Designing Interventions to Persuade Software Developers to Adopt Security Tools

Brooke Jordan, Brittany Johnson, Jim Witschey, and Emerson Murphy-Hill
North Carolina State University
Raleigh, NC 27606
{tbjorda2, bijohnso, jwshephe}@ncsu.edu, emerson@csc.ncsu.edu

ABSTRACT

The nature of security information workers' jobs requires a certain level of care and attention to detail. There exist tools that can assist these workers with their daily tasks; however, workers may not be using these tools. Research suggests persuasive techniques can positively affect a worker's outlook on a given technology. We attempt to develop an effective way to motivate security workers to adopt and use security tools by using persuasive design guidelines. We present an system that generates automated emails to inform software developers of FindBugs, a tool that detects potential vulnerabilities within a project. We discuss the decisions supporting our overall design of the automated emails.

1. INTRODUCTION

Security information workers deal with several issues within their work, one being maintaining the integrity of the data or source code they create or modify. Lack of awareness regarding the potential vulnerabilities in a project or program can lead to security threats that could effect an entire system. In order for work to be secure, workers need to understand where vulnerabilities exist. There exists tools that can be effectively detect or prevent these vulnerabilities, however, these tools are only useful if they are used.

Software developers, intelligence analysts, and system administrators require different types of security tools within their respective fields. Developers could use program analysis tools that detect potential security vulnerabilities in code, such as FindBugs.[1] Intelligence analysts might use tools, such as *flash fill* in Excel,[2] to prevent future vulnerabilities. Finally, system administrators could use tools, like *BelSecure*, to run a vulnerability assessment of their system.[3] In this paper, we focus on security tools used by software developers. However, this does not mean our approach cannot be applied to other security information workers. In Section 6, we show while our work currently focuses on software developers it can be generalized for other security workers to adopt relevant tools to their work.

[1]http://findbugs.sourceforge.net/
[2]http://blogs.office.com/2012/08/09/flash-fill/
[3]http://www.belarc.com/belsecure.html

SIW'14, November 7, 2014, Scottsdale, Arizona, USA.
Copyright is held by the owner/author(s). Publication rights licensed to ACM.
ACM ACM 978-1-4503-3152-4/14/11 $15.00
http://dx.doi.org/10.1145/2663887.2663900.

We plan to use persuasive technology to motivate developers to adopt security tools. Persuasive technology is designed to change behaviors of users through persuasion and social influence [8]. It has been used to motivate various behavior changes, suggesting this is an effective approach to potentially increasing tool adoption [9, 6, 10]. Persuasion involves motivating a worker and increasing the ability for a behavior change [8]. We have created a system we believe will achieve our goal to persuade software developers to adopt security tools.

Our approach involves emailing developers contributing to open source projects regarding the recent removal of vulnerabilities that may or may not have been purposefully removed. We chose to analyze projects that commit enough changes to produce results, in the form of resolved vulnerabilities. The developer will be presented with our results and a simple introduction to a tool and its usefulness.

The contribution of this paper is the design of a system that creates emails based on a set of design guidelines, developed from previous persuasive technologies, for persuading security workers to adopt security tools. We begin with a discussion of related work about persuasion theory and motivating developers to change behavior. Section 3 details the implementation of our system. Section 4 explains our methodology. In Section 5 we present our design decisions broken-down into subsections detailing specific choices. Section 6 describes an example of how our system can be related to other security information workers. Lastly, we discuss our plans for future work.

2. RELATED WORK

Research involving persuading software developers, a subset of security information workers, to complete a requested action or change their behavior exists [5, 6, 10]. Much of this research is based on Fogg's persuasion research [4, 3]. Pribik and Felfernig developed and evaluated a persuasive software system, named PerSoDeMetrics, to study its effects on persuading developers to adopt and use software metrics tools [8]. The system provides recommendations for improving code maintainability through calculations of various software metrics. Their system was built based on dimensions relevant to changing behavior: trigger, availability, and motivation [4]. They found their tool had a positive effect on software metrics tool adoption and usage in a small real-world development team.

Johnston and Warkentin developed a conceptual model to show the influences of source credibility on users' attitudes towards information technology recommendations [6]. They found factors such as ability to trust the source determine engagement in recommended information technology actions, emphasizing the importance of communication in the persuasion process. Smith and colleagues presented factors for consideration when using emails to persuade developers to participate in surveys. Using factors from persuasion research, they analyzed recruitment emails from already completed surveys to determine which led to more success and why. Effective emails included at least two of the factors relevant to persuasion research, such as authority and similarity [10].

Similar to these studies, we use findings from Fogg's persuasion research to motivate behavior change. Our work builds on existing research, such as the ones discussed above, by attempting to apply persuasion research findings to motivating software developers' adoption of tools that find security vulnerabilities.

3. IMPLEMENTATION

The goal of our system is to persuade software developers who are not currently using security tools to use a security tool within their project. We have chosen FindBugs as our tool to analyze these projects. FindBugs can be used to find vulnerabilities, although it is not exclusively used in this manner, so we consider it a security tool for our purposes. Our approach currently involves searching for appropriate repositories, building each, and analyzing with FindBugs. We use Jenkins, a continuous integration system, to build and analyze the projects.[4] Each project is cloned into Jenkins using its GitHub repository URL. Jenkins gives the option to poll a repository, with set intervals between each poll; if Jenkins polls the repository and changes have been made in the form of new commits, Jenkins builds the project. We currently poll the repositories every 15 minutes. We chose this time interval based on the highest frequency of commits in a given day across projects.

We chose FindBugs as our security tool; it is a mature tool and can be easily added to projects within Jenkins. To include FindBugs in our build process, we altered each project's build. During each new build, FindBugs is executed and outputs a file, `findbugs.xml`, containing the list of bugs found. A script outputs the differences between the last build's `findbugs.xml` and the current build's `findbugs.xml` file; the output of this diff is then emailed to us. However, this email is only sent if there was information regarding the removal of a FindBugs warning. We currently retrieve relevant information, such as class name, line of code, and type of bug, to look at the source code and determine if the fix from the FindBugs report is a true fix. We discuss this process in more detail in Section 4.

4. METHODOLOGY

With our properly configured system, the next step was to find projects to use. We selected projects that are large and have frequent commits. Below, we outline how we selected projects and remove false reports.

[4]http://jenkins-ci.org/

4.1 Projects

We searched for large projects on GitHub that did not already use FindBugs. We chose larger projects because we believe larger projects were more likely to include vulnerabilities. Furthermore, larger projects may have more contributors, which increased the chances of new commits, new security vulnerabilities, which manifest themselves as bugs in the software code, and removed vulnerabilities. With these projects, we found approximately one build a day where one or more bugs have been removed.

4.2 False Postives

We plan to send an email to a developer when he fixes a bug. However, the removal of a FindBugs warning does not always mean the vulnerability was fixed. We have come across instances where a bug fix was reported but once we checked the code we noticed an entire method or class was deleted or commented out. Since we did not know the reasons behind this, we chose to ignore these fixes. We also ran into an issue where FindBugs would report new and fixed bugs that turned out to be the same bugs. While we are uncertain on the cause of this issue, we have reduced its effect by using a difference tool to report only those bugs that are not found in the current build. Before we create an email for the developer, we use this information to verify the results from FindBugs.

5. DESIGN DECISIONS

To effectively persuade developers to adopt tools encouraging and facilitating secure practices, we need to determine an effective design for our emails. We plan to achieve our goal of motivating tool adoption by applying the following design guidelines:

- Send *personalized emails* to increase ownership and build trust.
- Use *emails sent by a human* as the trigger to change.
- When possible, include *information concerning peer behavior or credibility* of the moderator.
- Consider *planned timing* when sending the emails.
- Use *positive reinforcement* to encourage behavior change.

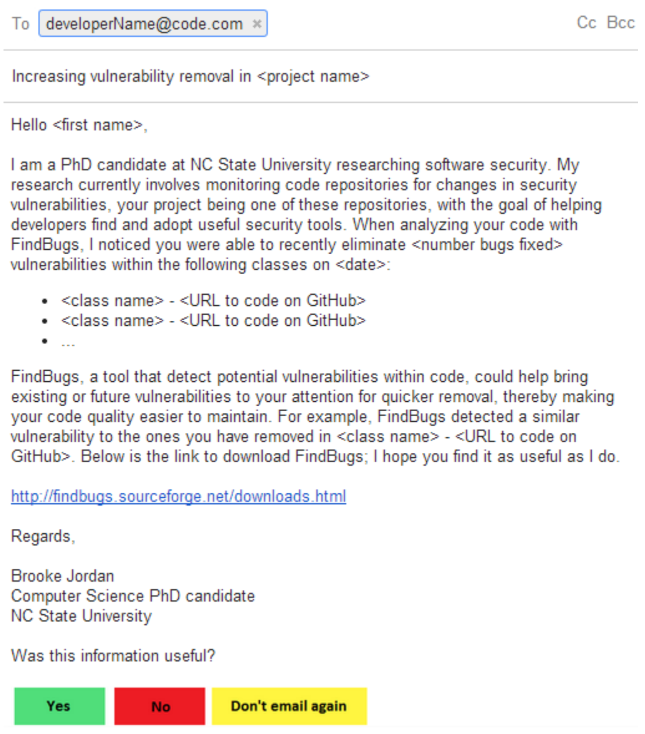
Though our implementation currently focuses on software developers, a subset of security workers, the design decisions discussed can be generalized beyond developers (Section 6).

5.1 Mass Distribution vs. Personalized Emails

Research suggests that emails sent directly to developers are more effective in generating a response than mass produced emails [10]. This informed our first design decision: to make developers feel that we personally reached out to them rather than email the entire group of contributors for each project. Previous research indicates developers are more likely to respond to targeted study solicitations using their names than to impersonal appeals [7]. Thus, we have chosen to use the developer's name and information regarding his recent commit to personalize the email.

5.2 Increasing Ability vs. Motivation

Two dominant, sometimes competing, factors in changing behavior are motivation and ability [3]. Motivation, which is intrinsic, greatly depends on the individual; this makes


To developerName@code.com Cc Bcc

Increasing vulnerability removal in <project name>

Hello <first name>,

I am a PhD candidate at NC State University researching software security. My research currently involves monitoring code repositories for changes in security vulnerabilities, your project being one of these repositories, with the goal of helping developers find and adopt useful security tools. When analyzing your code with FindBugs, I noticed you were able to recently eliminate <number bugs fixed> vulnerabilities within the following classes on <date>:

- <class name> - <URL to code on GitHub>
- <class name> - <URL to code on GitHub>
- ...

FindBugs, a tool that detect potential vulnerabilities within code, could help bring existing or future vulnerabilities to your attention for quicker removal, thereby making your code quality easier to maintain. For example, FindBugs detected a similar vulnerability to the ones you have removed in <class name> - <URL to code on GitHub>. Below is the link to download FindBugs; I hope you find it as useful as I do.

http://findbugs.sourceforge.net/downloads.html

Regards,

Brooke Jordan
Computer Science PhD candidate
NC State University

Was this information useful?

Yes No Don't email again

Figure 1: An email designed based on our guidelines.

determining the best method for motivating behavior change inherently difficult. Ability is extrinsic; outside forces can more easily influence the ability to change the behavior. Thus, our focus is the email increases the ability to change behavior rather than increases motivation. In our current implementation, we increase ability to change behavior by providing the URL for the tool, as seen in Figure 1.

5.3 Emails from a System vs. Emails from a Human

According to persuasion research, an important part of a persuasive system is the trigger used to change the target audience's behavior [3]. If there exists sufficient motivation and ability, a trigger is all that is needed to cause a behavior change. In our case, the trigger is an email. Fogg suggests a human trigger is better than a system; however, it is infeasible for us to visit every developer contributing to a project every time they change the vulnerability count. This informed our next design decision: to have the emails drafted by a system then verified and sent by a human. Though email creation is automated in our current implementation, the first author acts as email moderator; this entails manually inspecting each email for accuracy prior to sending. For example, if the email moderator noticed a vulnerability was fixed by a change unrelated to bug or security fixes, such as the removal of an obsolete feature, the moderator could choose not to send an email. This type of moderation can only be done by a human.

5.4 General Developer Behavior vs. Peer Behavior

Developers are generally more willing to take advice from their peers or persons they deem credible than they are from someone with whom they are unfamiliar [5, 6, 10]. They are also more likely to listen to someone with which they are on good terms [10]. This led us to our next design decision: to use phrasing, and possibly data, that suggests credibility or familiarity. For example, to convince a developer that we are trustworthy and credible, we mention in the email our status as doctoral candidates at NC State performing research on security tools (Figure 1). We understand as a Ph.D. student, particularly one with no professional or open source development experience, may not be considered a peer to developers; when applying this guideline, it is important to consider who the target audience will be. Our current implementation further attempts to increase trustworthiness and credibility by including details from generated reports, presented for readability. This gives the developer all he needs to verify the validity of the report and possibly even the tool that has been recommended.

5.5 Planned Timing vs. Immediate Emails

Timing is another important aspect to motivating a change in behavior [4]. Smith and colleagues noticed higher response rates on Monday in the late afternoon and Wednesday early afternoon when using emails to recruit developers to complete a survey, something that requires time just as installing or adopting a new tool [10]. Our research pertains to potentially reoccurring events, such as writing and maintaining code, rather than events that typically occur once, like a filling out a survey. Because of this, we believe it is better to send emails mid–afternoon every day rather than right after a vulnerability has been removed. This also helps with overwhelming the developer with a large number of emails at once. Majority of the users on GitHub who commit to these projects have listed their location, either state or country, on their user profile. We plan on using this information to send an email at their mid–afternoon time relative to their location. Our next design decision was as follows: to time the sending of emails per person's location rather than sending emails multiple times throughout the day.

5.6 Positive vs. Negative Reinforcement

Another aspect to persuasion is the use of positive or negative reinforcement [5, 6]. Research suggests that positive reinforcement is more effective in a persuasive system than negative reinforcement. This led to the design decision to include positive reinforcement in our emails. For example, telling a developer that they are unprofessional for not using a tool and if they do not use a tool their code will never work would be a form of negative reinforcement. Our approach to incorporate positive reinforcement is, as we present in Figure 1, to tell the developer how obvious it is they care about the quality of the code and then recommend the tool(s) that could make it easier to maintain that quality. We believe the likelihood that the email is taken seriously and action is taken in the form of tool adoption will increase.

6. GENERALIZING BEYOND DEVELOPERS

Though our current implementation caters to software developers, we believe our approach can be applied to any security information worker. Consider system administrators, whose responsibilities include ensuring system security through managing system settings and firewalls [2]. With the various tasks involved in maintaining system security, ensuring a system is secure can be a difficult process. One tedious and error–prone task system administrators are responsible for is creating and maintaining rulebases for system firewalls; as rulebases grow, they become more susceptible to vulnerabilities [1]. For example, an administrator has been assigned to make changes to an existing firewall rulebase and there is a system, like the one presented in this paper, in place. As the admin makes changes to a firewall rulebase, she realizes that there is a potential security vulnerability across two of the system's firewalls. She makes the changes to remove the vulnerability; when she does, the persuasive system realizes this change and that it removed a security vulnerability. A human moderator receives an email with information relevant to the vulnerability removed and then sends an email to the author of those changes including a recommendation to use a tool to help prevent the vulnerability and find others like it; in this case, we might recommend *Skybox Firewall Assurance*, firewall management software that helps admins find and remove system firewall vulnerabilities.[5]

7. FUTURE WORK

The decisions documented here are the first step toward future studies in which we can use our design to study the effects of persuasive interventions on security tool adoption. Though we cannot run a controlled experiment using this technique, we can use case studies or quasi–experimental study designs to evaluate the success of our intervention. The emails will be addressed to the developer who fixed a vulnerability within a project. They will follow the guidelines of our design decisions from Section 5. The design is still a work–in–progress and may incur changes. For example, we may need to evaluate whether the moderator's title or experience effects a developer's trust in the email's message before evaluating the message as a whole. Once we evaluate our overall design, we can then see if the developer chooses to adopt the tool by noticing if the build file changes to include FindBugs or a similar tool.

We also plan to monitor specific branches of a project. Our thought is that one branch of a project may use a security tool while another may not. We could clone one of those branches, then introduce the tool into the branch where it is not present. Monitoring that branch to see if there are any results from the tool could be used to mention to the developer the usefulness of the certain tool. Showing the developer proof of a co–developer using the tool also allows for the use of peer behavior as another persuasive tactic, increasing the likelihood of the developer to adopt the tool.

8. CONCLUSION

Secure code and data is very important to all security information workers. The goal of our research is to persuade software developers to adopt security tools. We have presented our overall design for our system that we plan to use to motivate software developers, a subset of security information workers, to adopt the usage of security tools. We believe that by injecting FindBugs to various projects cloned from GitHub and emailing the developers will be enough persuasion to allow for this adoption. We mentioned and discussed the current design of our emails and the reasoning behind each of these decisions. While we have no immediate data on the adoption of FindBugs, we plan to continue our work and using our template, directly email the developers. We may alter our template if feedback from developers does not follow our expectations. We expect to influence the developers to use FindBugs in their own code and find our emails useful.

[5]http://go.ncsu.edu/wsiw14-firewall

9. ACKNOWLEDGMENTS

We would like to thank the Developer Liberation Front for assistance with paper preparation. Thanks to the National Security Agency for funding this research. This material is based upon work supported by a National Science Foundation grant No. 1318323 and a Graduate Research Fellowship under Grant No. DGE 0946818.

10. REFERENCES

[1] B. Anderson. Check Point firewalls – rulebase cleanup and performance tuning. Technical report, SANS Institute, 2008.

[2] D. Farmer and E. H. Spafford. The cops security checker system. 1990.

[3] B. Fogg. A behavior model for persuasive design. In *Proceedings of the 4th international Conference on Persuasive Technology*, page 40. ACM, 2009.

[4] B. J. Fogg. Persuasive technology: using computers to change what we think and do. *Ubiquity*, 2002(December):5, 2002.

[5] T. Herath and H. R. Rao. Encouraging information security behaviors in organizations: Role of penalties, pressures and perceived effectiveness. *Decision Support Systems*, 47(2):154–165, 2009.

[6] A. C. Johnston and M. Warkentin. The influence of perceived source credibility on end user attitudes and intentions to comply with recommended it actions. *Journal of Organizational and End User Computing (JOEUC)*, 22(3):1–21, 2010.

[7] E. Murphy-Hill, T. Zimmermann, C. Bird, and N. Nagappan. The design of bug fixes. In *Software Engineering (ICSE), 2013 35th International Conference on*, pages 332–341. IEEE, 2013.

[8] I. Pribik and A. Felfernig. Towards persuasive technology for software development environments: an empirical study. In *Persuasive Technology. Design for Health and Safety*, pages 227–238. Springer, 2012.

[9] L. Singer. *Improving the adoption of software engineering practices through persuasive interventions.* PhD thesis, Ph. D. dissertation, Gottfried Wilhelm Leibniz Universität Hannover, 2013.

[10] E. Smith, R. Loftin, E. Murphy-Hill, C. Bird, and T. Zimmermann. Improving developer participation rates in surveys. In *Cooperative and Human Aspects of Software Engineering (CHASE), 2013 6th International Workshop on*, pages 89–92. IEEE, 2013.