Gamifying Software Security Education and Training via Secure Coding Duels in Code Hunt

Tao Xie
University of Illinois at Urbana-Champaign
Urbana, IL, USA
taoxie@illinois.edu

Judith Bishop
Microsoft Research
Redmond, WA, USA
jbishop@microsoft.com

Nikolai Tillmann, Jonathan de Halleux
Microsoft Research
Redmond, WA, USA
nikolait@microsoft.com
jhalleux@microsoft.com

ABSTRACT

Sophistication and flexibility of software development make it easy to leave security vulnerabilities in software applications for attackers. It is critical to educate and train software engineers to avoid introducing vulnerabilities in software applications in the first place such as adopting secure coding mechanisms and conducting security testing. A number of websites provide training grounds to train people's hacking skills, which are highly related to security testing skills, and train people's secure coding skills. However, there exists no interactive gaming platform for instilling gaming aspects into the education and training of secure coding. To address this issue, we propose to construct secure coding duels in Code Hunt, a high-impact serious gaming platform released by Microsoft Research. In Code Hunt, a coding duel consists of two code segments: a secret code segment and a player-visible code segment. To solve a coding duel, a player iteratively modifies the player-visible code segment to match the functional behaviors of the secret code segment. During the duel-solving process, the player is given clues as a set of automatically generated test cases to characterize sample functional behaviors of the secret code segment. The game aspect in Code Hunt is to recognize a pattern from the test cases, and to re-engineer the player-visible code segment to exhibit the expected behaviors. Secure coding duels proposed in this work are coding duels that are carefully designed to train players' secure coding skills, such as sufficient input validation and access control.

1. INTRODUCTION

Sophistication and flexibility of software development make it easy to leave security vulnerabilities in software applications for attackers. Approaches such as anti-virus applications and network firewalls offer comparatively secure protection at the host and network levels, but not at the software application level. It is critical to educate and train software engineers to avoid introducing vulnerabilities in software applications in the first place such as adopting secure coding mechanisms and conducting security testing.

In the "hacker" community, a number of websites (such as HackThisSite [3] and Hacker Test [2]) provide training grounds to train people's hacking skills, which are highly related to security testing skills. To train people's secure coding skills, Gera's website on Insecure Programming by example [1] offers a list of vulnerable code examples with increasing difficulty. However, there exists no interactive gaming platform for instilling gaming aspects into the education and training of secure coding.

HotSoS '15, Apr 21-22, 2015, Urbana, IL, USA
ACM 978-1-4503-3376-4/15/04.
http://dx.doi.org/10.1145/2746194.2746220.

Figure 1: User interface of game playing in Code Hunt

To address this issue, we propose to construct secure coding duels in Code Hunt (`https://www.codehunt.com/`) [4], a high-impact serious gaming platform released by Microsoft Research. Since its release in early 2014, Code Hunt has attracted over 100,000 users and accumulated their contributed solutions when playing coding duel games. In April 2014, Code Hunt was used at a very large competition called Beauty of Programming in the Greater China Region. In three rounds, 2,353 students scored in the game, with an average 55.7% puzzles solved across this large number. Code Hunt is being offered for more competitions, as ongoing efforts. The Code Hunt Challenge being run as part of Microsoft Imagine Cup (`https://www.imaginecup.com/codehunt`) has hundreds of participants per month from all around the world. Code Hunt's predecessor Pex4Fun [10, 11] also gained popularity in the community: since it was released to the public in June 2010, the number of clicks of the "Ask Pex!" button (indicating the attempts made by users to solve games at Pex4Fun) has reached over 1.6 millions as of January 2015.

2. CODE HUNT

Coding duel [10] is the main game type in Code Hunt. Written in either C# or Java, a coding duel includes a secret code segment and a player-visible code segment, both embodied in a `Puzzle` method sharing the same method signature. The player-visible code segment is typically empty in content or an incorrect/incomplete ver-

sion of the secret code segment. To solve a coding duel, a player iteratively observes clues given by Code Hunt and modifies the player-visible code segment to match the functional behaviors of the secret code segment.

Figure 1 shows the user interface of game playing in Code Hunt. The player-visible code segment in the coding duel is displayed on the left hand side of the user interface. After the player clicks the button "Capture Code" (shown in the top-middle part), Code Hunt relies on the test cases automatically generated by a state-of-the-art test generation tool (Pex [8, 9]) to characterize and display sample common and different functional behaviors (shown on the right hand side) between the secret code segment and the player-visible code segment being modified by the player. The game aspect in Code Hunt is to recognize a pattern from the test cases, and to re-engineer the player-visible code segment to exhibit the expected behaviors.

3. SECURE CODING DUELS

To train software engineers' secure coding skills, we construct secure coding duels on Code Hunt. Our initial efforts focus on the topics of input validation and access control along with integer overflow, with initial artifacts from our previous work [5, 7], and those artifacts from public repositories for security, e.g., the NIST National Vulnerability Database (NVD) (`http://nvd.nist.gov/`).

Input validation duels. User-input validators are the first barricade that protects a software application such as a web application from application-level attacks such as buffer overflow, code-injection attack, hidden-field manipulation, and cross-site scripting. In our previous work [7], we conducted multiple-implementation testing based on Pex [8, 9] on 53 different input validators (of 6 common types) for web applications. Our results show that MiTV detected real defects in 70% of the input validators. Such results demonstrate the strong need of educating software engineers on improving their skills for writing correct input validators.

To construct secure coding duels for training input validation, we go through two steps: (1) collect a faulty input validator IV_f (e.g., from the ones collected in our previous work [7]) and then fix it to produce a correct input validator IV_c; (2) construct a coding duel by turning IV_f into the player-visible code segment and turning IV_c into the secret code segment. Basically, the resulting secure coding duel is to ask the players to half-guess the expected behaviors of the given input validator, and then fix the given input validator to match the expected behaviors. By reading the input validator code, the players can easily infer the input validator type (e.g., credit card number, email address, phone number, SSN, URL, and zip code) even if the type is not explicitly stated in the player-visible code segment. However, the players may not immediately realize the exact expected behaviors of the given input validator for that type.

Access control duels. Access control is one of the most fundamental and widely used privacy and security mechanisms, especially in systems where people share information about themselves in dynamic and mobile situations. Access control mechanisms control which principals such as users or processes have access to which resources in a system. To facilitate managing and maintaining access control, access control policies are increasingly written in specification languages. The specification of access control policies is often a challenging problem. In our previous work [12], we view an access control policy as a software module, which returns permit or deny when given an access request from a principal. Then for each policy language, we have developed a translator that automatically translates a policy written in that policy language to a software module (e.g., a Java or C# method). We can then directly reuse existing software testing tools such as Pex [8, 9] to test the policy indirectly.

To construct secure coding duels for training access control, we go through three steps: (1) collect an access control policy ACP_c (e.g., from the ones collected in our previous work [5]) and then apply a mutation/fault-seeding tool (e.g., one from our previous work [6]) to seed a simple fault into the original policy to produce a faulty policy ACP_f; (2) apply the policy-to-code translator [12] on ACP_c and ACP_f to produce their code versions $ACPC_c$ and $ACPC_f$, respectively; (3) construct a coding duel by turning $ACPC_f$ into the player-visible code segment and turning $ACPC_c$ into the secret code segment. Basically, the resulting secure coding duel is to ask the players to guess the expected behaviors of the given policy, and then fix the given policy to match the expected behaviors.

Acknowledgments

This material is based upon Tao Xie's work supported by the Maryland Procurement Office under Contract No. H98230-14-C-0141, as well as a Microsoft Research Award, NSF grants CNS-1434582, CNS-1439481, CCF-1349666, CCF-1409423, CCF-1434590, and CCF-1434596.

4. REFERENCES

[1] Gera's Insecure Programming by example. `http://community.coresecurity.com/~gera/InsecureProgramming/`.

[2] Hacker Test. `http://www.hackertest.net/`.

[3] HackThisSite Missions. `http://www.hackthissite.org/missions/`.

[4] J. Bishop, N. Horspool, T. Xie, N. Tillmann, and J. de Halleux. Code Hunt: Experience with coding contests at scale. In *Proc. ICSE, JSEET*, 2015.

[5] J. Hwang, T. Xie, V. Hu, and M. Altunay. ACPT: A tool for modeling and verifying access control policies. In *Proc. POLICY, System Demo*, pages 40–43, 2010.

[6] E. Martin and T. Xie. A fault model and mutation testing of access control policies. In *Proc. WWW*, pages 667–676, 2007.

[7] K. Taneja, N. Li, M. Marri, T. Xie, and N. Tillmann. MiTV: Multiple-implementation testing of user-input validators for web applications. In *Proc. ASE*, pages 131–134, 2010.

[8] N. Tillmann and J. de Halleux. Pex – white box test generation for .NET. In *Proc. TAP*, pages 134–153, 2008.

[9] N. Tillmann, J. de Halleux, and T. Xie. Transferring an automated test generation tool to practice: From Pex to Fakes and Code Digger. In *Proc. ASE, Experience Papers*, pages 385–396, 2014.

[10] N. Tillmann, J. de Halleux, T. Xie, S. Gulwani, and J. Bishop. Teaching and learning programming and software engineering via interactive gaming. In *Proc. ICSE SEE*, pages 1117–1126, 2013.

[11] T. Xie, N. Tillmann, J. de Halleux, and J. Bishop. Educational software engineering: Where software engineering, education, and gaming meet. In *Computer Games and Software Engineering*. Taylor & Francis Group, 2015.

[12] T. Yu, D. Sivasubramanian, and T. Xie. Security policy testing via automated program code generation (extended abstract). In *Proc. CSIIRW*, 2009.