Analysing and Improving Security Code Reviews

Max Tijssen*, Erik Poll*, Theodoor Scholte† and Haiyun Xu†
*Dept. of Computer Science
Radboud University Nijmegen
Nijmegen, Netherlands
Email: maxtijssen@gmail.com
erikpoll@cs.ru.nl
†SIG : Software Improvement Group
Amsterdam, Netherlands
Email: {t.scholte,h.xu}@sig.eu

***Abstract*—Security code reviews are an important part of many secure code development processes [6]. This research aims to improve this process, by means of an empirical study. The data we looked at are reports generated by seventeen groups of students performing a security code review, on the same system, using the OWASP ASVS. The ASVS is a guideline produced by OWASP which serves as a tool to help perform structured security code reviews for web applications [18]. The document itself lists 166 different defensively worded security verification requirements. We analyzed three different aspects. Firstly we gave a overview of how consent was spread between different requirements. Next we showed for which requirements tools could be used to assist, we found this to be a small subset of requirements with most of them still needing manual inspection. Lastly we calculated correlation between expertise and inter-rater agreement but failed to find any correlations strong enough to be proven by our current number of test subjects.**

I. Introduction

Security has become an increasingly important aspect of any information system, with more being spent on it every year[1]. Security code reviews have shown to be an invaluable tool [6] to help reach the goal of creating and maintaining a secure system. Code reviews are the process of manually analyzing (part of) the source code of a system. To support this process automatic tools can be used. With security code reviews this is done mostly to find vulnerabilities such as SQL injection or XSS vulnerable functions.

Whether done during development to stop vulnerabilities from ever making it to release [12] or on the code of a live system to find vulnerabilities before malicious attackers do, it should be clear how these reviews can help make a system more secure.

Security code reviews are for now largely done by hand. Although there are many different static code analysis tools (also known as source code scanners/analysers, abbreviated to SAT) available that can scan for security vulnerabilities, these will often only check a very specific subset of the things you would like to have reviewed. Examples of these kinds of tools include CheckMarx, HP Foritfy or FindBugs This means they leave much which should be done manually [2]. Which parts can and should be assisted by the use of automated tools is something we will investigate further on in this paper.

The problem with these reviews mainly being done by hand is that the influence of the individual reviewer on the end result is very strong, since some of the things you check are quite subjective (when is something secure enough?). This means that the results will be largely irreproducible, and thus it will also become hard to compare two different reviews with each other.

Another common occurrence is these reviews being quite unstructured, with reviewers simply seeing what vulnerabilities or bad practices they can find, and reporting on these. This further increases the role in which the individual reviewer influences the end result. This ad-hoc way a working further increases how time consuming and error-prone the process is.

The ASVS was created as a way to bring more structure to this process. It contains a large (166) set of security verification requirements (simply called requirements in the rest of this paper). These requirements describe a certain attribute that OWASP considers a secure system should have. An example of this is requirement V2.2 [18]: (Chapter two, requirement two)

> *Verify all password fields do not echo the users password when it is entered.*

.The ASVS is divided into chapters of related requirements, for instance requirements related to session management or access control. Each requirement is also assigned a certain level from one to three, which dictates how in depth the review is, with one being most basic and three the most advanced.

In this paper our main goal is to verify the OWASP ASVS's effectiveness in the code review process. We explore the security code review results of seventeen groups reviewing the same software system using ASVS. We will first focus on whether this process produces sufficiently reproducible results, and which areas are the most controversial. We will then examine how expertise and automatic tools can assist in producing reproducible results.

II. Research questions

The central goal of this research can be split into three questions.

[1]http://www.gartner.com/newsroom/id/3135617

1) Is the OWASP ASVS a good way of producing reproducible results, and which parts produce the most controversy among reviewers?
2) Where can tools be used to improve the process when using the ASVS?
3) What role does expertise play when looking at how reproducible review results are?

With the answer to the first question you can make sure that the most controversial requirements are tackled by reviewers with more knowledge or experience. It might also assist writers of future versions of the ASVS or similar documents take a look at where the most controversy is currently found. The first part of the question might also help prospective user decide whether they want to use the ASVS as the basis for their reviews.

The answer to the second question can help reviewers have a good idea of where static analysis tools are beneficial and where there is potential for automation.

The last question can of course help with the setup of the code review process, by making sure that the reviewers you have available are assigned in the most efficient way.

III. Methods

A. Data collection

The data used for this research was collected by having seventeen groups of three to four Master students in Computer Science do a security code review of the same code using the OWASP ASVS (2014 version).

For the purpose of this research the groups were asked to check all level two requirements and lower, and disregard chapters 11 and 13 (since these were not relevant for the system). Groups were free to also check level three requirements, but the verdicts for these requirements were discarded for the purposes of this research. This left 107 requirements to evaluate.

The system under question is the same as the one used by Edumdson et al [1] in their research, which is a modified version of the Anchor CMS called TestCMS. The modifications basically come down to re-introducing security vulnerabilities which were fixed in the CMS but that were there originally, in order to make sure there were significant vulnerabilities to find. The reasons for using this system are basically the same as described in [1] (known vulnerabilities, small size and permissive license) and also simply because it was available to us.

The students were instructed to atleast try out some of the static code analysis tools (SATs) provided (RATS[2], RIPS[3] and two large commercial SATs) and otherwise use them as they saw fit.

The reviews produced a report consisting of a verdict per requirement of the ASVS, with an argumentation, and a reflection consisting of their experiences with the reviewing process itself and using the static code analysis tools. The possible verdicts were the following:

- **Pass**: The requirement is met by the system.
- **Fail**: The system fails the requirement.
- **NA**: The requirement is not relevant to the system.
- **Don't Know**: Due to either time constraints or lack of knowledge the students were unable to reach a verdict.

Some students added their own verdicts as specializations of these, but for the purpose of the research these are ignored.

Prior to the review each group was also asked to fill out a questionnaire where the individual members were asked to rate their expertise levels in web application development, PHP, web application security and penetration testing on a scale of one to five (one being the lowest and five the highest amount of expertise).

This way each group produced two documents to be processed:

- A *review* consisting of a list of requirements with the verdict the reviewers found, along with an explanation of how they came to this verdict and a reflection on both the usage of tools and the process of using the ASVS.
- A *questionnaire* where each group member indicated their expertise level in four areas. This questionnaire was simply four closed questions with a scale from 0 to 5 indicating knowledge.

B. Processing

As described in section II. there are three questions we wish to answer with this data, each will have it's own processing.

a) Consensus:

In order to measure reproducibillity of certain requirements we will use consensus. If there is a lot of consensus about what a certain requirement should be, then it logically follows that a different reviewer reviewing this same requirement would come to the same verdict. Here we will use a consensus measure to measure how much different groups agree with each other with regard to the validation of requirements. The consensus factor is the following:

$$\frac{max(Pass, Fail, NA)}{sum(Pass, Fail, NA)} \tag{1}$$

where Pass, Fail and NA are the number of occurrences of Pass, Fail or NA as a verdict respectively. The function *max()* takes the value of the maximum number of the same verdict given, while *sum()* sums up all verdicts. Note that sum is not necessarily seventeen, since groups could also indicate "don't know", which are thus not taken into account in this measure.

For example, if seven groups gave a Pass, four a fail, and six a NA then this measure will give the following.

$$\frac{max(7, 4, 6)}{sum(7, 4, 6)} = \frac{7}{17} = 0.41 \tag{2}$$

This measures how large a segment of groups chose the majority verdict. The measure ranges from 0.33 (perfect distribution between the three verdicts) to 1 (complete consensus). We feel this is a useful measure since it gives a good overview

[2]https://code.google.com/p/rough-auditing-tool-for-security/

[3]http://rips-scanner.sourceforge.net

of the maximum amount of groups that agreed with one another.

b) Tool Usage:
In order to find out which parts of the review might be improved by using static analysis tools we will tally up all uses of the tools per requirement, giving a good overview of where tools can be used. We will also calculate the consensus that the tool documenting groups had on these different requirements, to see if this differs from the other requirements.

c) Expertise:

For the third question we will first need to compute a perceived expertise level for each of the groups. For this we will use two different measures, the average expertise of the group members and the maximum value that any group member reported. Depending on these two measures we will try to find out if there is a significant correlation regarding how many other groups agree with their verdict.

Next we will for each group calculate how many of the other groups agreed with their verdict for each requirement (the inter-rater agreement), and calculate an average over all ASVS requirements. This will yield a rough estimate of how reproducible their results are, since it shows how much of the other groups on average agreed with them.

For example, if for a requirement the groups gave the following verdicts: seven Pass, four Fail, six NA. A group that gave Pass, will have an inter-rater agreement of seven for this requirement, while one that give Fail will have a score of four.

These different knowledge factors are then correlated with the inter-rater agreement to find out whether these are in fact correlated with each other.

IV. Results & Analysis

A. Consensus

Table I shows the average consensus factor as described in section IV.B for both chapters and the ASVS as a whole. Table II show how the individual requirement's consensus is distributed, note that due to rounding these do not add up to 100. Figure 1 shows a boxplot of how the consent of different requirements is distributed.

As can be seen in table I the overall consensus of the ASVS is 0.69, meaning that about 69% of groups agreed with each other on a arbitrary requirement. While this is not particularly low it should be kept in mind that only three different verdicts were counted to come to this consensus factor, which naturally increases consensus. Figure 1 and Table II also show that many of the requirements score far below this average.

From this figure we feel that there is plenty of room for improvement in both the ASVS and the security review process in general.

The per chapter consensus factor as seen in Table I varies between 0.52 and 0.81. Respectively these two chapters are,V16: Files and Resources Verification Requirements and V3: Session Management Verification Requirements.

TABLE I
Agreement per chapter

Chapter	Mean	Standard Deviation
Overall	0.69	0.19
2	0.78	0.21
3	0.81	0.14
4	0.69	0.19
5	0.62	0.14
7	0.63	0.23
8	0.71	0.16
9	0.66	0.19
10	0.73	0.13
11	0.65	0.19
15	0.70	0.15
16	0.52	0.08

As can be seen in the table most other chapters either show a mean quite close to the overall mean, or have a standard deviation such that there is little point speaking on a per chapter basis. There is little point since the requirements within chapters differ so widely that it makes a lot more sense to look at individual requirements. This is further compounded by Figure 1 which shows that the chapters (apart from three and sixteen) have similar wide distributions.

The trend we do notice on a per requirement basis is that scope is very often a problem when it comes to consensus. When it is hard to judge whether a certain requirement is (partially) in scope of the system then groups tend to have different approaches. Some will indicate a hard NA, otherwise will make it a implicit pass (if they don't use it they don't need to protect it) while others might think it was in scope and simply give a Fail. See V2.16,V16.6 or V4.4 for examples.

Wording that can be interpreted multiple ways can be another problem. An example of this is requirement V16.9 [18]:

> *Verify the application code does not execute uploaded data obtained from untrusted sources.*

which also has a very low consensus of 0.41. You need a very specific definition of trusted sources, which will not always be the case. Furthermore there is no solid definition of executing code. When talking about about web applications it can be argued that any HTML outputted is executed code (on the client side) while it might also be interpreted only as explicitly executed PHP code. Which of these interpretations is technically correct is irrelevant, since this very discussion decreases consensus.

There are of course many different interpretations why certain chapters or requirements produce more disagreement then others, none of which this current data can fully (dis)prove.

B. Tool Usage

The usage of tools by the different groups per requirement can be found in Table III. It is important here that there has been made no distinction between which tools were used, even if they were not one of the four presented, as long as they were static analysis tools.

TABLE II
DISTRIBUTION OF REQUIREMENTS PER CONSENSUS LEVEL

Consensus	% of requirements
1	8%
0.8 up to 1	25%
0.6 up to 0.8	26%
0.4 up to 0.6	37%
0.4 and lower	2%

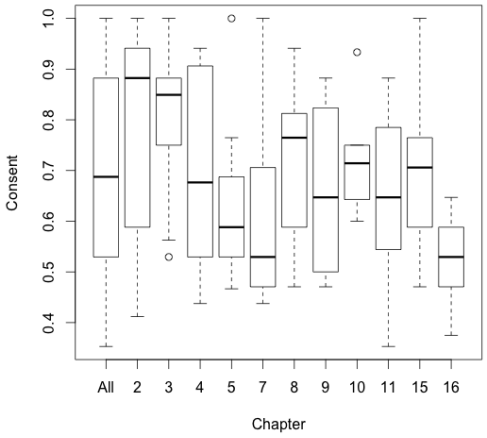

Fig. 1. Boxplot of consent in individual chapters.

Another important note to make here is that only half of the groups actually documented for which specific requirement they used tools, so the numbers in the table reflect a population of eight groups, instead of the seventeen in the rest of the research.

Also included in the table is the average consensus of the eight tool using groups on these requirements.

A first important observation from the table is just where tools can be used. For instance, for requirement V5.16 six out of eight tool registering groups indicated tool usage. This would indicate that requirement V5.16, and similar requirements, would be prime candidates for tool usage. The requirement reads [18]:

> *Verify that all untrusted data that are output to HTML (including HTML elements, HTML attributes, JavaScript data values, CSS blocks, and URI attributes) are properly escaped for the applicable context.*

This basically means to check whether XSS attacks are possible. Although this requirement sounds a lot like requirement V16.9 ,discussed in section VI.B., this requirement asks whether a specific protection is in place. This leads to a much more objective requirement, which in turn creates a higher consensus. V5.10, which has five indicated uses, similarly asks to reviewer to check whether SQL injection is possible.

The other requirements follow a similar trend as these two, asking to check whether certain technical flaws are present in the code, or if the needed protection is in place. From our results it can be concluded that reviewers found tools to be helpful in these cases.

TABLE III
NUMBER OF GROUPS INDICATING TOOL USAGE PER REQUIREMENT

# Groups	Requirements	Avg consensus
6	5.16	1
5	5.10	0.875
3	3.14	1
2	2.13, 5.12, 7.2, 7.6, 7.7, 9.1	0.75
1	2.2, 3.3, 4.8, 5.3, 5.11, 5.18, 5.19, 10.6, 11.8, 11.9, 11.10, 15.5, 16.9	0.626

As you can see from Table III, only quite a small subset of requirements has actually been tackled using tools (see Table V and VI for all requirements). Only for 22 out of 107 (21%), requirements tools were used, when looking at only 3+ groups per requirement this fall even further, 3 out of 107 (3%). Meaning that most of the requirements must still be checked by hand. This is also found in the reflection parts of the reviews. Here nine out of seventeen groups indicated that the tools were a good complementary resource to give quick insight, but not to judge individual requirements. This is further compacted by five groups explicitly indicating that they regret the tools can only be used for few verdicts.

Another interesting observation is about how tool usage impacts the consensus of requirements. The consensus of requirements where three or more groups indicated tool usage have much higher consensus then average (0.70 for tool documenting groups), with only one group deviating from the verdicts the others gave in V5.10. This seems to indicate that using tools, wherever possible, (as you would expect) would be a good way to increase reproducibility.

C. Expertise

Table IV contains the correlation of the different knowledge areas with the inter-rater agreement. The table also contains the p-values, obtained by using a Chi-squared Test of Independence, with null-hypothesis that the knowledge area and inter-rater agreement are not correlated.

The correlations shows that for a few areas there is a correlation as high as 0.48. These areas are Buidling Web Application Max and PHP Max. This would indicate that, apart from Penetration Testing knowledge which generated quite low results, the knowledge areas tested positively contribute to creating more reproducible results.

Furthermore we can see that in three out of four knowledge areas the maximum amount of knowledge in a group contributes more then the average. This would mean that it would be beneficial to pair spread out your highest knowledge reviewers as much as possible, in order to maximize this value among different groups.

A glaring problem here is the p-value generated by doing a Chi-squared Test of Independence. The p-value has a range from about 0.2 to 0.4, significantly higher then the standard measure of 0.05, meaning that we cannot make statistically sound conclusions from this data. This can be explained by

TABLE IV
CORRELATION OF KNOWLEDGE WITH INTER-RATER AGREEMENT

Knowledge area	Correlation	p-value
Building Web Application Average	0.31	0.21
Building Web Application Max	0.48	0.22
PHP Average	0.40	0.28
PHP Max	0.48	0.32
Web Application Security Average	0.29	0.35
Web Application Security Max	0.10	0.40
Pen Testing Average	0.00	0.26
Pen Testing Max	0.16	0.38

the relatively low test population. With only seventeen groups the chances to create an equally extreme data set are of course quite high.

Despite this we do feel this does show some interesting facts, even if only to stimulate further research with a larger group of test subjects.

V. RELATED WORK

Firstly the most relevant paper for this research is a paper by Edmundson, et al [1]. It could be said that their research was the starting point of our own. In this paper they hired thirty developers to do a code review of a small web system they supplied, the same system used in this research. They also asked the reviewers to rate a few different factors which they hypothesized might predict their effectiveness. From these reviews and effectiveness factors they drew a few conclusions. Firstly that no reviewer found all vulnerabilities in the system. Next that predicted effectiveness does not seem to increase the amount of vulnerabilities found. Lastly that there was a significant correlation between the number of correct vulnerabilities found and the number of incorrect ones reported. Our research differs in a few key areas. Firstly the reviewers from the 2013 research were tasked to find concrete vulnerabilities, while our students will pass verdicts on a set of more abstract set of requirements (the ASVS). Next, where the research by Edmundson et al focused on the amount of vulnerabilities found (correctness) ours focuses on reproducibillity. In this way (see the research questions) our goals differ quite a bit from this earlier research.

Code reviews, both security focused and general, have seen quite some research the last few years, because of their earlier described importance. This research has taken many different directions. For some examples see [5], [8], [11], [14], [16], [17].

The usage of tools within this process in particular has also received a fair share of attention. See [2], [3], [10], [13], [15].

VI. DISCUSSION

We feel the results of research answer most of the research questions. The major caveat is the question regarding expertise, unfortunately our sample size is to low generate any meaningful conclusions about statistical correlation. This could be due to either to a correlation that is to weak to be proved by our test size or simply a lack of correlation. Refer to the section Results & Analysis to see how our results fulfill the other research questions.

We did also get some interesting insights not directly pertaining to the research questions. One of them is the way the ASVS is worded versus the way that the static analysis tools work. The ASVS is very much worded in a defensive way (see V5.16 for a good example), where they note which defenses should be in place, while the tools are very aggressively oriented, where they actively try to find vulnerabilities.

To our knowledge this research is the first to take a look at the reproducibility of the OWASP ASVS, so these findings can not be compared to other research.

Our findings regarding tool usage however can be compared quite closely. The presentation by OWASP regarding manual versus automatic code review [2], agrees that for using the ASVS you should use a combination of manual inspection with tools wherever possible. We also see similar results as them regarding what requirements can be checked using tools. For instance V5.16, one of the requirements they give as an example, where six of our eight tool documenting groups indicated they did indeed use tools. Our findings do not completely agree with their view of where tools can not be used however. V2.13, about password hashing and salting, for instance they give as an example of a requirement that cannot be checked using a tool, while two of our groups did indicate that they did this. A possible explanation for this is that this presentation stems from 2009, and tools have developed further in this time.

Our research results regarding the effect of knowledge and experience can be compared to [1]. In this work they found there to be no correlation between Web development, while in our research we did find one of 0.48. In their research they use a high p-value (0.88 compared to our 0.40) to dismiss this correlation, while we feel that with a larger set of test subjects there is a correlation to be found and proven. An important distinction though is that their research correlates this experience with correctness, while ours focuses on reproducibility.

Another interesting comparison can be made by a research from 2007 by Hatton [17]. Here it was found that checklists during code reviews offered no significant advantage over a more ad-hoc approach when it came to finding faults within code. The ASVS in our research fills a similar purpose as this checklist, only on a more high level, so requirements and not individual faults. It would of course not be possible to compare the results of groups using the ASVS with groups simply looking for vulnerabilities but it would be interesting to do some research regarding just how much value such a structured approach brings.

This research has in our opinion helped in increasing the insight in where the major points of contention lie when conducting security code reviews, by showing where consensus is lowest. This knowledge can be applied to both the ASVS specifically as well as code reviews in general. Furthermore we have increased the knowledge about possibilities of automation and tool usage when conducting security code reviews. Lastly

it has provided further evidence that it is hard to produce reproducible results when conducting security reviews, even when using a structured approach such as the ASVS. This shows that verdicts regarding security requirements do heavily depend on the reviewer.

VII. Recommendations

A. Practitioners

Based on this research we give the following recommendations to security code reviewers and their managers. Please note that this research was done using the ASVS 2014 version, if you wish to apply it to other version that refer to OWASP's change log to see how certain requirements have changed.

Firstly, we highly recommend to usage of static analysis tools, wherever they can be used. We have shown that when judging code using these tools, verdicts given show much higher consensus. It is on the other hand very important to remember that these tools should serve as a complementary resources, and that the majority of the work will have to be done by hand, until the tools undergo some major changes. This way of working was also indicated by our test subjects as most beneficial, and other research backs this up [2].

Next we have some recommendations regarding the distribution of knowledge and experience between review groups. Making sure the maximum knowledge within each group is as high as possible has been shown to produce the most inter-group agreement, and thus reproducibility.

Furthermore we heartily recommend looking at Tables V and VI to see which requirements have the lowest consensus, and creating strict guidelines within your group of reviewers about when these should pass or fail. Furthermore you might wish to assign you most knowledgeable people to these requirements, in a hope of increasing consensus.

B. Researchers

We recommend future researchers we are interested in doing empirical research regarding (security) code reviews the following. A document such as the ASVS is a great way to get data subjects which might not have done code reviews before to produce usable reviews. Of course some technical knowledge of the subjects is necessary, but experience with doing reviews is not a requirement.

We found our used consensus measure to be easy to understand, use and analyze, and would recommend it for similar research.

Furthermore we recommend any people looking to improve the ASVS or similar documents to take a look at the consensus results we found, to see where there is still a lot of controversy regarding requirements.

VIII. Limitations of research

This research faces a limitations which we will discuss in this section. We will discuss internal validity limitations, which are threats to the correctness of conclusions drawn from this research, as well as external validity, which are threats in how far this research can be generalised to other examples.

A. Internal Validity

One possible shortcoming is the selection of subjects. Since there has been no prior screening to the students (other then that they were for computer science master students) they might have very different previous experiences. Picking a group of subjects with more closely spread experience levels might have produced more consensus within the requirements. A mitigating factor here is that experience in real world reviewers will vary at least as much.

The program we had our groups review can also be seen as a threat to internal validity. The program was quite small, due to time constraints of the reviewers, and large scale projects might generate quite different results. The programming language this program was in (PHP) will also have influenced the reviews, with more modern languages possibly being harder or easier to verify. Similar research on other languages might shed some interesting light on this.

The program is also a threat in another way, due to certain vulnerabilities (not) being present. This means that tools were able to track these vulnerabilities and report on them, thus increasing tool usage numbers for the relevant requirement. If this vulnerability was not present the tool usage would be much lower, since you cannot use tools to prove the absence of something.

Lastly the limited population size presents a very real threat to internal validity. As can especially be seen in table II our population size makes for quite high p-values. For this reason this research might benefit a lot from being repeated with more data subjects.

B. External Validity

Our usage of students for this research presents a few more threats to external validity.

Firstly most obvious one is selection bias. Since our population is entirely made up of students there is no way to prove that a more general group of reviewers will behave similarly.

Another threat is in how far participation to this research was voluntary and accordingly how much effort was put in. As a part of the course it was mandatory that the students do the review. This might have led students to simply put in the minimum amount of effort to generate the reports, while paid reviewers might be motivated to put in more time.

Our knowledge measurement presents another problem. Both by being only perceived expertise and focusing on four quite specific areas this measure might not be usable for a wider area.

IX. Conclusion and Future Work

Our research has produced some interesting and useful results, both for future researchers in the area of (security) code reviews and for practitioners using the ASVS or similar resources.

Firstly we have shown how consent is distributed over the ASVS, with some requirements having perfect consensus, and others nearing total disagreement. We have given some

possible reasons requirements produce a certain consensus, but there is still a lot more work to do in this area.

Next we have shown where SATs can be used when using the ASVS. Unfortunately the SATs could only be used on quite a specific subset of requirements, although our groups did indicate they were very helpful for creating a good overview of the system. We did also see that the requirements where tools could be used had very high consent. This might seem obvious but because of the extra layer of interpretation that happens (since the ASVS is defensively worded and the tools aggressively) it is still a good idea to check this.

Lastly when looking at expertise we were unfortunately not able to find sufficiently strong correlation factors for the size of our data set.

These results combined can help practitioners allocate their resources in a efficient way and possible improve the ASVS or the usage of it. Furthermore we have shown that there is still more work to be done in this area, in order to improve security code reviews.

Currently another batch of students are following the same course that produced the reviews used for this research. These students will similarly to the subjects of this research produce reviews of the same program.

This means that for further research of size of our data set will be effectively doubled. A change with the current approach is that students are explicitly asked to fill in a data sheet with their verdicts, argumentation and whatever tool they used to get this verdict. This way our amount of tool documenting groups will effectively triple (since currently only half of the groups did this).

Besides simply running the same research with a larger set of test subjects we are also planning to do some more in depth research about why certain requirements produce less consent then other. Here things like wording, length of requirements and how specific the requirements are could be analyzed, along with their consent.

Our consensus measure colour the results in a significant way. Our consensus measure will only reach under 0.5 when groups doubt whether the requirement is within scope or not (since there can otherwise only be a spread between Pass and Fail). This means that requirements where scoping is not entirely clear have a tendency to have much lower consensus. Future research that wishes to place less emphasis on scoping issues could decide to use a different measure.

Acknowledgment

The authors would firstly like to thank the students who produced the reviews for our research. Next we would like to thank Georgios Gousios for his valuable input on empirical research. We would also like to thank the Software Improvement Group (SIG) for the use of their material, space and time for the writing of this paper and Berkeley University for the use of their testCMS system as a case study.

References

[1] Edmundson, A., Holtkamp, B., Rivera, E., Finifter, M., Mettler, A., & Wagner, D. (2013). An empirical study on the effectiveness of security code review. In *Engineering Secure Software and Systems* (pp. 197-212). Springer.

[2] Kesäniemi, A., & Oy, N. (2009). *Automatic vs. Manual Code Analysis*[Pdf document, presentation slides for presentation held 2009-11-17]. Retrieved from https://austin.owasp.org/images/5/53/Ari_kesaniemi_nixu_manual-vs-automatic-analysis.pdf

[3] Baca, D., Petersen, K., Carlsson, B., & Lundberg, L. (2009). Static Code Analysis to Detect Software Security Vulnerabilities-Does Experience Matter?. In *Availability, Reliability and Security, 2009. ARES'09. International Conference on* (pp. 804-810). IEEE.

[4] Chess, B., & McGraw, G. (2004). Static analysis for security. IEEE Security & Privacy, (6), 76-79.

[5] Da Cunha, A. D., & Greathead, D. (2007). Does personality matter?: an analysis of code-review ability. *Communications of the ACM,* 50(5), 109-112.

[6] Lipner, S. (2004). The trustworthy computing security development lifecycle. In *Computer Security Applications Conference, 2004. 20th Annual* (pp. 2-13). IEEE.

[7] Van Wyk, K. R., & McGraw, G. (2005). Bridging the gap between software development and information security. *Security & Privacy, IEEE,* 3(5), 75-79.

[8] Zheng, J., Williams, L., Nagappan, N., Snipes, W., Hudepohl, J. P., & Vouk, M. (2006). On the value of static analysis for fault detection in software. *Software Engineering, IEEE Transactions on,* 32(4), 240-253.

[9] Rigby, P. C., & German, D. M. (2006). A preliminary examination of code review processes in open source projects. University of Victoria, Canada, Tech. Rep. DCS-305-IR.

[10] Finifter, M., & Wagner, D. (2011). Exploring the relationship between Web application development tools and security. In *USENIX conference on Web application development.*

[11] Scandariato, R., Walden, J., & Joosen, W. (2013, November). Static analysis versus penetration testing: A controlled experiment. In *Software Reliability Engineering (ISSRE), 2013 IEEE 24th International Symposium on (pp. 451-460).* IEEE.

[12] McGraw, G. (2004). *Software security. Security & Privacy, IEEE,* 2(2), 80-83.

[13] Remillard, J. (2005). Source code review systems. Software, IEEE, 22(1), 74-77.

[14] Wang, Y., Yijun, L. I., Collins, M., & Liu, P. (2008). Process improvement of peer code review and behavior analysis of its participants. In ACM SIGCSE Bulletin (Vol. 40, No. 1, pp. 107-111). ACM. Chicago

[15] Belli, F., & Crian, R. (1996). Towards automation of checklist-based code-reviews. In Software Reliability Engineering, 1996. Proceedings., Seventh International Symposium on (pp. 24-33). IEEE.

[16] Fagan, M. (2002). Design and code inspections to reduce errors in program development. In Software pioneers (pp. 575-607). Springer Berlin Heidelberg.

[17] Hatton, L. (2008). Testing the value of checklists in code inspections. Software, IEEE, 25(4), 82-88.

[18] Open Web Application Security Project (OWASP) (2014) *Application Security Verification Standard Project (ASVS)*[Pdf document]. Retrieved from: https://www.owasp.org/index.php/Category:OWASP_Application_Security_Verification_Standard_Project

Appendix A
Consensus per Requirement

TABLE V
CONSENSUS PER REQUIREMENT (V2.1 - V5.18)

Requirement	Agreement
V2.1	0.53
V2.2	0.59
V2.4	0.94
V2.6	0.65
V2.7	1
V2.8	0.69
V2.9	0.53
V2.12	1
V2.13	1
V2.16	0.41
V2.17	0.88
V2.18	0.59
V2.19	0.94
V2.20	0.88
V2.21	0.41
V2.22	0.88
V2.23	1
V2.24	0.94
V2.25	0.88
V3.1	0.88
V3.2	0.65
V3.3	0.81
V3.4	0.88
V3.5	1
V3.6	0.82
V3.7	0.94
V3.8	0.88
v3.10	0.53
V3.11	0.75
V3.12	0.56
V3.14	0.88
V3.15	0.82
V4.1	0.75
V4.2	0.65
V4.3	0.53
V4.4	0.53
V4.5	0.88
V4.8	0.47
V4.9	0.44
V4.10	0.53
V4.11	0.71
V4.14	0.94
V4.16	0.94
V4.17	0.94
V5.1	0.56
V5.3	0.65
V5.4	0.59
V5.5	0.69
V5.8	0.53
V5.10	0.76
V5.11	0.6
V5.12	0.71
V5.13	0.47
V5.14	0.5
V5.16	1
V5.17	0.53
V5.18	0.53

TABLE VI
CONSENSUS PER REQUIREMENT (V7.1 - V16.10)

Requirement	Agreement
V7.1	1
V7.2	0.47
V7.3	0.44
v7.6	0.71
V7.9	0.53
V8.1	0.75
V8.2	0.81
V8.3	0.82
V8.4	0.47
V8.5	0.59
V8.6	0.76
V8.8	0.76
V8.10	0.47
V8.11	0.94
V9.1	0.76
V9.3	0.88
V9.4	0.53
V9.5	0.47
V10.1	0.71
V10.3	0.6
V10.4	0.64
V10.6	0.75
V10.7	0.93
V11.2	0.69
V11.3	0.59
V11.6	0.65
V11.8	0.88
V11.9	0.35
V11.10	0.88
V11.12	0.5
V15.1	0.81
V15.2	0.47
V15.3	0.76
V15.4	0.71
V15.5	0.71
V15.6	1
V15.7	0.71
v15.8	0.59
V15.9	0.53
V15.10	0.71
V16.1	0.59
v16.2	0.47
V16.3	0.65
V16.4	0.47
V16.5	0.53
V16.6	0.38
V16.7	0.53
V16.8	0.56
V16.9	0.41
V16.10	0.59