Toward Effective Adoption of Security Practices

Shams Al-Amin†, Nirav Ajmeri‡, Emily Berglund†, Jon Doyle‡, Munindar P. Singh‡
†Department of Civil, Environmental, and Construction Engineering, ‡Department of Computer Science
North Carolina State University,Raleigh, NC 27695
{salamin,najmeri,emily_berglund,Jon_Doyle,mpsingh}@ncsu.edu

ACM Reference format:
Shams Al-Amin†, Nirav Ajmeri‡, Emily Berglund†, Jon Doyle‡, Munindar P. Singh‡ . 2017. Toward Effective Adoption of Security Practices. In *Proceedings of HotSoS Symposium and Bootcamp, Hanover, Maryland USA, April 2017 (HotSoS'17),* 1 pages.
DOI: 10.475/123_4

1 INTRODUCTION

Security tools guide developers to identify potential vulnerabilities in their codes. However, use of security tool is not very common [2]. Sanctions are a way to enforce adoption of security practices. We address the research question of which sanctioning mechanism promotes adoption of security practices, and propose a simulation framework to explore sanctioning mechanisms [1] for greater adoption.

Example 1.1. Alex, Bob, Charlie and Dave are a team of software developers working to deliver a product. Erin is their manager. Erin divides the project into multiple tasks, and assigns those to the developers. Erin also makes sure the product comply to the timeliness, functionality and security requirements. The team uses certain security tools to identify security bugs. Alex and Bob are experienced developers and aware of security tools, whereas Charlie and Dave are new.

Consider a scenario where, Charlie and Dave learn to use security tools. Everyone follows the standard security practices and delivers the product in time. Erin evaluates the functionality and security requirements, and finds those as satisfactory. She rewards her team for compliance.

Now consider a time-critical project where Charlie skips executing security tools, and delivers the product only with functionality. This can lead to multiple outcomes: (1) Erin identifies that Charlie did not follow the standard security practices, and scolds him. (2) Erin evaluates the product, and finds the security unsatisfactory. She scolds her team for non-compliance. (3) Alex finds that Charlie is not following the security practices, and prompts him to comply. The illustrative example distinguishes *individual*, *group* and *peer* sanctions. Erin sanctioning Charlie is an example of *individual sanction* where the manager sanctions individuals who do not follow a *norm*. Erin scolding everyone is a *group sanction* where the sanctions are imposed on a group regardless who violated or satisfied a *norm*. Alex prompting Charlie is a *peer sanction* where a peer sanctions another peer.

Permission to make digital or hard copies of part or all of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for third-party components of this work must be honored. For all other uses, contact the owner/author(s).
HotSoS'17, Hanover, Maryland USA
© 2017 Copyright held by the owner/author(s). 123-4567-24-567/08/06...$15.00
DOI: 10.475/123_4

2 SIMULATION FRAMEWORK

We develop a framework that simulates the dynamic interactions between developers and a manager in a time-critical project. The manager assigns task to the developers; each of who have heterogeneous skills of coding and running security tests. Each task corresponds to an artifact of a product. An artifact is considered functional when the corresponding task is coded by a developer. It is more secured if a developer further tests it with a security tool. The expected utility of a developer is derived from the functionality and security of the artifacts of his tasks. Some developers might need to learn before they can code or run security tests, if they lack required skills for a task. Learning increases the skills of a developer, and reduces the time taken to code or run security tests. Learning also increases the functionality and security of the corresponding artifacts. A rational developer makes decisions to maximize his utility.

A manager can impose sanctions for timeliness, functionality or security. As illustrated in the example, by offering rewards (positive sanction) or punishments (negative sanction), a manager can change the expected utilities of the sanctioned developers. For example, on receiving a sanction for functionality, the perceived utility of learning to code, and to code a task may increase for a developer. Similarly, a sanction for security can increase the utility a developer perceives for learning to run, or doing security tests.

Our framework aims to improve the understanding of adoption of security practices among developers. It assists in comparing functionality and security compliance in projects under different sanctioning regimes, by (1) analyzing the recurrence of norm violation after sanctioning, referred to as sanction efficacy, and (2) computing system liveliness and resilience.

ACKNOWLEDGMENTS

We thank the US Department of Defense for support through the Science of Security Lablet grant to NC State University.

REFERENCES

[1] L. G. Nardin, T. Balke-Visser, N. Ajmeri, A. K. Kalia, J. S. Sichman, and M. P. Singh. Classifying sanctions and designing a conceptual sanctioning process model for socio-technical systems. *KER*, 31:142–166, Mar. 2016.
[2] J. Witschey, O. Zielinska, A. Welk, E. Murphy-Hill, C. Mayhorn, and T. Zimmermann. Quantifying developers' adoption of security tools. In *Proc. FSE*, pages 260–271, Bergamo, Italy, 2015. ACM.