International Journal of Engineering in Computer Science

E-ISSN: 2663-3590
P-ISSN: 2663-3582
IJECS 2023; 5(2): 06-12
Received: 16-05-2023
Accepted: 22-06-2023

Tyler W Thomas
Department of Mathematics Statistics and Computer Science, University of Wisconsin-Stout, Menomonie, United States

Corresponding Author:
Tyler W Thomas
Department of Mathematics Statistics and Computer Science, University of Wisconsin-Stout, Menomonie, United States

The intersection of static analysis and security code reviews: A collaborative model

Tyler W Thomas

DOI: https://doi.org/10.33545/26633582.2023.v5.i2a.93

Abstract
The development of code review methods highlights a growing demand for more robust systems to detect security vulnerabilities. Despite their benefits, conventional code review techniques including "Over the shoulder," "Pair programming," and "Email pass around," have shown persistent effectiveness gaps. Better synchrony between stated review goals and outcomes can be achieved with advancements in code comprehension among reviewers and facilitating automation in review tasks.
In this paper, I present a design and prototype of an experimental tool that combines static analysis with security code reviews to boost efficiency. Initiated by static analysis, developers make subsequent corrections that are later melded into the security review process. Developers, in liaison with security experts, aim to remedy any potential issues before the code is added to the codebase.
Three pivotal roles are recognized in this tool design - the primary developer, additional developers, and a security expert, which underscores the need for efficient collaboration. The tool is equipped with features like immediate messaging, conversation recording, synchronization of warnings and annotations, and a system to sort issues accordingly. In alliance with the open-source lightweight code review tool, Gerrit, this tool design could enhance code review productivity and stimulate developers' acceptance of security code reviews. Future research will be crucial in gauging the impact and efficacy of such tools in practical implementations.

Keywords: Code review, security, cyber security, code review, static analysis, interactive static analysis

Introduction
As previous work demonstrates, code review is very effective at detecting bugs in code and can also be used to detect security vulnerabilities in code [5, 2]. However, it has been shown to be ineffective on its own for detecting security vulnerabilities [7, 6, 4]. Researchers have called for security code review as a separate process, and in my previous security auditor study [11], participants reported that this does occur [4]. Various attempts have been made to combine static analysis and standard code review for the purposes of finding defects in code [9, 3, 8]. Most have been successful at this task, but have not focused on security issues [9, 3, 8]. Previous research has shown that interactive static analysis can help mitigate the problems of static analysis and make it something developers are more willing to perform [15, 14, 13] and has also shown that it can help train the developer [15, 14, 13]. However, in previous work on interactive annotation, I have shown that developers need more assurance that their solution is correct and do not always know how to resolve vulnerabilities [10]. Code review involving a security expert in a security code review could provide that assurance and catch incorrect solutions. Therefore, combining interactive static analysis with security code review may prove effective.
In this paper, I first discuss key design considerations about the security code review process. Moreover, I provide details of a tool design from the design considerations. I propose a new lightweight, tool assisted, security code review process. I finally discuss the security code review tool I built, an implementation of this tool design.
My previous work has shown an aggregate security auditor workflow model based on the workflows described by my participants [11]. If security code review fed by interactive static analysis is to be useful, it must be inserted into existing processes somehow.
It seems most intuitive to place this process after functional testing, independent of audits.
In this new workflow model, developers would conduct interactive static analysis or standard static analysis, apply remediations, and feed the output into the security code review process.

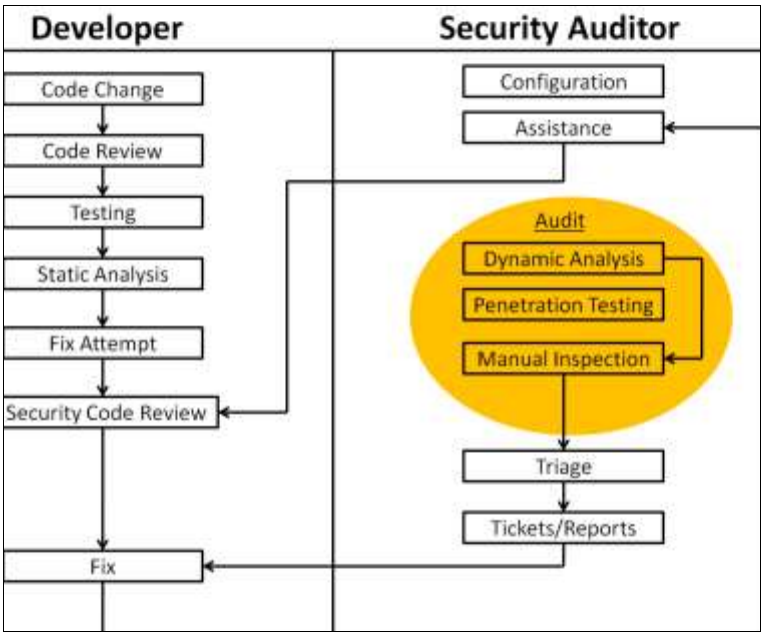


Fig 1: Security auditor workflow model with security code review

They would then communicate with security experts during security code reviews. Once this is done, they would attempt to fix issues based on received guidance. Audits would remain the same as before, with the removal of static analysis. Ideally audits would be conducted by security experts as an independent process. However, for organizations with limited budgets or application security experts, audits could skip manual code inspection since the code would have already been inspected in stages. For organizations with extremely limited budgets, audits could be removed entirely. The most ideal process, with audits included, is illustrated in Figure 1.

Design Considerations

In this section, I discuss three categories of design considerations necessary to my security code review tool design. These design considerations include roles, communication and collaboration, and warnings. Moreover, I discuss why each one is important.

Design Roles

In performing security code review, three roles quickly become apparent. The first of these roles is that of the primary developer. The primary developer is considered to be the developer who wrote the code. This person requests a security code review and alters the code directly to fix the vulnerabilities. The second role is that of the reviewer or alternate developer. Alternate developers work on similar projects or code as the main developer and serve as peers. They raise issues or comment on issues raised by the IDE, as well as questions from the primary developer. Lastly, I propose that security code review involve at least one security expert to ensure the security effectiveness of the code or suggestions raised by the alternative developers. The security expert fills a similar role to the alternative developers, but is in a unique position to comment on the security effectiveness of vulnerability resolutions. Alternate developers, on the other hand, are likely to possess more knowledge of software engineering best practices and the functionality of the underlying project code.

Communication and Collaboration

Security code review should involve much collaboration between the primary developer, alternate developers, and security experts. Primary developers have direct knowledge of the code being reviewed, as they have either written all of it or a large portion of it. However, they may have very little security knowledge to accurately address security warnings from static analysis and spot other security issues in code that the algorithms may miss. Security experts, on the other hand, know much about the security vulnerabilities, but are very unlikely to have much understanding of the current project code. Alternate developers are likely to have expertise somewhere between the two. Alternate developers are likely to have much knowledge of software engineering best practices and some knowledge of the project. They may also know alternative software engineering solutions to problems that the primary developer does not know. However, it is extremely unlikely that they will have the security knowledge of a true security expert. They may each provide different perspectives and different kinds of reviews and responses in this type of code review.

Warnings

When the primary developer interacts with a static analysis tool such as ASIDE prior to conducting a code review, they will be shown many vulnerability warnings. These include warnings for SQL injection, cross site scripting, CSRF, and access control vulnerabilities. Contextual help based on the actual underlying code will be provided when the warnings are selected. The primary developer will interact with these warnings and attempt to resolve them. The primary developer will also perform interactive annotation for annotation requests. When the code review process occurs, the alternate developers and the security expert should both be able to observe the warnings and comment on the warnings.

Research Questions

Roles

- What are the activities and contributions of people in each of the three roles in security code review?

- How can a tool support these roles and activities?
- What are the perceptions of people in different roles of performing security code review?
- What type of training is required for developers to effectively perform security code review?

Communication and Collaboration

- How do participants in each role communicate and collaborate with each other regarding code review information?
- Which features in a tool are most effective for collaboration between people in three roles and security experts?
- How should feedback from other participants be displayed to users in each role?

Warnings

- How to effectively communicate static analysis warnings, mitigations, etc to code reviewers?
- How do code reviewers interpret and respond to different kinds of security warning and vulnerability information?
- How do code reviewers resolve security vulnerabilities through code review?
- How effective are they?
- What is the role of security experts in security code review?

Sec View Prototype

With these design considerations in mind, I have created a tool prototype to study the security code review process. In this section, I describe the features of this design. I also discuss an example implementation of these features. Additionally, I discuss collaboration through the tool, and I provide a summary of the tool's research contributions.

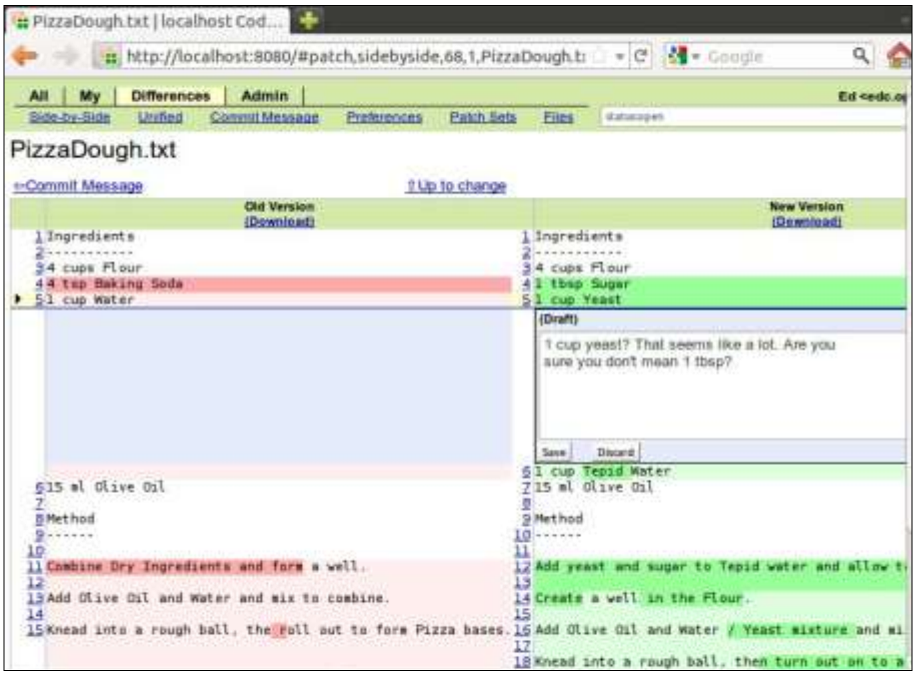
Prototype Features

The Sec View prototype extends Gerritt and adds an embedded web server, embedded application server, Javascript files, and a database. As a reminder, Gerritt is an open source light code review tool with a web based interface [1]. It can intercept commits in route to a git repository and postpone the commit until a code review has been completed. These features allow the tool to serve as a dedicated security code review tool. The tool will leverage all of the existing features of Gerritt which it uses for normal code review. However, the annotations, annotation requests, and warnings are shown to other developers and security experts as a part of a security code review. Security experts are then able to login through a web interface and collaborate on the issues. The interface for the security experts is designed slightly differently than the interface for other developers. The tool enables collaboration between the primary developer, other developers, and the security experts. The details of this design are based on the results from my previous security auditor study [11]. The tool design includes the following features:

- Instant messaging between developers in the IDE and security experts using the web interface, handled per warning.
- Log of instant messaging conversations for each warning.
- Synchronization of annotations and warning information, made viewable to the security experts and developers.
- Ability to mark issues as correctly resolved, requiring modification, or unresolved
- Ability to push final code to repository.

Prototype Roles

When the tool is deployed, users are able to fill any of the three roles. The primary developer interacts with the tool through the IDE and initiates a code review. The security experts interact with the tool by logging in through a web interface, synced in real time. However, they do not initiate the code review and will instead receive notices of ongoing code reviews. The IDE of the primary developer is synchronized during code reviews. Additionally, other developers assigned as reviewers are able to log in and contribute to the code review through a web interface.

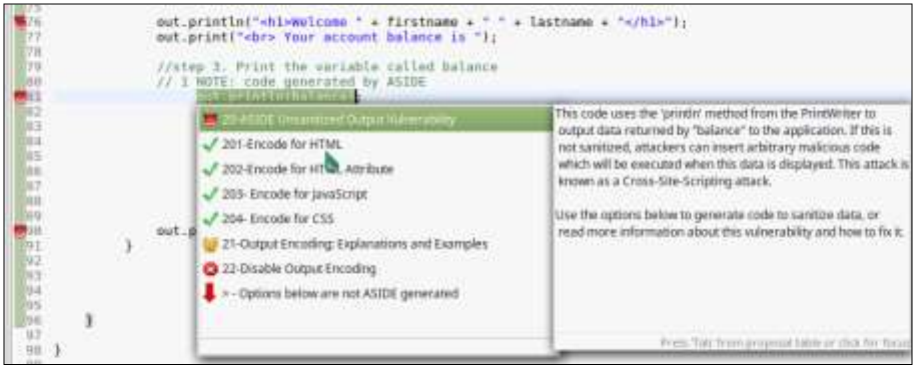


```
Ingredients
-----------
4 cups Flour
4 tsp Baking Soda
1 cup Water
15 ml Olive Oil

Method
------

Combine Dry Ingredients and form a well.

Add Olive Oil and Water and mix to combine.

Knead into a rough ball, the Roll out to form Pizza bases.
```

```
Ingredients
-----------
4 cups Flour
1 tbsp Sugar
1 cup Yeast
1 cup Tepid Water
15 ml Olive Oil

Method
------

Add yeast and sugar to Tepid water and allow t

Create a well in the Flour.

Add Olive Oil and Water / Yeast mixture and mi

Knead into a rough ball, then turn out on to a
```

1 cup yeast? That seems like a lot. Are you sure you don't mean 1 tbsp?

Fig 2: A screenshot of the modern lightweight code review tool, Gerrit, showing its web based interface.



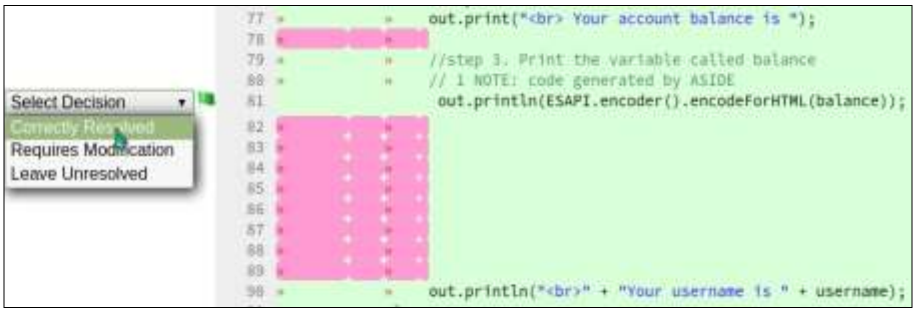
```
        out.println("<h1>Welcome " + firstname + " " + lastname + "</h1>");
        out.print("<br> Your account balance is ");

        //step 3. Print the variable called balance
        // 1 NOTE: code generated by ASIDE
            out.println(balance);



        out.p
    }


}

}
```

20-ASIDE Unsanitized Output Vulnerability
201-Encode for HTML
202-Encode for HTML Attribute
203- Encode for JavaScript
204- Encode for CSS
21-Output Encoding: Explanations and Examples
22-Disable Output Encoding
> - Options below are not ASIDE generated

This code uses the 'println' method from the PrintWriter to output data returned by "balance" to the application. If this is not sanitized, attackers can insert arbitrary malicious code which will be executed when this data is displayed. This attack is known as a Cross-Site-Scripting attack.

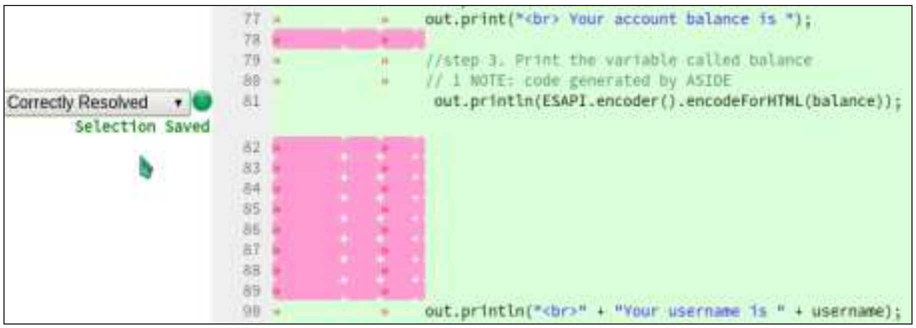
Use the options below to generate code to sanitize data, or read more information about this vulnerability and how to fix it.

Fig 3: A screenshot showing an ASIDE Interactive Static Analysis IDE warning being resolved prior to a security code review



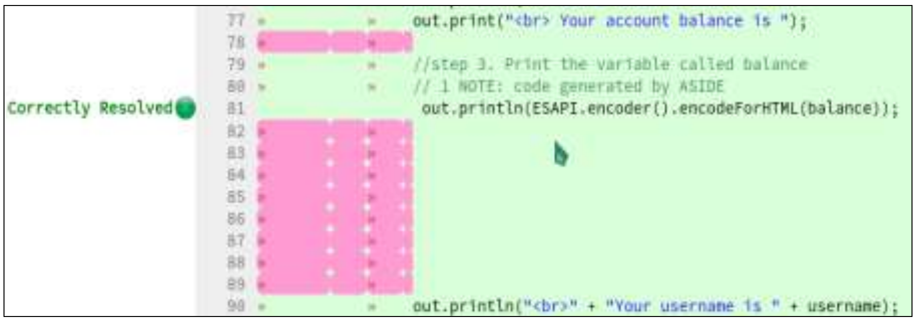
```
out.print("<br> Your account balance is ");

//step 3. Print the variable called balance
// 1 NOTE: code generated by ASIDE
 out.println(ESAPI.encoder().encodeForHTML(balance));

out.println("<br>" + "Your username is " + username);
```

Fig 4: A security expert marking a resolved interactive static analysis warning in a security code review



```
out.print("<br> Your account balance is ");

//step 3. Print the variable called balance
// 1 NOTE: code generated by ASIDE
 out.println(ESAPI.encoder().encodeForHTML(balance));

out.println("<br>" + "Your username is " + username);
```

Fig 5: Security expert interface after a developer-resolved interactive static analysis warning is marked as correctly resolved



```
out.print("<br> Your account balance is ");

//step 3. Print the variable called balance
// 1 NOTE: code generated by ASIDE
 out.println(ESAPI.encoder().encodeForHTML(balance));

out.println("<br>" + "Your username is " + username);
```

Fig 6: Developer view of an interactive static analysis warning marked as correctly resolved by the security expert during a security code review

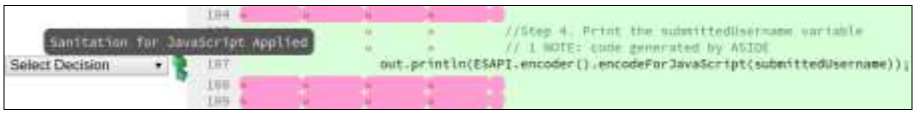
Prototype Warning Details

The tool design supports all existing ASIDE warnings. These warnings represent all of the major categories of web security threats and include SQL Injection, Cross Site Scripting, Cross Site Request Forgery, and Access Control vulnerability warnings. These are presented in the IDE to the left of the code in the primary developer's interface. These warnings are also present in the security expert's web interface. Warnings are presented in the same manner and style as my ASIDE tool, building on lessons learned from my numerous studies on interactive static analysis. The specific warnings present in the web interface are as follows:

- Red Flag. Presented to the security expert when the developer has ignored a static analysis warning
- Blue Flag. Presented to the security expert when the developer has marked a static analysis warning as a false positive

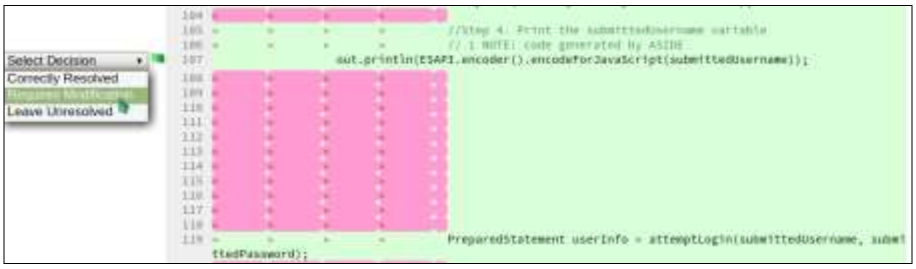
- Green Flag. Presented to the security expert when the developer has attempted to resolve a static analysis warning
- Red Devil. Presented to developers when a security expert has marked a warning as "Requires Modification"
- Yellow Orb. Presented to developers when a security expert has marked a warning as "Leave Unresolved"
- Green Orb. Presented to developers when a security expert has marked a warning as "Correctly Resolved"

Security experts and developers may contextually collaborate on the warnings at any time, regardless of whether or not the warnings are resolved. Security experts may also change their judgments of a warning at any time before the entire code review change has been completed.



```
                    //Step 4. Print the submittedUsername variable
                    // 1 NOTE: code generated by ASIDE
        out.println(ESAPI.encoder().encodeForJavaScript(submittedUsername));
```

Fig 7: Tooltip showing what type of sanitization was applied to a developer-resolved warning which was resolved incorrectly

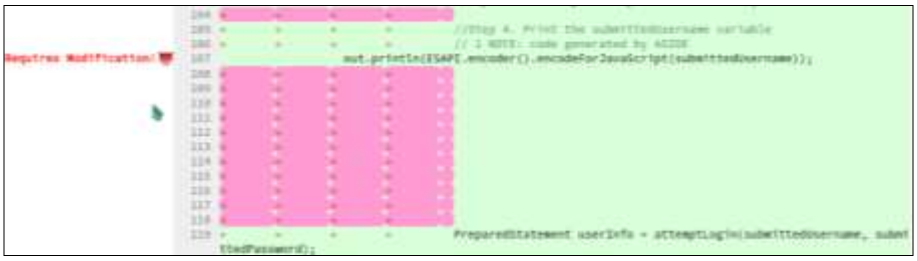


```
                    //Step 4. Print the submittedUsername variable
                    // 1 NOTE: code generated by ASIDE
        out.println(ESAPI.encoder().encodeForJavaScript(submittedUsername));
                    PreparedStatement userInfo = attemptLogin(submittedUsername, submit
tedPassword);
```

Fig 8: A security expert marking a developer-resolved warning which was resolved incorrectly

Collaboration

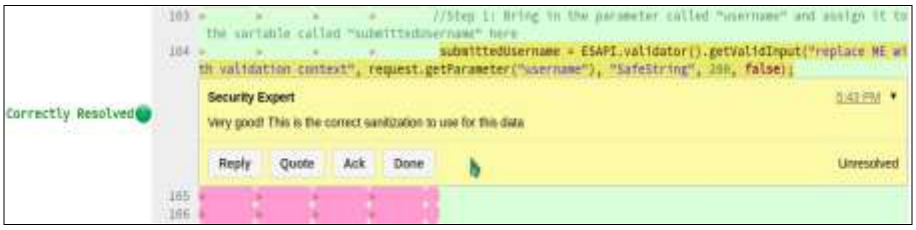
Collaboration is a key part of the tool's design. I have learned in several studies that developers wanted assurance as to whether or not their resolutions to vulnerability warnings and annotations were correct [10, 14, 13, 15]. Studies have also shown that security experts desire easily available records of vulnerabilities or issues [12]. The design of this tool should enable developers and security experts to communicate easily. It also groups records of this communication around the vulnerability warnings themselves. This enables users of the tool to avoid being overburdened with information which may come about as a result of single channel instant messaging. The records will also be retrievable later for the creation of reports.



```
                    //Step 4. Print the submittedUsername variable
                    // 1 NOTE: code generated by ASIDE
        out.println(ESAPI.encoder().encodeForJavaScript(submittedUsername));
                    PreparedStatement userInfo = attemptLogin(submittedUsername, submit
tedPassword);
```

Fig 9: Developer interface showing warning marked as requiring modification

Additionally, developers and experts have additional buttons called "ack" and "done." The quote button is the same as the reply button, except it also copies the prior response and quotes it. The "ack" button stands for acknowledge, and is a one-click reply. The "done" button is also a one click reply, but says "done" instead of "ack." Ack can be used to indicate that a message was received and understood, while "done" can be used to indicate that a resolution has been carried out.



```
                    //Step 1: Bring in the parameter called "username" and assign it to
the variable called "submittedUsername" here
                    submittedUsername = ESAPI.validator().getValidInput("replace ME wi
th validation context", request.getParameter("username"), "SafeString", 200, false);
```

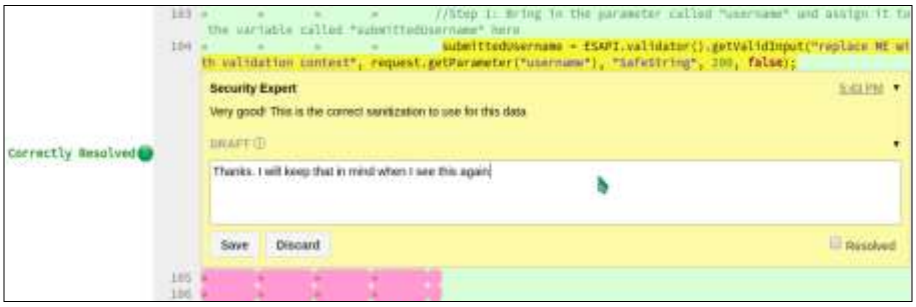
Security Expert 5:43 PM

Very good! This is the correct sanitization to use for this data

Fig 10: Screenshot showing the interface for contextualized warning collaboration

Implementation

Implementation of these features was achieved by the use of several key pieces of infrastructure. The Eclipse plugin of the primary developer submits the code to a



```
//Step 1: Bring in the parameter called "username" and assign it to the variable called "submittedUsername" here
submittedUsername = ESAPI.validator().getValidInput("replace ME with validation context", request.getParameter("username"), "SafeString", 200, false);
```

Security Expert

Very good! This is the correct sanitization to use for this data

Thanks. I will keep that in mind when I see this again

Fig 11: Developer replying to contextualized comment from security expert

git repository running Gerrit when the developer is ready to request a code review. When this occurs, the Eclipse plugin also submits HTTP requests containing the state of the user's annotations, interactions, and warnings. These calls will be made to a separate web and application server, which will accept and store the updates. The developer then logs into the modified Gerritt through a web based interface and selects a security expert to conduct a security code review. The source code of Gerritt was modified to include additional Javascript files and calls to those files. From this point on, tool-related logic is executed by custom Java script included within Gerritt. When security experts perform security code review, they login to Gerrit's web based interface and begin the reviewing process. When they do so, the custom Javascript modifies the HTML produced by Gerrit and shows additional functionality not provided by Gerritt. This includes warning-contextualized comments, warnings themselves, and the solutions the primary developer has chosen. When the security expert interacts with these additional elements, the integrated web and application server receives the input and updates its database. The custom Java script files pull updates from the web and application servers, using AJAX, and the HTML is modified to display these updates. When the review is finished, the final changes are submitted through Gerrit and the application and web servers maintain the additional content in their database for later retrieval and display.

Conclusion
Although building a working prototype design and tool was a significant endeavor, it provides many research contributions. The most important of these contributions is the ability to study the security code review process and find answers to my key research questions. With this tool, it will be possible to study the interactions of participants in various roles during code review. It is also possible to determine which features in a code review tool are useful for collaboration and how feedback from the security code review process should be displayed to users in all three roles. Lastly, the tool itself is a small research contribution, since other researchers can expand on the tool and use it for further security code review studies. Companies may also be able to build a commercial version of the tool and use it in their own security code review practices.

Acknowledgements
I would like to thank Heather R Lipford for her contributions to this work. I would also like to thank the Graduate Assistance in Areas of National Need (GAAN) Fellowship for its support of this work.

References
1. Gerrit. Introduction; c2023. https://wiki.qt.io/Gerrit Introduction.
2. Anderson P, Reps T, Teitelbaum T, Zarins M. Tool support for fine-grained software inspection. Software, IEEE. 2003;20(4):42-50.
3. Balachandran V. Reducing human effort and improving quality in peer code reviews using automatic static analysis and reviewer recommendation. In Proceedings of the 2013 International Conference on Software Engineering, ICSE '13, {940, Piscataway, NJ, USA, IEEE Press; c2013. p. 931.
4. Bosu JC, Carver M, Hafiz P, Hilley, Janni D. Identifying the characteristics of vulnerable code changes: An empirical study. In Proceedings of the 22Nd ACM SIGSOFT International Symposium on Foundations of Software Engineering, FSE, New York, NY, USA, 2014. ACM; c2014. p. 257-268.
5. Fagan ME. Design and code inspections to reduce errors in program development. IBM Syst. J. 1999;38(2-3):258-287.
6. McIntosh S, Kamei Y, Adams B, Hassan AE. The impact of code review coverage and code review participation on software quality: A case study of the qt, vtk, and itk projects. In Proceedings of the 11th Working Conference on Mining Software Repositories, MSR, New York, NY, USA. ACM; c2014. p. 192-201.
7. Meneely ACR, Tejeda B, Spates S, Trudeau D, Neuberger K, Whitlock C, *et al.* An empirical investigation of socio-technical code review metrics and security vulnerabilities. In Proceedings of the 6th International Workshop on Social Software Engineering, SSE, New York, NY, USA, ACM; c2014. p. 37-44.
8. Panichella S, Arnaoudova V, Di Penta M, Antoniol G. Would static analysis tools help developers with code reviews? In Software Analysis, Evolution and Reengineering (SANER), 2015 IEEE 22nd International Conference on; c2015. p. 161-170.
9. Sadowski C, van Gogh J, Jaspan C, Soderberg E, Winter C. Tricorder: Building a program analysis ecosystem. In Proceedings of the 37th International Conference on Software Engineering - ICSE '15, Piscataway, NJ, USA, IEEE Press. 2015;1:598-608.
10. Thomas T, Chu B, Lipford H, Smith J, Murphy-Hill E. A study of interactive code annotation for access control vulnerabilities. In Proceedings of the 2015 IEEE Symposium on Visual Languages and Human-Centric Computing, VLHCC '15, Washington, DC, USA, IEEE Computer Society; c2015.
11. Thomas TW, Tabassum M, Chu B, Lipford H. Security during application development: An application

security expert perspective. In Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems, CHI '18, Montreal QC Canada, ACM; c2018. p. 262:1-262:12.

12. Werlinger R, Hawkey K, Beznosov K. Security practitioners in context: Their activities and interactions. In CHI '08 Extended Abstracts on Human Factors in Computing Systems, CHI EA '08, New York, NY, USA, ACM; c2008. p. 3789-3794.
13. Xie J, Chu B, Lipford HR, Melton JT. Aside: Ide support for web application security. In Proceedings of the 27th Annual Computer Security Applications Conference, ACSAC '11, pages 267{276, New York, NY, USA, ACM; c2011.
14. Xie J, Lipford H, Chu BT. Evaluating interactive support for secure programming. In Proceedings of the SIGCHI Conference on Human Factors in Computing Systems, CHI '12, New York, NY, USA, ACM; c2012. p. 2707-2716.
15. Zhu J, Xie J, Lipford HR, Chu B. Supporting secure programming in web applications through interactive static analysis. Journal of Advanced Research. 2014;5(4):449-462.