app/views/profile.html	57	True Positive	False Negative	False Negative	False Negative
/.../app/views/profile.html	61	True Positive	False Negative	False Negative	False Negative
/.../app/views/profile.html	66	True Positive	False Negative	False Negative	False Negative
/.../app/views/profile.html	68	False Positive	True Negative	True Negative	True Negative
/.../app/views/signup.html	57	False Positive	True Negative	True Negative	True Negative
/.../app/views/signup.html	74	False Positive	True Negative	True Negative	True Negative
/.../app/views/signup.html	79	False Positive	True Negative	True Negative	True Negative
/.../app/views/signup.html	85	False Positive	True Negative	True Negative	True Negative
/.../app/views/signup.html	90	False Positive	True Negative	True Negative	False Positive
/.../app/views/signup.html	95	False Positive	True Negative	True Negative	False Positive
/.../app/views/signup.html	101	False Positive	True Negative	True Negative	True Negative
/.../app/views/signup.html	103	False Positive	True Negative	True Negative	True Negative
/.../app/views/signup.html	125	True Negative	False Positive	True Negative	True Negative
/.../app/views/signup.html (auto)	90	False Negative	False Negative	False Negative	True Positive
/.../app/views/signup.html (auto)	96	False Negative	False Negative	False Negative	True Positive
/.../artifacts/db-reset.js	15	True Positive	False Negative	False Negative	False Negative
/.../artifacts/db-reset.js	23	True Positive	False Negative	False Negative	False Negative
/.../artifacts/db-reset.js	31	True Positive	False Negative	False Negative	False Negative
/.../artifacts/db-reset.js	18	True Positive	False Negative	False Negative	False Negative
/.../artifacts/db-reset.js	27	True Positive	False Negative	False Negative	False Negative
/.../artifacts/db-reset.js	35	True Positive	False Negative	False Negative	False Negative
/.../data/contributions-dao.js	28	False Positive	True Negative	True Negative	True Negative
/.../data/contributions-dao.js	100	False Negative	False Negative	False Negative	False Negative
/.../data/contributions-dao.js	71	False Negative	False Negative	False Negative	False Negative
/.../data/contributions-dao.js	13	False Negative	False Negative	False Negative	False Negative
/.../data/contributions-dao.js	57	False Positive	True Negative	True Negative	True Negative
/.../server.js	136	True Positive	False Negative	False Negative	False Negative
/.../server.js	21	False Negative	False Negative	False Negative	False Negative

© 2018 The SANS Institute Author retains full rights.

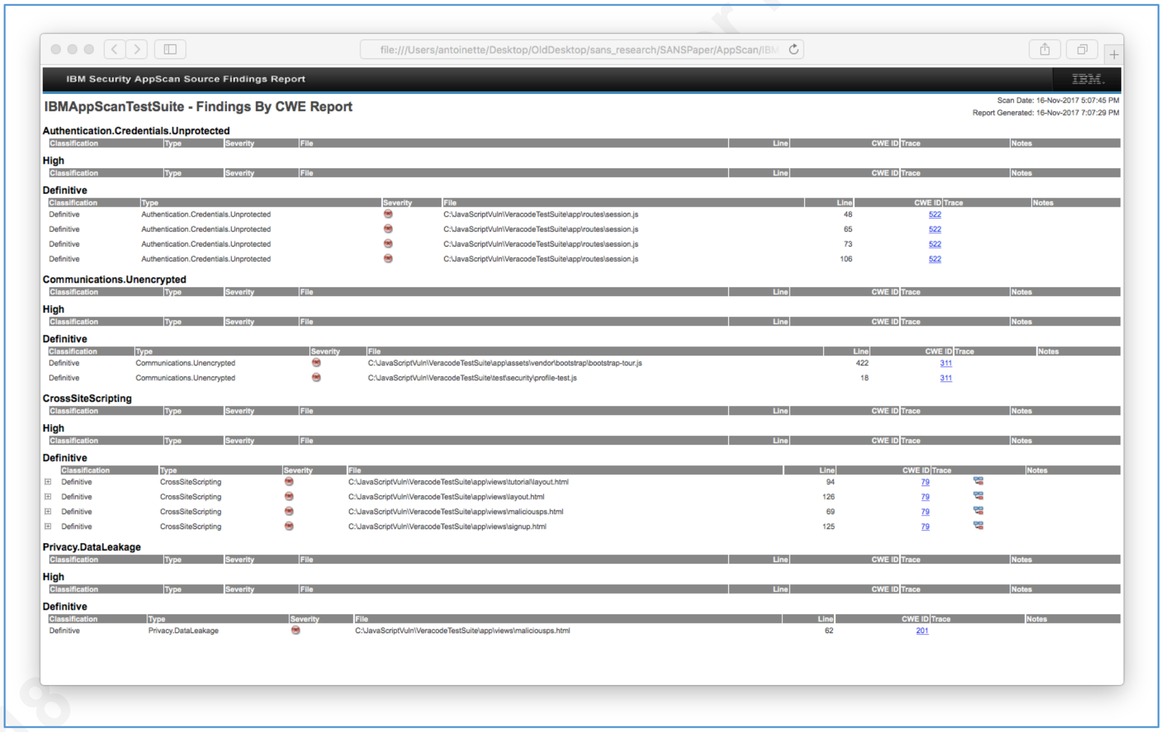


/.../server.js	25	False Negative	False Negative	False Negative	False Negative
/.../server.js	80	True Positive	False Negative	False Negative	False Negative
/.../server.js 16	16	False Positive	True Negative	True Negative	True Negative
/.../test/security/profile-test.js	36	True Positive	False Negative	False Negative	False Negative
/.../test/security/profile-test.js	37	True Positive	False Negative	False Negative	False Negative
/.../test/security/profile-test.js	270	True Positive	False Negative	False Negative	False Negative
/.../views/error-template.html	11	False Positive	True Negative	True Negative	True Negative
/VeracodeTestSuite/.../env/all.js	5	True Positive	False Negative	False Negative	False Negative
\bootstrap\bootstrap-tour.js	422	False Negative	True Positive	False Negative	False Negative
app\views\maliciousps.html	104	False Negative	True Positive	False Negative	False Negative
app\views\maliciousps.html	66	False Negative	True Positive	False Negative	True Positive
app\views\maliciousps.html	69	False Negative	True Positive	False Negative	True Positive
app\views\maliciousps.html	62	False Negative	True Positive	False Negative	False Negative
app\views\maliciousps.html	114	False Negative	False Negative	False Negative	False Negative
app\views\maliciousps.html	124	False Negative	False Negative	False Negative	False Negative
app\views\maliciousps.html	129	False Negative	False Negative	False Negative	False Negative
app\views\maliciousps.html	100	False Negative	False Negative	False Negative	False Negative
app\views\tutorial\layout.html	94	False Negative	False Negative	False Negative	True Positive
assets/angular/sonarrules.js	4	False Negative	False Negative	True Positive	False Negative
assets/angular/sonarrules.js	10	False Negative	False Negative	True Positive	True Positive
assets/angular/sonarrules.js	21	False Negative	False Negative	True Positive	False Negative
assets/angular/sonarrules.js	27	False Negative	False Negative	True Positive	True Positive
assets/angular/sonarrules.js	40	False Negative	False Negative	False Negative	False Negative
assets/angular/sonarrules.js	43	False Negative	False Negative	False Negative	False Negative
assets/angular/sonarrules.js	30	False Negative	False Negative	False Negative	True Positive
assets/angular/sonarrules.js	31	False Negative	False Negative	False Negative	True Positive
assets/angular/sonarrules.js	13	False Negative	False Negative	False Negative	False Negative
config/env/development.js	15	False Negative	False Negative	False Negative	True Positive
config/env/test.js	3	False Negative	False Negative	False Negative	True Positive
app\routes\profile.js	37	False Negative	False Negative	False Negative	False Negative
app\routes\profile.js	64	False Negative	False Negative	False Negative	False Negative
Recall Rate		**47,22%**	**6,94%**	**9,72%**	**16,67%**
Percentage of False Positives		**24,27%**	**5,83%**	**0,00%**	**3,88%**
Recall Rate - Enhanced Rule-Set		**79,17%**	**50,00%**	**48,61%**	**50,00%**

© 2018 The SANS Institute

Author retains full rights.

Appendix B
Summary of IBM AppScan Results



file:///Users/antoinette/Desktop/OldDesktop/sans_research/SANSPaper/AppScan/IBM
IBM Security AppScan Source Findings Report
IBM.

IBMAppScanTestSuite - Findings By CWE Report

Scan Date: 16-Nov-2017 5:07:45 PM
Report Generated: 16-Nov-2017 7:07:29 PM

Authentication.Credentials.Unprotected

Classification	Type	Severity	File	Line	CWE ID	Trace	Notes

High

Classification	Type	Severity	File	Line	CWE ID	Trace	Notes

Definitive

Classification	Type	Severity	File	Line	CWE ID	Trace	Notes
Definitive	Authentication.Credentials.Unprotected	⊗	C:\JavaScriptVuln\VeracodeTestSuite\app\routes\session.js	48	522		
Definitive	Authentication.Credentials.Unprotected	⊗	C:\JavaScriptVuln\VeracodeTestSuite\app\routes\session.js	65	522		
Definitive	Authentication.Credentials.Unprotected	⊗	C:\JavaScriptVuln\VeracodeTestSuite\app\routes\session.js	73	522		
Definitive	Authentication.Credentials.Unprotected	⊗	C:\JavaScriptVuln\VeracodeTestSuite\app\routes\session.js	106	522		

Communications.Unencrypted

Classification	Type	Severity	File	Line	CWE ID	Trace	Notes

High

Classification	Type	Severity	File	Line	CWE ID	Trace	Notes

Definitive

Classification	Type	Severity	File	Line	CWE ID	Trace	Notes
Definitive	Communications.Unencrypted	⊗	C:\JavaScriptVuln\VeracodeTestSuite\app\assets\vendor\bootstrap\bootstrap-tour.js	422	311		
Definitive	Communications.Unencrypted	⊗	C:\JavaScriptVuln\VeracodeTestSuite\test\security\profile-test.js	18	311		

CrossSiteScripting

Classification	Type	Severity	File	Line	CWE ID	Trace	Notes

High

Classification	Type	Severity	File	Line	CWE ID	Trace	Notes

Definitive

	Classification	Type	Severity	File	Line	CWE ID	Trace	Notes
⊞	Definitive	CrossSiteScripting	⊗	C:\JavaScriptVuln\VeracodeTestSuite\app\views\tutorial\layout.html	94	79	⧉	
⊞	Definitive	CrossSiteScripting	⊗	C:\JavaScriptVuln\VeracodeTestSuite\app\views\layout.html	126	79	⧉	
⊞	Definitive	CrossSiteScripting	⊗	C:\JavaScriptVuln\VeracodeTestSuite\app\views\maliciousps.html	69	79	⧉	
⊞	Definitive	CrossSiteScripting	⊗	C:\JavaScriptVuln\VeracodeTestSuite\app\views\signup.html	125	79	⧉	

Privacy.DataLeakage

Classification	Type	Severity	File	Line	CWE ID	Trace	Notes

High

Classification	Type	Severity	File	Line	CWE ID	Trace	Notes

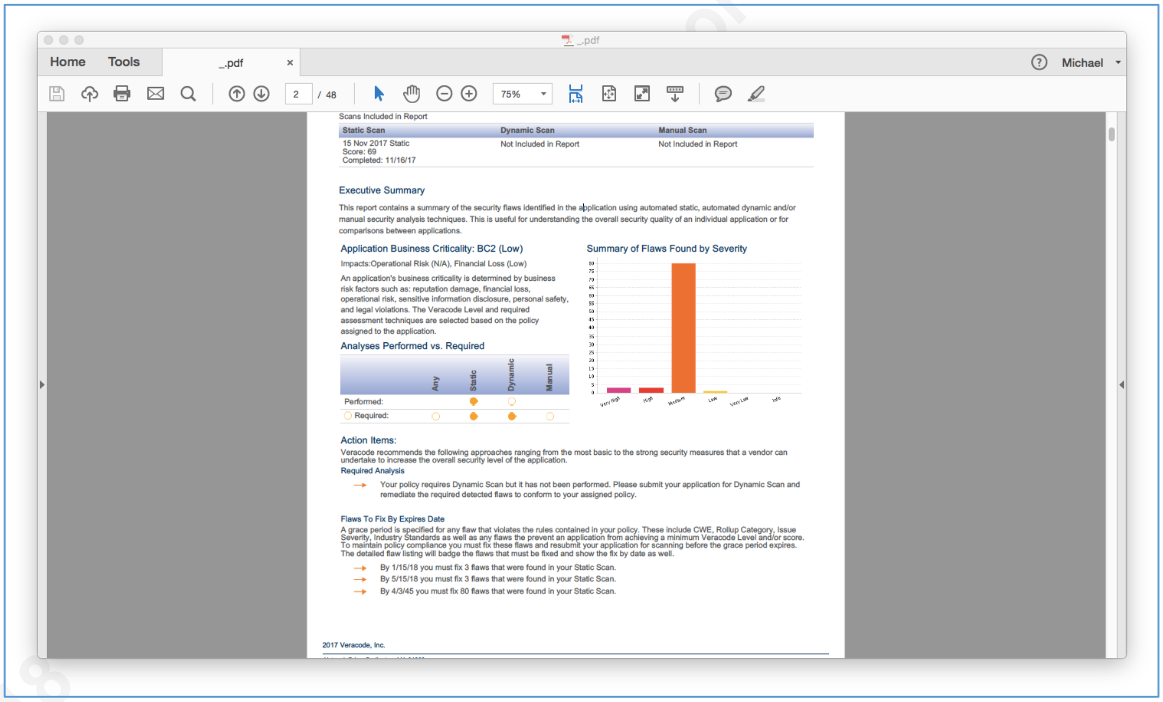
Definitive

Classification	Type	Severity	File	Line	CWE ID	Trace	Notes
Definitive	Privacy.DataLeakage	⊗	C:\JavaScriptVuln\VeracodeTestSuite\app\views\maliciousps.html	62	201		

© 2018 Author Retains Full Rights

© 2018 The SANS Institute
Author retains full rights.

Appendix C
Summary of Veracode Results



Scans Included in Report

Static Scan	Dynamic Scan	Manual Scan
15 Nov 2017 Static Score: 69 Completed: 11/16/17	Not Included in Report	Not Included in Report

Executive Summary

This report contains a summary of the security flaws identified in the application using automated static, automated dynamic and/or manual security analysis techniques. This is useful for understanding the overall security quality of an individual application or for comparisons between applications.

Application Business Criticality: BC2 (Low)

Impacts:Operational Risk (N/A), Financial Loss (Low)

An application's business criticality is determined by business risk factors such as: reputation damage, financial loss, operational risk, sensitive information disclosure, personal safety, and legal violations. The Veracode Level and required assessment techniques are selected based on the policy assigned to the application.

Analyses Performed vs. Required

	Any	Static	Dynamic	Manual
Performed:		●	○	
○ Required:	○	●	●	○

Summary of Flaws Found by Severity

Action Items:

Veracode recommends the following approaches ranging from the most basic to the strong security measures that a vendor can undertake to increase the overall security level of the application.

Required Analysis

→ Your policy requires Dynamic Scan but it has not been performed. Please submit your application for Dynamic Scan and remediate the required detected flaws to conform to your assigned policy.

Flaws To Fix By Expires Date

A grace period is specified for any flaw that violates the rules contained in your policy. These include CWE, Rollup Category, Issue Severity, Industry Standards as well as any flaws the prevent an application from achieving a minimum Veracode Level and/or score. To maintain policy compliance you must fix these flaws and resubmit your application for scanning before the grace period expires. The detailed flaw listing will badge the flaws that must be fixed and show the fix by date as well.

→ By 1/15/18 you must fix 3 flaws that were found in your Static Scan.
→ By 5/15/18 you must fix 3 flaws that were found in your Static Scan.
→ By 4/3/45 you must fix 80 flaws that were found in your Static Scan.

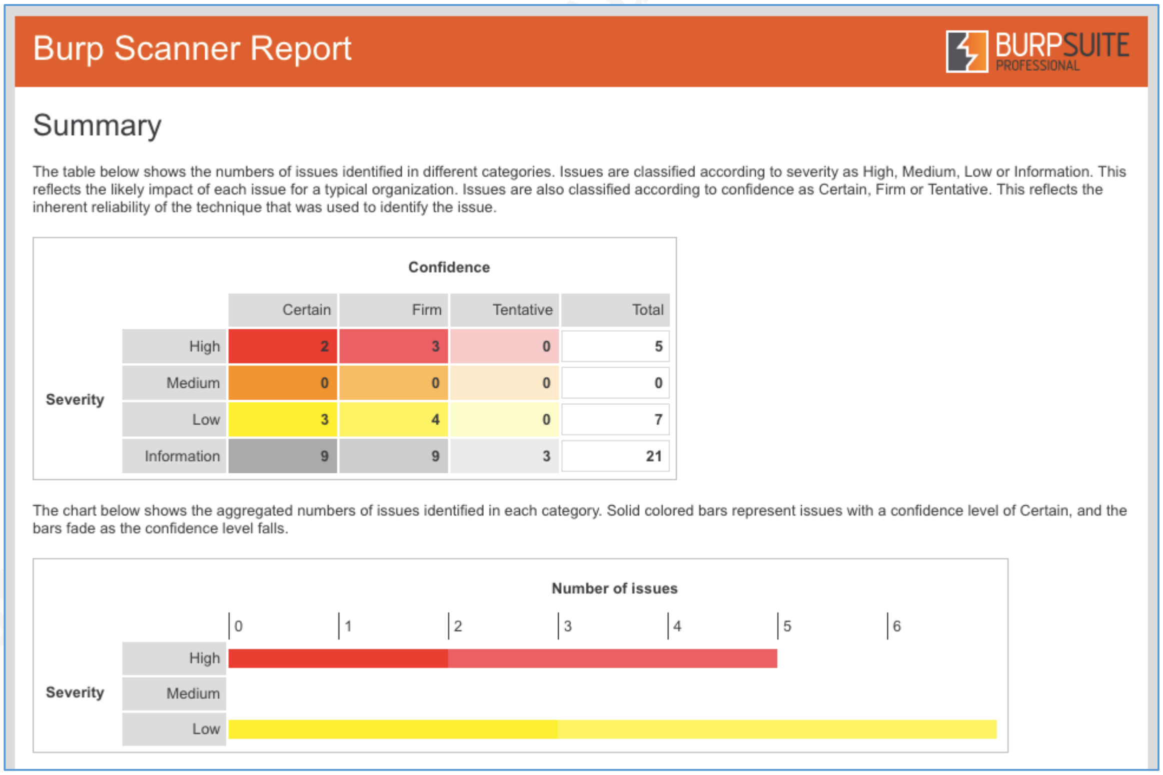
2017 Veracode, Inc.

© 2018 The SANS Institute Author retains full rights.

© 2018 The SANS Institute, Author Retains Full Rights

© 2018 The SANS Institute, Author Retains Full Rights

Appendix D
Summary of Burp Static Scanner Results



Burp Scanner Report

Summary

The table below shows the numbers of issues identified in different categories. Issues are classified according to severity as High, Medium, Low or Information. This reflects the likely impact of each issue for a typical organization. Issues are also classified according to confidence as Certain, Firm or Tentative. This reflects the inherent reliability of the technique that was used to identify the issue.

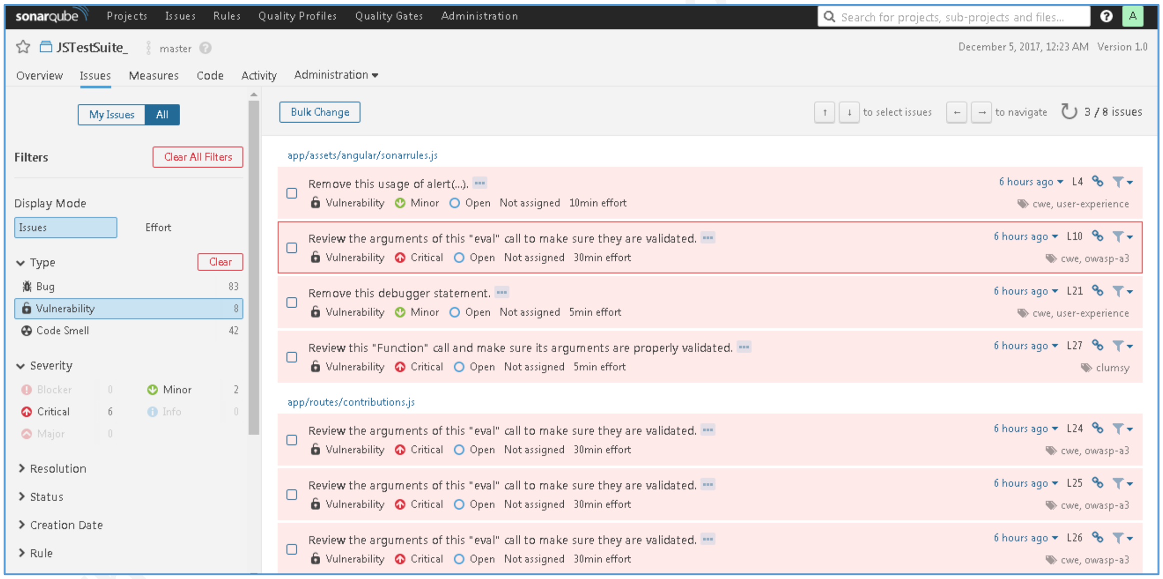
		Confidence			
		Certain	Firm	Tentative	Total
Severity	High	2	3	0	5
	Medium	0	0	0	0
	Low	3	4	0	7
	Information	9	9	3	21

The chart below shows the aggregated numbers of issues identified in each category. Solid colored bars represent issues with a confidence level of Certain, and the bars fade as the confidence level falls.

Michael Matthee,
michael.h.matthee@protonmail.com

© 2018 The SANS Institute, Author Retains Full Rights

Appendix E
Summary of SonarQube Results



JSTestSuite_ master

December 5, 2017, 12:23 AM Version 1.0

3 / 8 issues

Filters

Display Mode: Issues | Effort

Type

Bug	83
Vulnerability	8
Code Smell	42

Severity

Blocker	0	Minor	2
Critical	6	Info	0
Major	0		

> Resolution
> Status
> Creation Date
> Rule

app/assets/angular/sonarrules.js

Remove this usage of alert(...).
Vulnerability Minor Open Not assigned 10min effort
6 hours ago L4 cwe, user-experience

Review the arguments of this "eval" call to make sure they are validated.
Vulnerability Critical Open Not assigned 30min effort
6 hours ago L10 cwe, owasp-a3

Remove this debugger statement.
Vulnerability Minor Open Not assigned 5min effort
6 hours ago L21 cwe, user-experience

Review this "Function" call and make sure its arguments are properly validated.
Vulnerability Critical Open Not assigned 5min effort
6 hours ago L27 clumsy

app/routes/contributions.js

Review the arguments of this "eval" call to make sure they are validated.
Vulnerability Critical Open Not assigned 30min effort
6 hours ago L24 cwe, owasp-a3

Review the arguments of this "eval" call to make sure they are validated.
Vulnerability Critical Open Not assigned 30min effort
6 hours ago L25 cwe, owasp-a3

Review the arguments of this "eval" call to make sure they are validated.
Vulnerability Critical Open Not assigned 30min effort
6 hours ago L26 cwe, owasp-a3

© 2018 The SANS Institute Author retains full rights.