An Insider Threat Activity in a Software Security Course

Daniel E. Krutz, Andrew Meneely, and Samuel A. Malachowsky
Rochester Institute of Technology
{dxkvse, axmvse, samvse}@rit.edu

***Abstract*—Software development teams face a critical threat to the security of their systems: insiders. A malicious insider is a person who violates an authorized level of access in a software system. Unfortunately, when creating software, developers do not typically account for insider threat. Students learning software development are unaware of the impacts of malicious actors and are far too often untrained in prevention methods against them. A few of the defensive mechanisms to protect against insider threats include eliminating system access once an employee leaves an organization, enforcing principle of least privilege, code reviews, and constant monitoring for suspicious activity.**

At the Department of Software Engineering at the Rochester Institute of Technology, we require a course titled Engineering of Secure Software and have created an activity designed to prepare students for the problem of insider threats. At the beginning of this activity, student teams are given the task of designing a moderately sized secure software system. The goal of this insider is to manipulate the team into creating a flawed system design that would allow attackers to perform malicious activities once the system has been created. When the insider is revealed at the conclusion of the project, students discuss countermeasures regarding the malicious actions the insiders were able to plan or complete, along with methods of prevention that may have been employed by the team to detect the malicious developer.

In this paper, we describe the activity along with the results of a survey. We discuss the benefits and challenges of the activity with the goal of giving other instructors the tools they need to conduct this activity at their institution. While many institutions do not offer courses in computer security, this self-contained activity may be used in any computing course to enforce the importance of protecting against insider threats.

Keywords*—*Software Security, Software Engineering, Computing Education

I. Introduction

Organizations devote vast amounts of time and resources protecting themselves against outside threats. These protection mechanisms include but are not limited to firewalls, data encryption, and defensive coding practices. A much more difficult threat to protect against is one which comes from within the company. An *insider threat* is a current or former employee, business partner, or contractor who has access to an organization's data, network, source code, or other sensitive information who may intentionally misuse this information and negatively affect the availability, integrity, or confidentiality of the organization's information system. Examples of insider attacks include data harvesting, abuse of privileges sabotage, and masquerading attacks. Additionally, a user could act as an insider threat and have no actual malicious intent. Inadvertent use of computing resources or data could make a system susceptible to attack, or could unintentionally release sensitive data [6]. Insider threats are much more difficult to identify since potential threats are often users of a system which they themselves developed (potential leaving back doors and having deep intimate knowledge of the system) while, as developers, not being considered to be potential threats to the system [12]. Companies often choose mitigate the risk of insider threats through the use of policies and regulations [7].

Far too often, software engineers are not prepared to deal with fact that their co-workers, people who they should trust as allies, are in fact potentially malicious users capable of performing a wide range of destructive actions. In educational environments, students typically either work alone, or with teams — with instructors preaching the importance of teamwork and trust. While these concepts are important real-world examples dictate that the opportunity for a security threat must be considered.

At the Rochester Institute of Technology, we created an upper division Engineering of Secure Software applications course to help students understand how to incorporate proper security protection practices when designing, creating, and maintaining software. Some course topics include defensive coding practices, deployment & distribution strategies, vulnerability assessments, and threat modeling. In this course, we created an activity to help acclimate students with how to understand, protect, and recognize insider threats. Students teams are formed and students are given the task of designing a small application and planning for proper security. One student from each team is quietly pulled aside and told that they are the insider threat or *mole* for their team. Their goal is to have their team design an application containing a vulnerability which the mole would be able to later use. After the activity, the moles are revealed and a discussion takes place regarding how the moles were able to create the vulnerability, if the team recognized this vulnerability, and what could have been done to prevent this insider threat.

The rest of the paper is organized as follows: Section II describes the course including learning objectives. Section III discusses how the activity was conducted. Section IV provides student feedback about the project including quotes and post activity survey analytics. Section V presents some related works and Section VI discusses possible future work and improvements to the activity. Section VII provides concluding remarks about our work.

II. About the course

Primarily comprised of upper division Software Engineering students, the Engineering of Secure Software course[1] was created in 2012 and is focused on instructing students in the proper practices of design and creating secure software. The only prerequisite is the Introduction to Software Engineering course in which students are introduced to core concepts in software engineering such as development methodologies, team work in software development, basic testing principles, and software design.

The course has a primary learning outcome of preparing students to mitigate security threats in software systems and processes. The focus is on proper methods of designing, developing, testing, and maintaining secure software. While the course is language-agnostic and focuses on principles and practices, specific tools and technologies are used to reinforce the learning objectives of the course. For instance, Microsoft's SDL Threat Modeling Tool[2] is used to instruct students on the proper methods of designing the architecture of a secure system. Specific Java-based examples are used to demonstrate SQL injection attacks, log overflow attacks, hashing and salt, and path traversal exploits. Short Vulnerability-of-the-Day activities serve to introduce students to real world examples of exploits and demonstrate the importance of software security [11]. Students work in small teams on several course projects including the creation of a web fuzz testing tool and a case study which examines a real-world software project for vulnerabilities.

While we do not expect all students taking the course to become security experts, our goal is to instill fundamental principles of secure software development in the students while demonstrating its importance in the real world. Students are graded on several criteria such as three exams, several short projects, and brief in-class activities. Class size is typically 25-35 students and is a required course in the Software Engineering major.

In the course, we also discuss several ways of protecting against insider threats. While there is no easy or simple silver bullet protection mechanism against insider threats, there are some best practices which may be used to help alleviate this risk. Some of these protection practices include properly screening potential employees, implementing end point data leak protection, monitoring databases & sensitive records, and the proper use of rights management systems [15].

III. Insider Threat Activity

In this section we describe the sample project, the activity conducted by the students, provide example vulnerabilities which the insider threats were able to create, and finally describe goals of the post activity discussion.

A. Sample Project

To begin, students are told that they are being asked to design a secure software system. In our course instances, we asked them to design a student grading system, much like ones which are typically used at many institutions. We selected this example since we felt it was complex enough for students to have to actively consider numerous possible vulnerabilities and would allow for our insider threats to act maliciously, but simple enough for teams to design it in a class meeting or two. Additionally, we felt that students would be reasonably familiar with a system of this type.

Some basic requirements for this system were: Students should be able to view their grades, but not alter them or view the grades of their classmates. Instructors should be able to view the grades for all students, but only modify grades for students in the specific courses they were teaching. School administrators should be able to view and alter the grades of all students, at any time. All users are required to access the system using their username and password, and if any they have forgotten their password, the system should send a reminder to the person via email and allow them update the password from that link. The application also needed to be accessible from anywhere in the world using a basic web connection. A non-functional requirement was that the application database should be backed up on a nightly basis to an off-site location. For debugging purposes, it was required that all errors and user actions should be logged for system administrators.

B. Activity

Once tstudents were given the basic requirements for their system, student teams of 4-6 students are formed, as this is often the size of groups in industry and has been found to be conducive to student learning in previous research [8], [14]. Before the start of the activity, a subset students were emailed asking them to be insider threats. They were asked to not share this information with anyone and to act just like any other member of the team. Their goal was to have their team design a system that would leave the door open for them to act maliciously in some manner. Examples included being able to view other student's grades view at a higher level of access, or change data which they should not be able to modify.

Example sections of our courses were offered in a 50 minute long format with this activity spanning two class periods. Teams were allowed the entire first class period to work on designing their system while the instructor answered any requirements specific questions the teams had. Teams were then asked to work on the activity outside of class. At the beginning of the second class meeting, teams were asked to review their security design and make any last minute updates. Teams were then informally polled to see how many felt their system was secure. This serves to gauge the confidence each team has in their design. Based on the confidence of the teams and the instructor's knowledge of each team's design, the team with the highest confidence and best design was asked to briefly present their design to the class and talk about why they felt it was secure. The class then asked questions about the design and try to discover any vulnerabilities.

After this brief discussion, the insider threat student for the presenting team is asked to make themselves public and describe the vulnerability they had left in the system design. The insider threats for other teams are then asked to expose themselves and describe the vulnerability they introduced into their team's design as well.

[1] http://www.se.rit.edu/~swen-331/

[2] www.microsoft.com/security/sdl/adopt/threatmodeling.aspx

While the insider threats added vulnerabilities in a variety of ways, there were some commonalities in the methods chosen to ensure the application was exploitable. Most often, it was as simple as recommending poor security strategies to leave vulnerabilities in the system. In some instances, insiders would add a vulnerability into the team's design between class meetings which would not be caught during the review on the second class meeting. This mimics the real world situation of an insider adding malicious code into a project during off hours or without the knowledge of the rest of the team. Interestingly, insiders would often ensure that vulnerabilities existed in a system by simply doing nothing. They would notice their teammates leaving a vulnerability in the design due to a simple but honest mistake, doing nothing to fix this known issue.

C. Examples of Introduced Vulnerabilities

In order to provide some context regarding some of the introduced vulnerabilities, we will explain some of them in further detail and provide some examples of how they were used within the activity.

Not Limiting Failed Login Attempts
Limiting the number of failed login attempts its an important protection mechanism against brute force attacks. Without this limitation, attackers may systematically try password combinations until they achieve the correct password. This type of attack has been successfully made against a wide range of organizations, including Apple [13]. In one team, the insider made the argument that users should not be limited by their login attempts since this could be annoying to users and could hurt their user experience with the application. The team agreed and no limiting was done.

Publicly Saving Logs
A requirement of the system was to log all system errors and actions taken by users. Attackers could use these error logs to potentially gain information which could help them compromise the system. Some of this could include call stacks, information about how to create a log overflow, or any other sensitive information which was output to the log messages. Additionally, since the log files contained many alterations make to the system, this could expose potentially sensitive changes to malicious users such as the grades of students. In one team, the insider argued that these logs should be publicly available since it would provide easier access to developers when troubleshooting issues. On another team, they placed the generated log files outside of the trust zone of the system, which would expose these files to outside users. Their teams did not raise any objections to doing this.

Openly Transmitting Data Backups
A requirement was to backup the system database on a nightly basis to an off-site location. In a properly designed system, the database information would be encrypted before it leaves the trust zone, or sent via a secure or encrypted transmission so malicious users could not intercept the data being sent. Several insiders made sure that this database backup file was sent via insecure channels, meaning that it could have been intercepted by a malicious or unintended party.

Use of Improper Cipher Techniques
Cipher techniques are a way of encrypting information so they may not be deciphered by anyone but the intended parties was a topic covered in the class. While advanced cipher techniques such as AES and RSA continue to provide high levels of security, older techniques such as DES no longer provide an adequate level of protection and can be broken with relative ease. In several groups, the insider was able to convince their team to use DES or another outdated cipher technique. This would have left the encrypted information susceptible to attackers.

Storing Passwords in Source Code
Developers should not store passwords in the source code of an application, even if it has been compiled. This creates several potential problems including the ability of a malicious user to reverse engineer the code to discover the user name and password. Since passwords should be routinely changed, altering them in compiled code is typically more difficult and much more infrequently done. In several instances, insider threats convinced their teams to store the database login information in compiled code since they argued that the information would be secure since it was compiled (an inaccurate statement) and that the passwords did not need to be updated (also inaccurate). This left the applications not only vulnerable to reverse engineering, but in the event a developer left the company, they would know that the database login and password information was unlikely to be changed making it susceptible to their attack. Interestingly, one insider was able to convince their team to make the login information be merely "root" and no password since this would be simpler for development. Giving an application root access with violates several security standards, including the *principle of least privilege* and having no password at all is bad for a variety of reasons.

Providing Too Many Privileges for User or Component
A basic principle of software security is the *principle of least privilege*, or the granting of the minimum number of privileges that an application needs to properly function [16]. Granting more privileges than the application needs creates security problems since individuals or software components could intentionally or unintentionally use these extra privileges for malicious reasons. In a few instances, insider threats observed instances of too many privileges and chose not to disclose the error, allowing it to propagate into future designs.

D. Post Activity Discussion and Goals

After the insiders were identified and the ensuing discussion about what malicious activities could have been performed on the software, a post-activity discussion takes place. The goal of this discussion is to foster student thinking about insiders, ways that insiders could maliciously act, and prevention methods against them. This discussion should be easy going to foster and encourage student led discussions and to encourage free thinking among the students. Due to

the nature of the activity, there are a wide range of potentially beneficial discussion topics. Some of these are outlined below.

Who Can Be Insider Threats?
Before the activity, many students thought of threats as being people external to the organization, and did not consider inside actors to be threats. Students who may have considered the possibility may not have realized that these threats which could be planned for. One potential outcome is a discussion of who can be insider threats. Regardless of the role on the team, each individual or group of individuals is a potential insider threat who needs to be protected against. In our activities, students discussed the various roles of teammates who were insiders and how similar roles in real-world projects could be hazardous.

What Malicious Actions did the Insiders Take?
Insiders conducted a wide variety of malicious actions in our activity using many different methods — many with relative ease. Students are encouraged to discuss the actions that insiders took on their systems and what some of the negative ramifications could have been. They are also encouraged to significantly explore and analyze the negative ramifications of the threats. As an example, in the situation where student records were publicly exposed, consequences which may not be immediately considered are the legal ramifications involved and potential lawsuits by students with publicly exposed grades. A related activity that could be done to augment this discussion would be to to ask students to explore and report upon real-world examples of malicious actions taken by insiders.

What Damage Did the Insiders Cause?
In our activity instances, insider threats would have typically been able to inflict a significant amount of damage on the software project. Understanding the possible negative ramifications is a good way for students to realize the importance of protecting against insider threats and to plan for similar occurrences in their real-world development teams.

Did the Students Realize the Insiders Were Doing Anything Wrong?
In our discussions, students often reported that they had at least a feeling that the insiders were acting in a malicious manner, but failed to stop them. Some reasons included not wanting to create controversy, the feeling that security was not a prominent area of concern, or that their teammate *must have known what they were doing*. Points of discussion could include warning signs of vulnerabilities being placed in a system either for intentional or unintentional reasons, methods of alerting teammates about potential issues in a constructive manner and individual project ownership and empowerment.

How Could Thinking Like an Insider Help to Protect Against Them?
Thinking like an insider is a good way to prevent against them. If developers are always considering different ways that their system can be compromised, they will be more likely to develop their application using proper defensive measures and to detect malicious actions by real inside threats.

What Could Have Been Done to Prevent Insider Threats?
One of the most important discussion topics should be what could have been done to prevent these insider threats (and similar ones) from occurring in future projects. Students are encouraged to discuss ways of preventing these threats from occurring in real-world projects. Some discussed methods include code reviews, internal accountability, maintaining an open culture, increasing auditability, and analysis by outside security auditors.

IV. Student Feedback

Students have expressed a significant amount of satisfaction in this activity and it has contributed to their overall satisfaction with the course. At the conclusion of the project, students are asked to submit an anonymous survey asking them to provide feedback regardingt he project. Some of the questions were based upon the Likert scale, while other asked students to provide written feedback. Several of these questions and student responses are shown in Table I. The survey has been posed to students in the last three course offerings, all of which have used this activity component. A total of 68 students from these sections have chosen to respond.

These results indicate that the vast majority of students not only enjoyed the activity, but would also recommend it to a classmate as well. Additionally, most students felt that it reassembled a project which they were likely to encounter in the real world and were similar to tasks they were asked to complete while on cooperative internships.

The following are samples of written feedback that have been received:

> "The surprise not only teaches the lesson but leaves an impression. This is probably due to the deception aspect of the activity."

> "It was really interesting to see how few people were looking for an insider threat and many threats went completely unnoticed. It showed that we weren't prepared to consider our classmates as threats."

> "I liked how it showed me how easily insider threat can destroy a project.."

> "Being continually consciously aware of all possible security threats, for not all risks lie within the implementation."

> "It might seem silly, but the first-hand experience of having someone betray the team, even on an insignificant level, leaves an impression that the same could actually happen in the real world."

This feedback indicates that students not only enjoyed the activity, but felt that it was an effective learning mechanism as well.

TABLE I: Student Responses

	Strongly Agree	Agree	Undecided	Disagree	Strongly Disagree
You enjoyed the activity	19	37	10	2	0
You learned a lot as a result of the activity	14	39	11	2	0
The activity better prepared you for insider threats	21	39	5	1	2
You feel you are more prepared to deal with an insider threat	16	38	9	5	0
You would be likely to recommend the activity to a friend	16	29	15	7	0

V. Related Work

Numerous works have examined insider threats from a general security perspective. Leyden [10] and Brancik [5] both discussed the importance of protecting against insider threats and the possible negative ramifications. Nostro et al. [12] developed a process for insider threat detection and mitigation using a variety of existing tools and new techniques with the goal being to define the objectives of the attacker and subsequently determining appropriate countermeasures. Other works have discussed various interesting ways of detecting insider threats; Almehmadi et al [2] investigated the use of using physiological features to detect attackers and found that an abnormal deviation in a user's electrocardiogram amplitude could properly predict an attack before it occurred.

To our knowledge, this activity was the first of its kind to introduce an insider threat into a software security classroom activity. However, Krutz and Vallino [9] used a similar activity to teach freshman seminar students about problematic teammates. In this activity, moles were added to teams and performed roles such as non-contributors, side trackers and absentee team members. A post-activity discussion focused on these problematic team members and how how they can be properly addressed. Creating an activity that was not only informative but also enjoyable for the students is an important objective.

This activity fully engages students in the learning process, which is important as research has shown that students learn better when they are actively engaged in the learning process [1]. Various other learning techniques that use "active", "collaborate", "cooperative" learning techniques have been recommended over the years [3], [4], [17], [18].

VI. Future work

This project has been utilized in several sections of our Engineering of Secure Software course and has been very successful, but there are a few enhancements to the activity and further data which may be collected. In future iterations, we would like to further record and analyze the exploits created by the insiders. This would not only be helpful for better planning future offerings of this activity, but would likely be interesting to researchers as well. Once an insider's exploit is exposed during the class discussion, a secondary activity could be for the team to resolve that exploit like they would in a real-world environment. This could include steps taken to mitigate the exploit and measures which could be taken to ensure that a similar exploit did not occur in the future.

Should instructors have more time to complete this activity, a formal code inspection by the team before revealing the insiders could be a beneficial activity, teaching the students about the code inspections themselves and helping to solidify their importance in detecting insider threats. If the team did not identify the threats, this would help to demonstrate the many challenges in detecting an insider threat.

When conducting this activity, we chose to give the insiders a significant amount of freedom in deciding what kind of threats to add to the system since a large amount of variability existed between each team's design and the dynamics of the team. However, some insiders have expressed the desire for more direction on the types of threats they should be looking to create. Future instructors should take this into consideration, but also remember to provide balance: too much direction which would inhibit the creativity and ability of the insiders to create threats.

VII. Conclusion

Organizations suffer from insider threats on a constant basis — posing risks that could impact them not only monetarily, but also through the loss of invaluable and often private data. Some of this harm is irreversible for organizations from both a data and customer trust perspective. Unfortunately, students are typically unprepared to deal with this notion and do not understand that it can occur, and how it may occur.

We have described a novel and innovative activity to instruct software security students about the dangers of insider threats and some of the damaging ramifications such an actor could have on a system. Students have expressed their satisfaction with the activity from both an enjoyment and learning perspective. We encourage instructors at other institutions to use this activity in their security courses as well.

References

[1] *7 Principles for Good Practice in Undergraduate Education*. 1989.

[2] A. Almehmadi and K. El-Khatib. On the possibility of insider threat detection using physiological signal monitoring. In *Proceedings of the 7th International Conference on Security of Information and Networks*, SIN '14, pages 223:223–223:230, New York, NY, USA, 2014. ACM.

[3] T. Bailey and J. Forbes. Just-in-time teaching for cs0. *SIGCSE Bull.*, 37(1):366–370, Feb. 2005.

[4] C. Bonwell and J. Eison. *Active Learning: Creating Excitement in the Classroom*. Wiley, 1991.

[5] K. Brancik. Insider computer fraud an in-depth framework for detecting and defending against insider it attacks., 2008.

[6] R. F. T. Dawn M. Cappelli, Andrew P. Moore. The cert guide to insider threats: How to prevent, detect, and respond to information technology crime. In *The CERT Guide to Insider Threats: How to Prevent, Detect, and Respond to Information Technology Crime*, 2012.

[7] G. Doss and G. Tejay. Developing insider attack detection model: A grounded approach. In *Intelligence and Security Informatics, 2009. ISI '09. IEEE International Conference on*, pages 107–112, June 2009.

[8] J. Guo. Group projects in software engineering education. *J. Comput. Sci. Coll.*, 24(4):196–202, Apr. 2009.

[9] D. Krutz and J. Vallino. Experiencing disruptive behavior in a team using moles. In *Frontiers in Education Conference, 2013 IEEE*, pages 1492–1495, Oct 2013.

[10] . J. Leyden. geeks, squatters and saboteurs threaten corporate security. http://www.theregister.co.uk/2005/12/15/mcafee_internal_security_survey/.

[11] A. Meneely and S. Lucidi. Vulnerability of the day: concrete demonstrations for software engineering undergraduates. pages 1154–1157, 2013.

[12] N. Nostro, A. Ceccarelli, A. Bondavalli, and F. Brancati. Insider threat assessment: A model-based methodology. *SIGOPS Oper. Syst. Rev.*, 48(2):3–12, Dec. 2014.

[13] S. Oliver. Appleinsider. http://appleinsider.com/articles/14/09/25/researcher-accuses-apple-of-ignoring-icloud-brute-force-attack-for-6-months, September 2014.

[14] D. Petkovic, G. Thompson, and R. Todtenhoefer. Teaching practical software engineering and global software engineering: evaluation and comparison. *SIGCSE Bull.*, 38(3):294–298, June 2006.

[15] P. Rubens. Ten ways to protect your network from insider threats. http://www.enterprisenetworkingplanet.com/netsecur/article.php/10952_3882886_2/Ten-Ways-to-Protect-Your-Network-From-Insider-Threats.htm, 2010.

[16] J. H. Saltzer and M. D. Schroeder. The protection of information in computer systems. *Proceedings of the IEEE*, 63(9):1278–1308, 1975.

[17] D. Schweitzer and W. Brown. Interactive visualization for the active learning classroom. In *Proceedings of the 38th SIGCSE Technical Symposium on Computer Science Education*, SIGCSE '07, pages 208–212, New York, NY, USA, 2007. ACM.

[18] T. Sutherland and C. Bonwell. *Using Active Learning in College Classes: A Range of Options for Faculty: New Directions for Teaching and Learning, Number 67*. J-B TL Single Issue Teaching and Learning. Wiley, 1996.