Teaching the Security Mindset With Reference Monitors*

Justin Cappos
NYU Poly
Brooklyn, NY 11201, U.S.A.
jcappos@poly.edu

Richard Weiss
The Evergreen State College
Olympia, WA 98505 U.S.A.
weissr@evergreen.edu

ABSTRACT

One of the central skills in computer security is reasoning about how programs fail. As a result, computer security necessarily involves thinking about the corner cases that arise when software executes. An unfortunate side effect of this is that computer security assignments typically necessitate deep understanding of a topic, such as how the stack is laid out in memory or how web applications interact with databases.

This work presents a series of assignments that require very little background knowledge from students, yet provide them with the ability to reason about failures in programs. In this set of assignments, students implement two very simple programs in a high-level language (Python). Students first implement a reference monitor that tries to uphold a security property within a sandbox. For the second portion, the students are provided each others' reference monitors and then write attack code to try to bypass the reference monitors. By leveraging a Python-based sandbox, student code is isolated cleanly, which simplifies development and grading.

These assignments have been used in about a dozen classes in a range of environments, including a research university, online classes, and a four year liberal arts school. Student and instructor feedback has been overwhelmingly positive. Furthermore, survey results demonstrate that after a 2-3 week module, 76% of the students who did not understand reference monitors and access control learned these key security concepts.

Categories and Subject Descriptors

K.3.2 [**Computer and Information Science Education**]: Computer science education; D.4.6 [**Security and Protection**]: Access controls; K.6.5 [**Security and Protection**]: Unauthorized access (e.g., hacking, phreaking)

*This work was partially supported by the NWDCSD and NSF grants 1141341, 0834243, 1223588, 1205415, and 1241568.

SIGCSE'14, March 5–8, 2014, Atlanta, Georgia, USA.

http://dx.doi.org/10.1145/2538862.2538939

General Terms

Computer Security Education

Keywords

Security, Python, Access Control, Reference Monitor

1. INTRODUCTION

One of the central concepts in security is to understand how systems can fail, and be made to fail, in different ways. The ability to think about failures and how to trigger them is often termed having a security mindset. It also extends to questioning assumptions and think analytically about their implications. This could be something simple such as, "What can I assume about this web page that is asking for my password?" to "What is the set of invalid strings that an attacker could send to this application that would make it fail?" The analysis of the implications often leads to an exploration of the subtleties of failure cases and requires detailed knowledge of the underlying concept. For example, to understand buffer overflows, a student must understand memory layout and how the stack functions. To understand DNS cache poisoning, one must understand how name resolution functions on the Internet. To understand SQL injection and XSS, one must understand the basics of web applications and databases. To understand these specific applications of the security mindset, one needs some advanced domain knowledge. For this reason, security is typically only emphasized very late in the computer science curriculum which causes some students to miss it. However, the security mindset can be understood independently of the application, and it should be possible to give experiential training in lower division classes.

In this work, we describe a set of programming assignments that require very little background, but provide students with an experience in understanding failure modes (i.e. a security mindset) in the context of a reference monitor. A reference monitor is a method or function that implements an access control policy for a set of resources and is usually specified in terms of what capabilities are allowed. Our goal is to effectively teach this topic in a way that requires minimal background for the students.

In the first part of the assignment, students construct a defensive program that functions as a reference monitor and attempts to meet a specific security goal. This gives the students the experience of thinking as a defender and reinforces the concepts of how to build a secure system. The second part of the assignment involves the students attack-

ing each others' defensive programs created in the first part of the assignment. This allows students to see security from the attackers standpoint and think about how to trigger failures. The third part of the assignment provides the students with a working attack that bypassed their defense written by another student. The students are then asked to categorize the faults that led to the failure in their defensive program and to provide a patch that fixes these problems.

These assignments have been used in about a dozen classes in a range of environments, including both in class and online classes at NYU Poly as well as classes at The Evergreen State College, a four-year liberal arts school. The student feedback has been overwhelmingly positive, with students repeatedly remarking on the effectiveness of the assignments during course evaluations. Furthermore, a pre- and post-assessment of the learning objectives from this portion of the class shows significant improvement in student understanding. In the case of one content question that was directly about the exercise, and required some higher-level understanding, 76% of the students who got the question wrong on the pre-survey were able to answer it correctly on the post-survey.

2. BACKGROUND

The assignment described in this work uses the sandbox used in the Seattle testbed [2, 24, 12, 19]. Seattle's sandbox consists of a programming language virtual machine where students write code in a subset of Python and it executes locally on their computer. Seattle's sandbox is write-once, run anywhere, so the assignment works for students that have Linux, Mac, Windows, BSD, or other operating systems.

In addition to its use in this context, Seattle is a peer-to-peer platform upon which students and researchers can run their own code on end-user devices distributed across the world [2]. One of the main goals of Seattle was to be able to observe real Internet behavior by running software on the everyday devices that are used by the community. However, the security assignments described in this work do not utilize the distributed nature of Seattle.

Seattle has seen broad educational use in 45 classes (primarily on the topics of networking, distributed systems, and security) at over a dozen universities [24, 2]. Students have repeatedly lauded Seattle as easy to use and easy to learn. There exists a wide array of instructional modules that leverage Seattle which are available through different sources [7, 14]. As of August 2013, Seattle is the top rated educational resource on the ACM SIGCOMM Educational Resources Site [1]. Assignments using Seattle are being included into the next edition of the most widely used networking textbook [10].

Interestingly, the Seattle sandbox provides a good environment for the students to explore security because it isolates what both the defensive and attack programs can do. This is achieved with Seattle's multilayer security design, as is shown in Figure 1. User programs must access kernel operations through reference monitors that are isolated using the security layers technique [3]. Seattle itself has been designed to handle the insertion of security functionality and ensures that even if a student writes effective attack code, they can do no harm to other students' systems or the host system [3]. The student's security module sits on top of the security monitors that are built into Seattle.

A reference monitor enforces an access control policy on untrusted code. Seattle provides a mechanism for the user to create new reference monitors that sit between untrusted application code and methods in system libraries and the kernel that are to be protected. As an example of how this might be used, some files might have permissions set so that the operating system would allow all users to modify them arbitrarily. A reference monitor can programmatically decide what to do in response to file operations, and so can can do fine-grained operations such as block file writes that would write known malware signatures.

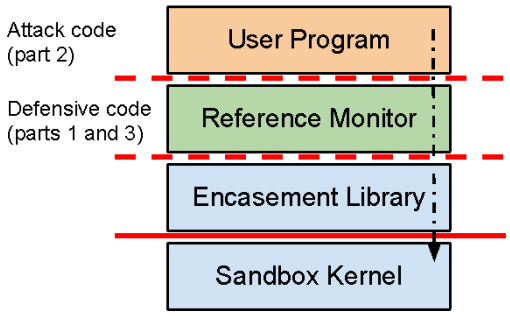

Figure 1: A reference monitor intercepts requests provided by the user program and decides how to process them. The encasement library provides memory isolation between the reference monitor and the user program. There is a clearly defined boundary between the reference monitor and user program, which are implemented in different files. The vertical arrow indicates the only possible call-path from user code passes through the reference monitor before going into the sandbox TCB.

3. THE ASSIGNMENT

The access control reference monitor exercise provides a good opportunity for students to get a feeling of what it means to have a security mindset and to own one's code. The assignment itself is general and can be customized when used, perhaps to prevent cheating or allow the assignment to be reused in multiple classes. For concreteness this section describes one instance of the assignment (the "MZ" version) before discussing other variations. A complete module writeup for each part with detailed instructions (and instructor-only solutions) is available online [17, 15]. There are three main parts to the exercise.

Part One. In the first part, students write a defensive program that enforces an access control policy [17]. For example, the first time the assignment was given, students implemented a reference monitor to prevent a malicious application from writing the characters "MZ" to the first two bytes of a file. (Historically, this is an indicator for an old executable format.) The exercise is to do this by adding security rules to the functions available for reading from a file and writing to a file. The exercise tests the student's ability to write secure code. The students are assessed on their successful use of three design paradigms: accuracy, efficiency, and security. An example of what the vulnerable code that

```
def writeat(self,data,offset):
if data.startswith("MZ") and (offset == 0):
     raise ValueError("Cannot start file with MZ!")
else:
     return self.file.writeat(data,offset)
```

Figure 2: An excerpt from the example (inadequate) defensive code students are provided.

attempts to enforce this looks like is shown in Figure 2.

- Accuracy: The reference monitor should only stop certain actions from being blocked. All other actions should be allowed. For example, If a user tries to write "MX" as the first two bytes of a file, this should be allowed. As a second example "ZM" should be allowed as well. In fact, "MZ" can be written in the file as well, so long as this is not at the first two characters. Accuracy ensures all valid operations are permitted.
- Efficiency: The reference monitor should use a minimum number of resources, so performance is not impacted. This means the defensive code must not do things like re-read the file before each write. Efficiency ensures the reference monitor is not overly costly in terms of unnecessary I/O.
- Security: The attacker should not be able to circumvent the reference monitor. Hence if the attacker can first write "Z" as the second character and later writes "M" as the first character, the action must be blocked. Security ensures the reference monitor does not allow an attacker to circumvent the goal of the reference monitor.

Part Two. For the second part of the exercise [15], each student's code from the first part is made available to every other student in the class. A student will create a series of one or more test cases that attempt to cause security problems or impact the accuracy of the system. To find these flaws, we have observed students employing a variety of techniques. Essentially all students look at each others code to find ways to compromise the reference monitor. This provides students with some experience in bug-finding through reading code. Many students also chose to write routines to fuzz input to the reference monitors to further uncover problems that they could exploit. A few students also wrote tests that attempt to trigger time-of-check-to-time-of-use (TOCTTOU) bugs by causing race conditions in the code.

Part Three. In the third part of the assignment (which has not been used at all of the schools), the students are given the attack code that bypassed their reference monitor. They are asked to fix the reference monitor so that the security it provides cannot be bypassed in the same way. The student also must write a document that classifies the bugs in their code according to type (such as TOCTTOU, error handling logic, or similar issues). This helps to reinforce to the student why these issues occurred and what they might do to avoid them in the future.

Types of Errors Students Make. Writing effective defensive code for this assignment is a deceptively hard task. In prior classes, about 80% of the students wrote a reference monitor that other students could find a flaw in. (In fact, most instructors who have attempted the assignment also have errors in their reference monitor.) While in a few cases, students simply do not adequately test their code and have obvious programming errors, this is not the common cause of issues. Students tend to write insecure code due to overlooking a few common issues (listed from most to least commonly exploited).

- The reference monitor needs to track the state of the information on disk, but cannot re-read it for every access (due to efficiency concerns). A common mistake is when the attacker can cause the reference monitor's state to diverge from the underlying system's state, especially in error conditions. For example, if the program attempts to write past the end of a file (or before the beginning of a file), the reference monitor may incorrectly update its state. As a result, the reference monitor will improperly block or allow writes since the disk contents differ. This can cause both security and accuracy issues.
- Time-of-check-to-time-of-use (TOCTTOU) bugs and other types of race conditions are a fairly common oversight for students. Some students write test cases that attempt to trigger a race condition to exploit these problems. This can result in essentially any sort of attack, even infinite loops in the reference monitor in some cases.
- Some reference monitors inappropriately share state for different files. For example, there may be a global set of state that is used to store the first two bytes. By opening multiple files, an attacker may be able to overwrite this state and cause security and accuracy issues.
- In rare cases, a student's reference monitor may inappropriately retain state for a file. For example, an attacker may create a file, write some data, then close and delete the file. If the attacker recreates a file with the same name, the state should be cleared.

One of the important features of this exercise is that students can get experience in both offensive and defensive roles. When defending a system, the more attacks one knows, the better. Getting students to own their code is a significant challenge. In order for students to write secure code, they must be able to see both sides. One unexpected benefit of the assignment is that students seemed to enjoy seeing each others' solutions to assignments. Many students said that being able to learn from (and exploit) these pieces of code was much more rewarding than submitting exercises and having them graded by the instructor or TA. From the instructor's standpoint, it has the additional benefit that student feedback can be used in grading. The defensive parts of the assignments can be graded based upon how well they hold up to the offensive portion. (We provide scripts that automate the grading process.) Simply by validating each student's attacks against the other students' defenses, makes an easy way to provide meaningful feedback and scoring.

3.1 Variations

The assignment can also be easily customized or modified to allow reuse. The ease of doing this is an important feature because most faculty have limited time to develop

ASSIGNMENT:

```
In this assignment you will create a reference moni-
tor which stops the attacker from reading data securely
written in a file.  You will create a function called
privatewrite which allows the user to write data into a
file which cannot be read back from the file.  The data
can be overwritten but isn't readable under any circum-
stance.  You will do this by adding security rules to
the functions available for reading from and writing to
a file.  Think of this as a method to write some data
securely into a file.  The future of the system depends
on your ability to write secure code!
```

Figure 3: An excerpt from the assignment description for the privatewrite variation.

new curriculum. One such variation that we have tried is the Private Write assignment. Instead of simply interposing on an existing file system API call, as the 'MZ' assignment does, the Private Write assignment adds a new call as is specified in Figure 3. The complete module writeup with detailed instructions for both the defense and attack code are also available [18, 16].

As with the original assignment, we provided the students an example reference monitor so they could see how to integrate and run their code. By examining the attack and defense scores, this variation was of about the same difficulty as the original assignment. This demonstrates that it is easy to adapt the assignment to the preferences of the individual instructor. As a result, the assignments can be reused or customized to better fit different portions of the curriculum.

4. METHODOLOGY FOR ASSESSMENT

This group of reference monitor exercises has been used in about a dozen classes at multiple schools (NYU Poly and The Evergreen State College). The assignment is very straightforward to modify in minor ways to fit the class, and this has been done by several instructors who used it. The ease of modifying the exercise makes it easy for faculty to adapt it to their classes and resist students posting and copying answers.

When this assignment has been given in regular classes, the students are usually given one week to complete each part. In order to make the assignments easier to complete, especially for students with no experience with Python, we gave the students a template consisting of a simple reference monitor which inadequately checks the desired security property. For example, this may check to see that the first two characters in a single call to write are not "MZ". A snippet of this code is shown in Figure 2. Students are also given an example of an attack program that tries to write "MZ" in the first two characters. This provides students with a starting point and demonstrates to them how to run the program and see how an ineffective attack can be blocked.

Description of the courses we taught and how the exercises were used: One of the faculty used this exercise in an upper-division class on network security at The Evergreen State College, a liberal arts college. Many of the students did not know Python, and they were still able to do the exercise. However, they made some mistakes due to the fact that they were not thinking about concurrency and thread safety which are covered in other classes such as operating systems.

	Q1	Q2	Q3	Q4
Pre-survey: number of correct answers out of 57	29 (51%)	27 (47%)	21 (37%)	42 (74%)
Post-survey: number of correct answers out of 38	32 (84%)	23 (61%)	20 (53%)	29 (76%)
For paired responses: number of incorrect answers on pre-survey	21	17	21	10
For paired responses: number of correct answers on post-survey, given incorrect answer on the pre-survey	16	11	12	5

Table 1: Survey results from classroom use of the exercise.

This exercise was also given to a faculty attending the CSAW Summer Research and Training for College Faculty, a summer workshop on security [4]. These faculty were not necessarily experts in security and they were only given one evening to work on the assignment. Several of the faculty did not know Python before starting the assignment. All of the faculty completed both the defensive and offensive assignments. While there were a few whose programs passed all of the tests, the majority made mistakes. We think that this shows that the exercise requires analysis skills and questioning assumptions (security mindset), and not just knowledge.

We were interested to know whether students fully engaged in the exercises and learned the main concept. We asked students a brief set of knowledge questions (listed in Figure 4) before and after they were assigned the reference monitor module. These questions were intended to obtain a baseline of student understanding with respect to the concepts that lie at the core of the module as well as related concepts. Our survey asked questions on how a reference monitor is used, how it relates to access control matrices and how it relates to capability systems. It also asked a general question about access control. The first question was a core one, and that showed a direct relationship to the exercise. The others potentially indicated higher cognitive levels at which the student could make connections to related concepts at the level of 'synthesis' in Bloom's taxonomy. With laboratory exercises it is often possible for students to go through the motions of doing the exercise without reflecting deeply on it or even understanding what they are doing. The four questions we asked were designed to measure understanding at a range of cognitive levels on a relative scale. The first question was a knowledge question about how a reference monitor is used. The other three questions were about access control and required different levels of understanding. The levels of difficulty can be inferred from the larger sample of students who took the post-survey and it agreed with the *a priori* ranking of difficulty and cognitive level. The four content questions were multiple choice, each with four possible answers except the first, which had three.

5. RESULTS

We analyzed the survey results from the introductory security class at NYU Poly in the Fall of 2012. Our results demonstrate that a significant number of students learned the main concept of reference monitor and how it relates to other concepts in access control. All of the students in the

Q1. Suppose that an application is running in an environment with a reference monitor. The application gains access to the operating system:

- Directly
- Through the reference monitor
- Either through the reference monitor or not at the application programmer's discretion

Q2. Which of the following is easy to do with an access matrix system:

- Trust revocation of "open" resources
- Listing all users with access to a resource
- Authentication of users
- Building a reference monitor

Q3. Which of the following is easy to do in a capability system:

- Trust revocation of "open" resources
- Listing all users with access to a resource
- Authentication of users
- Building a reference monitor

Q4. Which of the following is crucial to every access control system:

- Correct authentication of users
- Confidentiality
- Strong passwords
- Long cryptographic keys

Figure 4: Survey questions to measure learning in the exercise.

class turned all three portions of the assignment. The average number of lines of code for each part of the assignment was under 100. Although there were about 70 students in the class, the number students who completed both pre- and post-surveys using matching names was 34. The results are summarized in Table 1.

The most significant result from the surveys was for the first question (Q1) which was directly about the exercise. For this question, of the 34 students with paired surveys, 21 students answered the question incorrectly on the pre-survey. Of the 21, the number who answered it correctly on the post-survey was 16. This demonstrates a significant learning outcome. If none of those 21 students had learned the concept and had guessed on the post-survey, we would expect only 7 of them to have gotten the right answer. The probability that 16 or more would have guessed the right answer is less than 0.01. This result suggests that 76% of the students who did not know the concept before, were able to learn it by completing the exercise. Of the 13 who answered it correctly on the pre-survey, one got it wrong on the post-survey, suggesting that only one or two students in that group were guessing. Note that the answer to the question on the survey did not appear in the description of the exercise. As we know from experience, it is possible for students to do an exercise mechanically without grasping the meaning, and that does not seem to be the case here.

Question 4 (Q4) was a general question about access control and was not addressed by the assignment. Of the 10 students who got it wrong on the pre-survey, 5 got it right on the post-survey, which allows us to reject the null hypothesis ($p = .035$) that no one learned the concept in the the time frame of the assignments. Our explanation is that in the process of attending lectures and reading for the assignment, students also learned related material not directly covered. This is important to track because a good assignment will engage students in the subject and prime them for learning related information. Questions 2 and 3 were related to the assignment, and in this context required a higher level of cognitive understanding than question 1 because they were not specifically addressed in the assignments. For question 2, of the 17 students who got it wrong on the pre-survey, 11 got it correct on the post-, giving statistical significant positive outcome with a $p = .04$ and an estimated learning rate of 65%.

Question 3 is problematic for a couple of reasons. First, depending on the way that the capability system is implemeneted, there can be more than one correct answer. As a result, we consider two answers as being potentially correct. This greatly reduces the statistical guarantees that students are not guessing and that the improvements in scores are not due to chance. Thus, although the percentage of students who got it right increased whether it was scored with one or two correct answers, the results were not statistically significant in either case.

6. RELATED WORK

Security Injections [20] presents a compelling set of exercises for CS0, CS1 and CS2 that address the problem of adding security-related exercises to an existing curriculum at an introductory to intermediate level. However, their examples (such as buffer overflows and integer overflow) do not give students the opportunity to engage in both defensive and offensive roles. They focus on safe coding practices without giving them an opportunity to think actively about how to attack vulnerable code. The SEED project [6] presents a set of more advanced exercises and projects, but they also do not involve both defensive and offensive roles.

There are a number of hands-on exercises in the form of games. The competitive exercise firesim [23] does give students an opportunity to assume both roles in a competitive exercise about firewalls, although the attack role is limited to a small set of options. CyberSIEGE is an interactive video game with a number of high-level scenarios that emphasize defense and risk management [21]. Players decide how to allocate resources to defend a network. It includes scenarios about component configuration (e.g., ACLs; filters; VPNs; etc.), policies/training, patch management, network topology, use of cryptography, use of background checks, and physical security. In spirit it seems to address the security mindset, although it does not model vulnerability analysis at a high level of detail and does not put players in the role of attacker. EDURange [8] is an (as of yet unreleased) environment for developing competitive games with both offensive and defensive roles. However, EDURange requires deeper technical knowledge of networking, port scanning, and similar tasks than the assignments described here. The Control-Alt-Hack [5] card game teaches security awareness, but does not involve analysis or the security mindset.

There are at least two platforms that provide an environment for running cybersecurity exercises. They are DETER [11] and RAVE [13]. They provide a virtual private network which is flexible isolated from the Internet, so they can be used for offensive and defensive exercises. These are more suitable for advanced tasks like network exploitation

and penetration testing, while our assignments require much less background and expertise.

The Google Gruyere [9] and OWASP Webgoat [22] projects provide students with an opportunity to play the role of attacker of vulnerable web applications. Gruyere also includes tutorial material which helps students with a wide range of backgrounds. Neither one provides an opportunity to harden an application against attacks.

7. CONCLUSIONS AND FUTURE WORK

In order to be meaningful, practical, and engaging for students, we believe that the computer security curriculum must include hands-on exercises that develop both defensive and offensive skills. This is integrally related to the security mindset, which is about not trusting input and interfaces, examining failure modes and questioning assumptions. This topic is particularly important in the knowledge domain of access control, which is one of the fundamental concepts in computer security and arises with firewalls, file protections and information processing in general. We believe that the assignments we used challenged students to reassess failures and think through assumptions in this key domain.

We applied pre- and post-surveys to assess whether students learned the fundamental concept of a reference monitor in the context of access control. The results showed that there was a significant improvement at the knowledge level and at higher levels, as well. While we did not show causation, we did show correlation over a short period of time. We plan to collect additional survey data from other classes.

In the future, we would like to test more directly whether students can learn the security mindset from doing exercises that require them to take on both attacker and defender roles. We believe that this can be done with exercises similar to the one that we described in this paper. Although we do not have any data for CS1 or CS2, we believe that there is an opportunity to teach and assess the security mindset and the concepts of access control using the defender and attacker roles even in lower division classes. We intend to explore these assignments in lower division classes to determine their effectiveness early in the curriculum.

8. REFERENCES

[1] ACM SIGCOMM Educational Resources. `http://edusigcomm.info.ucl.ac.be/`.

[2] J. Cappos, I. Beschastnikh, A. Krishnamurthy, and T. Anderson. Seattle: a platform for educational cloud computing. *SIGCSE Bull.*, 41(1):111–115, 2009.

[3] J. Cappos, A. Dadgar, J. Rasley, J. Samuel, I. Beschastnikh, C. Barsan, A. Krishnamurthy, and T. Anderson. Retaining Sandbox Containment Despite Bugs in Privileged Memory-Safe Code. In *The 17th ACM Conference on Computer and Communications Security (CCS '10)*. ACM, 2010.

[4] Csaw summer research and training for college faculty. `http://www.poly.edu/events/2013/07/08/csaw-summer-research-and-training-college-faculty`.

[5] T. Denning, T. Kohno, and A. Shostack. Control-alt-hack: a card game for computer security outreach and education (abstract only). In *Proceeding of the 44th ACM technical symposium on Computer science education*, SIGCSE '13, pages 729–729, New York, NY, USA, 2013. ACM.

[6] W. Du and R. Wang. Seed: A suite of instructional laboratories for computer security education. *Journal on Educational Resources in Computing (JERIC)*, 8(1):3, 2008.

[7] Educational Portal –Seattle – Trac. `https://seattle.poly.edu/wiki/EducationalPortal`.

[8] EDURange Description. `http://blogs.evergreen.edu/edurange`.

[9] Google Gruyere Home. `http://google-gruyere.appspot.com/`.

[10] J. F. Kurose and K. W. Ross. *Computer Networking: A Top-Down Approach Featuring the Internet*. Pearson/Addison-Wesley, 2005.

[11] J. Mirkovic, T. Benzel, T. Faber, R. Braden, J. Wroclawski, and S. Schwab. The deter project: Advancing the science of cyber security experimentation and test. In *Technologies for Homeland Security (HST), 2010 IEEE International Conference on*, pages 1–7, 2010.

[12] Monzur Muhammad and Justin Cappos. Towards a Representive Testbed: Harnessing Volunteers for Networks Research. In *The First GENI Research and Educational Workshop*, GENI'12, 2012.

[13] K. Nance, B. Taylor, R. Dodge, and B. Hay. Creating shareable security modules. In *Information Assurance and Security Education and Training*, pages 156–163. Springer, 2013.

[14] Main Page – NW-DCSD. `http://ai.vancouver.wsu.edu/~nwdcsd/wiki/index.php/Main_Page`.

[15] Attacking a reference monitor that implements access control. `https://seattle.poly.edu/wiki/EducationalAssignments/SecurityLayerPartTwo`.

[16] Attacking private write. `https://seattle.poly.edu/wiki/EducationalAssignments/PrivateWritePartOne`.

[17] Building a reference monitor that implements access control. `https://seattle.poly.edu/wiki/EducationalAssignments/SecurityLayerPartOne`.

[18] Building a reference monitor that implements private write. `https://seattle.poly.edu/wiki/EducationalAssignments/PrivateWritePartOne`.

[19] Seattle. `https://seattle.poly.edu/`.

[20] B. Taylor and S. Kaza. Security injections: modules to help students remember, understand, and apply secure coding techniques. In *Proceedings of the 16th annual joint conference on Innovation and technology in computer science education*, pages 3–7. ACM, 2011.

[21] M. Thompson and C. Irvine. Active learning with the cyberciege video game. In *Proceedings of the 4th conference on Cyber security experimentation and test*, pages 10–10. USENIX Association, 2011.

[22] OWASP Webgoat Project. `http://www.owasp.org/index.php/Category:OWASP_WebGoat_Project`.

[23] K. Williams. Firesim Project. `http://williams.comp.ncat.edu/firesim`.

[24] Yanyan Zhuang and Albert Rafetseder and Justin Cappos. Experience with Seattle: A Community Platform for Research and Education. In *The Second GENI Research and Educational Workshop*, GREE'13, 2013.