Extending HARM to make Test Cases for Penetration Testing

Aparna Vegendla(✉), Thea Marie Søgaard, and Guttorm Sindre(✉)

Department of Computer and Information Science,
Norwegian University of Science and Technology (NTNU), Trondheim, Norway
{aparnav, guttors}@idi.ntnu.no

Abstract. **[Context and motivation]** Penetration testing is one key technique for discovering vulnerabilities, so that software can be made more secure. **[Question/problem]** Alignment between modeling techniques used earlier in a project and the development of penetration tests could enable a more systematic approach to such testing, and in some cases also enable creativity. **[Principal ideas/results]** This paper proposes an extension of HARM (Hacker Attack Representation Method) to achieve a systematic approach to penetration test development. **[Contributions]** The paper gives an outline of the approach, illustrated by an e-exam case study.

Keywords: Security · Penetration testing · Misuse cases · Socio-technical systems · e-exams

1 Introduction

The alignment of requirements and testing has been emphasized as an important problem in software development in general [1, 2] and also for security requirements in particular [3], where testing might then be a combination of penetration testing [4] and ethical hacking [5].

Penetration testing is often used for finding security vulnerabilities in software [6]. As observed by [4], it can be effective if combined with security-related findings from earlier lifecycle stages, but less effective if done completely ad hoc. Even with a systematic approach it is important to be aware that there may be other vulnerabilities remaining in addition to those the tests have uncovered [4].

Previously, our research group has been involved in the development of a method called HARM [7], with the purpose of representing hacker attacks in various ways. In the current paper, we explore how this method could be extended to provide a bridge between security requirements and testing. More precisely, our research question is **RQ1:** *How can HARM be extended to support the development of penetration test cases from security requirements?*

The rest of the paper is structured as follows: Sect. 2 provides background on HARM, illustrating the method with a running example related to the case study, as well as discussing related work. Section 3 discusses how HARM can be extended to include manual human attacks in addition to technical attacks, and to support the development of test cases. Section 4 then presents a case study where HARM is used to

J. Krogstie et al. (Eds.): CAiSE 2016, LNBIP 249, pp. 254–265, 2016.
DOI: 10.1007/978-3-319-39564-7_24

capture security requirements, analyze threats and suggest security test cases for a digital exam system. Section 5 concludes the paper and outlines some ideas for further work.

2 Background

2.1 Running Example: BYOD e-exams

Many universities are currently switching from traditional school exams using pen and paper to e-exams, in some cases performed at home (e.g., remote exams), in some cases in a controlled campus environment. For scalability and cost reduction, even the latter type will often require students to use their own laptops (BYOD, Bring Your Own Device), although this gives increased challenges with security [8]. Concentrating here on individual school exams with invigilators, it is typically necessary to ensure the rules/requirements related to cheating security as shown in Table 1.

Table 1. Some rules against cheating during controlled school exams

	Rule/requirement
R1	Only authenticated examinees shall be able to access and respond to exam questions
R2	It shall be possible to respond to exam questions only while seated at one's assigned place in a controlled venue
R3	Examinees are prohibited from communicating with each other or with outsiders during the exam
R4	Examinees are prohibited from using tools or resources other than those listed as being allowed for the specific exam
R5	Examinees are prohibited from peeking at and copying answers of other examinees

Since the focus here is on BYOD, it makes most sense to focus specifically on some key security requirements to prevent cheating via the laptop, such as:

SecR1. It shall be impossible to access other resources on the laptop than those specifically allowed for the exam.

SecR2. It shall be impossible to use the laptop for communication with co-examinees or outsiders during the exam.

A key approach to mitigate cheating with BYOD e-exams is the usage of so-called *lock-down browsers* [9]. By locking the screen in a way that cannot be escaped while connected to the exam server, this technology prevents examinees from starting up other programs, opening documents or accessing other web sites than the exam server. The e-exam application which delivers questions to the students and receives answers will typically be running on top a lockdown browser. By these measures, examinees should be prevented from accessing cheat material and getting illegitimate help from accomplices via their laptops − *if* the technology is a 100 % effective.

However, a number of attacks could circumvent lock-down browsers. One simple example: After starting up the lock-down browser, we may be unable to start up Skype to communicate with an accomplice. But what if we have Skype running *before* we start the lock-down browser? This is the outset of our running example. The problem of many hands can easily be envisioned here. The invigilators in the exam room − and the university administration, who give instructions for their conduct − might think that skyping via the laptops during the exams is made impossible by some component of the e-exam technology. The developers of the technology might have been thinking that Skype conversations is something that the invigilators should prevent. There could also be dispersion of responsibility between different technology providers. The developers of the e-exam application might believe that the lock-down browser prevents Skype conversations, while the developers of the lock-down browser consider this outside the scope of their tool, rather to be done by the e-exam application or monitoring software that the university should get from yet another vendor.

2.2 HARM (Hacker Attack Representation Method)

HARM [7] is a method for modeling threats and security attacks in combination with the system architecture, so as to better understand the potential attacks. In this section we summarize the method, so that the extensions that will be proposed later will be understandable. HARM combines several different specification formats to give a comprehensive view of the possible attacks. In the following, we will list these and illustrate them by means of our running example.

Attack Sequence Descriptions (ASD): These are simple natural language descriptions of the attack, forming a sequence of actions. An example ASD could be something like *“(1) Start up a Skype call with an outside accomplice, and have it run in the background. (2) Enter the exam venue and begin the exam in the normal way. (3) Communicate questions to the accomplice and get answers back via Skype, using a hidden wireless earpiece. (4) Type the answers into the e-exam system and submit.”*

Misuse Sequence Diagrams (MUSD): If preferring a more formal form of expression than the natural language ASD, a similar sequence can be described as a MUSD [10]. This is similar to a UML sequence diagram, but in addition to legitimate objects and message calls, it also contain attacking objects and message calls (having red boxes and red arrows). The diagram in Fig. 1 shows the cheating examinee setting up a Skype call with an accomplice before the start of the exam. Then the examinee starts up the lock-down browser and authenticates with an to get an access code to connect with the exam server. Via the Skype connection, the examinee communicates the questions to an accomplice, and the accomplice replies with answers. The dashed red ovals indicate vulnerabilities that are utilized to make the attack work, and their labels are explained to the right of the diagram.

Misuse Case Maps (MUCM): Like MUSD, MUCM [11] also show an attack sequence. The difference is that Misuse Case Maps put more focus on the relationship between the attack sequence and the architecture, showing each step in its architectural context [12], just like Use Case Maps show how legitimate functionality propagates through the architecture [13]. Figure 2 shows a MUCM for another one of the cheating threats investigated in our study, usage of disallowed material. The naïve approach of

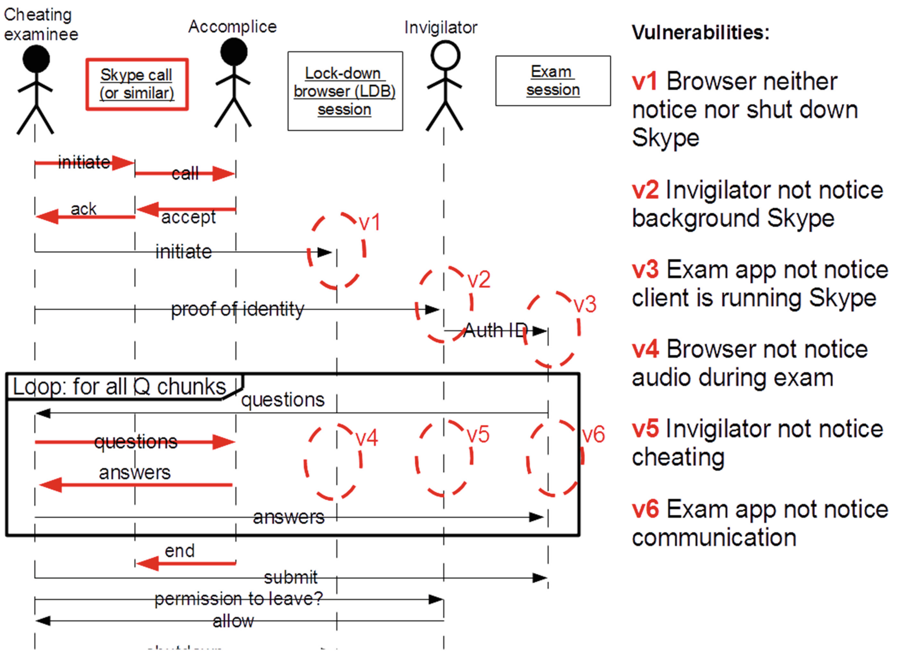
Cheating examinee
Skype call (or similar)
Accomplice
Lock-down browser (LDB) session
Invigilator
Exam session
initiate
call
ack
accept
initiate
v1
v2
proof of identity
Auth ID
v3
Loop: for all Q chunks
questions
questions
answers
v4
v5
v6
answers
end
submit
permission to leave?
allow
Vulnerabilities:
v1 Browser neither notice nor shut down Skype
v2 Invigilator not notice background Skype
v3 Exam app not notice client is running Skype
v4 Browser not notice audio during exam
v5 Invigilator not notice cheating
v6 Exam app not notice communication

Fig. 1. MUSD for a cheat with pre-connected Skype call (Color figure online)

putting cheat files on the laptop's disk or memory sticks might fail if the lock-down browser prevents the opening of any files during the exam. A more sophisticated approach, as pointed out by Dawson [8], is to use a USB key injector containing the cheat notes. It behaves just like a keyboard, and would thus be unlikely to raise suspicion if there is automated monitoring - as students might be allowed to use external keyboards to their laptops for improved ergonomics of typing a lot of text quickly.

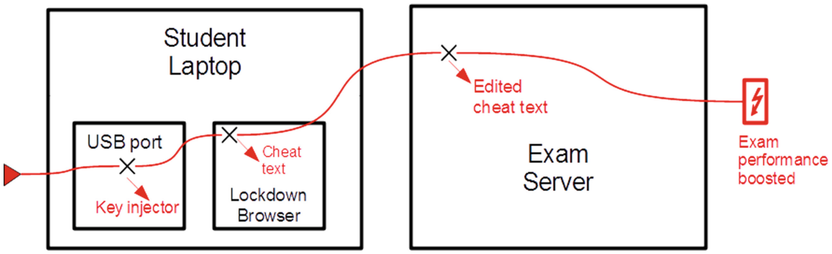

Fig. 2. MUCM for using a key injector with a cheat note

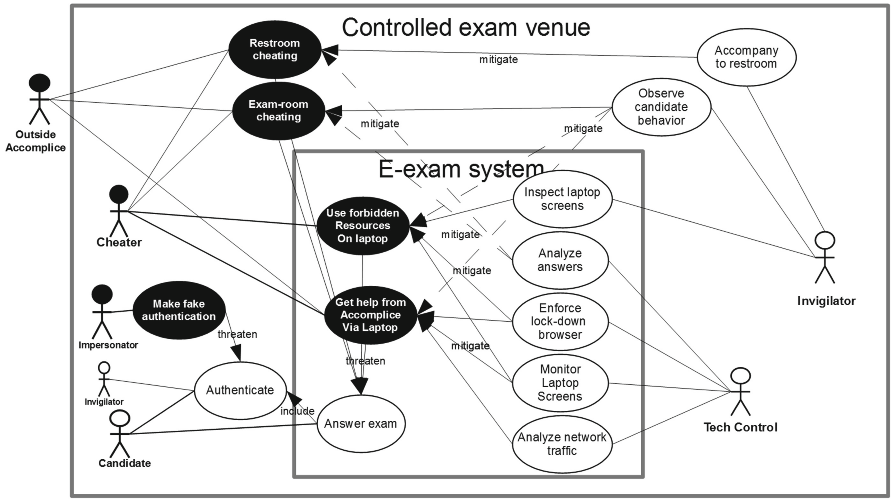

Fig. 3. Misuse case diagram including both electronic and traditional cheating

Misuse Case Diagrams (MUD): MUD extends UML use case diagrams to show how mis-users perform regular as well as irregular activities with the system. Figure 3 shows the MUD for cheating threats studied in our study. Compared to MUSD and MUCM, which show details of one particular type of attack, misuse case diagrams show a broader overview. In the particular diagram in Fig. 3, this overview is made extra broad by showing both the functions and threats particular to the e-exam application (inner system boundary) and cheating threats outside this (e.g., more traditional ways of cheating in the exam room).

Attack Trees (AT): These also show an overview of several threats. Unlike misuse case diagrams, which focus on relationships between threats and legitimate behavior, attack trees focus on the illegitimate behavior alone, breaking high level threats down to more detailed ones. The non-leaf nodes are decomposed into trees of conjunctive ("AND- branch") and disjunctive ("OR-branch") nodes. OR-nodes represent alternatives, while AND nodes represent sub goals where all must be fulfilled to achieve the goal. In Fig. 4, all branches are OR-branches, indicating various ways to perform the high level attack "Cheat during BYOD exams".

3 From Requirements to Penetration Test Cases via HARM

Whereas HARM as illustrated in the previous section has been described in earlier publications, the new contribution of this article is to propose a method to develop penetration test cases aided by HARM. Given some security requirements, like SecR1 and SecR2, there are actually two different approaches that can be used to develop a set of penetration tests:

- **Top down approach:** For each security requirement

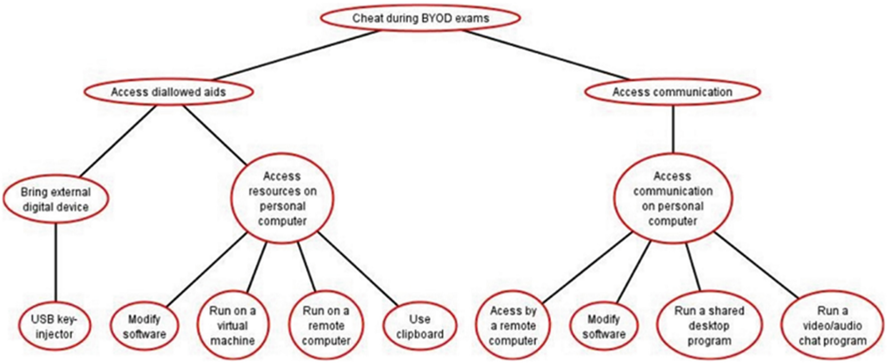
Cheat during BYOD exams
Access diallowed aids
Access communication
Bring external digital device
Access resources on personal computer
Access communication on personal computer
USB key-injector
Modify software
Run on a virtual machine
Run on a remote computer
Use clipboard
Acess by a remote computer
Modify software
Run a shared desktop program
Run a video/audio chat program

Fig. 4. Attack tree for using a key injector with a cheat note, from [14].

  - Make an attack tree, starting with the top level node being a generic violation of that security requirement, then gradually breaking down towards concrete attacks. Brainstorming might be one possible technique to use in developing this tree.
  - Make a misuse case diagram relating attacks to relevant legitimate use cases, including mitigations that are known to be in place. This can be used to eliminate from the attack tree those attacks that are not worth trying, or to adjust them to keep them worthwhile. For instance, if one attack is “Open document” with a cheat file during the exam, this should not be possible with the mitigating use case “Enforce lock-down browser” (cf. Fig. 3). So, to keep “Open document” it should have to be in an AND-relation with “Escape lock-down” in the attack tree.
  - Make attack sequence descriptions explaining how the attack is going to be executed. If necessary, e.g., to understand a technically complicated attack which can be performed in several different ways, complement the simple textual description of the attack sequence with MUCM (if it is useful to see it in the architectural context) or MUSD (if it is useful to see how the cheat attack propagates via various objects and agents).
  - This should be continued until there are attack sequences described for all the leaf nodes of the attack tree.
- **Bottom up approach:** For each security requirement
  - Start with finding some concrete ways of breaking them, and describe these as attack sequence diagrams, possibly also by MUCM and/or MUSD if this is helpful to understand possible attacks and different ways of doing things.
  - When you run out of ideas for concrete attacks, group the similar ones to make the higher level nodes and form the complete attack tree. Make the misuse case diagram to see relationship between attacks and possible countermeasures.
  - It could be a good idea here when the overall attack tree has been formed to work back down in a top down manner, to see if you get any new ideas for possible attacks after seeing the whole picture.

Whatever combination of top-down and bottom-up is chosen, the final step in the planning is to transform the attack sequence descriptions/misuse case maps/misuse sequence diagrams into penetration test scenarios, typically described in tabular form. With a situation similar to the e-exam case, tests would best be developed in two steps:

1. **lab tests**, with the purpose of finding out whether some attack is technically possible or not. Lab tests may investigate small partial attacks one at a time. Table 2 shows a lab penetration test scenario for the cheating via Skype example shown in Fig. 1.
2. **real world tests**, with the purpose of finding out whether attacks are likely to succeed in practice - which may hold bigger challenges than in the relaxed lab setting. Such a test scenario for the Skype example is shown in Table 3.

Since real world tests are more time consuming and expensive than lab tests, it is a good idea to describe the lab tests first. If it turns out that some type of attack was not even possible in the lab, it may be a waste of time to develop a real-life test for it, so resources should rather be spent on other attacks that were more likely of succeeding. (E.g., if we were not even able to have a Skype connection in the lab, there would be little point in trying in the exam-room with the additional challenge of invigilators, etc.). In the planning stage, the rightmost column of Table 2 (Result) would of course be left empty, to be filled in later, while here − to save space, we indicate at once the results that came out of our tests.

Table 2. Penetration test scenario for communicating via Skype

Lab penetration test scenario: *communicate via Skype*

Step	Action	Success criterion	Result
1	Establish Skype connection between examinee's laptop and accomplice's PC	Connection established	OK
2	Start lock-down browser (SEB) on examinee's laptop	SEB running normally	OK
3	Examinee give info to accomplice	At least one works:	OK
3a	Speak	Accomplice hears	OK
3b	Visual (e.g., blink eyes)	Accomplice sees	OK
3c	Share screen	Accomplice sees	–
4	Accomplice give info to examinee	At least one works:	OK
4a	Speak	Examinee hears	OK
4b	Visual (e.g., blink eyes)	Examinee sees	–
4c	Share screen	Examinee sees	–

It can be noted that the penetration test in Table 2 only explores vulnerability v1 and v4 of the MUSD in Fig. 1, namely those related to the lock-down browser. The other vulnerabilities would be explored in the real-world test as described in Table 3.

Table 3. Test case for cheating during exam through assistance from outsider

Real-world penetration test scenario: *get help during exam via Skype*

Step	Action	Success criterion	Result
1	Establish Skype connection	Connection established	
2	Start lock-down browser (SEB)	SEB running normally	
3	Authenticate and access e-exam app	E-exam app starting normally	
4	Open exam question	Exam question appearing on screen	
5	Communicate question to accomplice (e.g., quietly speaking w/wireless hidden mic)	Accomplice receives question; No cheating is detected	
6	Receive hints from accomplice (e.g., through wireless earpiece) and type answer into e-exam app	Examinee receives and types info; No cheating is detected	
7	Repeat 4-6 until all questions answered, then submit	Exam answer submitted; No cheating detected	

In Table 3 the Result column is empty because none of the real-world tests have been performed yet. Whereas lab tests will tend to either succeed or fail, the real-world tests will more often have some probability of succeeding. For instance, it may depend on how far the penetration tester is seated from the nearest invigilator, how clever the tester is at speaking so quietly that it is inaudible to others yet comes through clear enough to the accomplice, how good the tester is at appearing calm in spite of cheating, how attentive the invigilator is, and what kind of other mitigations are in place in the exam room, such as monitoring software to discover suspicious communication from laptops, not matching the profile of the typical interaction between the lock-down browser and the e-exam server. Hence, while the lab test in Table 2 may only need to be run once to establish that skyping was actually possible in spite of the lock-down browser, the test in Table 3 would best be run several times, with different testers and invigilators, in rooms with different types of background noise, seated in different positions. This would enable to gather some statistics, like probability of getting caught, or mean time to failure (i.e., getting caught), to rank the attack relative to other attacks to determine which ones are most urgent to deal with.

4 Case-Study: Cheating-Related Exam Security

As part of a student project by the second author (supervised by the first and third author), a number of attacks were tested on a certain lock-down browser, namely Safe Exam Browser [15]. This browser was chosen because it is open source, and because the e-exam tool that our university is using, partly relies on that browser for security during the exams. It should be noted that the project did not try to cover the complete set of security related to e-exams. The following limitations were chosen:

- only look at threats *during* the exam, not before (e.g., getting premature access to exam questions) or after (e.g., manipulating answers after delivery, or manipulating grades).
- only look at *cheating* threats, not other kinds of security threats (e.g., like sabotage of the exam, denial of service). Although such other threats may also need to be handled, they are not threats that give a grade advantage and thus not classified as cheating.
- due to time and resource limitations, only lab tests were actually executed, while the real-world tests remained at the idea level.

Table 4 sums up results for all the different test cases that were tried in the project. Note that "Success" in the Result column means from the penetration tester's (i.e., attacker's) point of view. From the secure e-exam point of view, then, it is the rows with "Fail" that are the successful ones. So, it can be seen that SEB prevents well against attempts to circumvent it by running on a virtual machine when starting the lockdown browser (if this was not prevented against, the examinee could during the exam shift execution from the virtual to the real machine and then run any forbidden application). It also protects well against attempts to hide cheat text in the clipboard and then try to paste it once the exam has started, and as far as we could find, the examinee would not be able to share her desktop with an accomplice. As the table indicates, however, several other cheating options were available, potentially enabling a candidate with very little subject knowledge to get help from somebody much more clever, in the worst case getting an A where an F would have been the correct account of the examinee's competence. The results of the tests have been communicated to SEB developers, so these weaknesses may likely be mended in future versions of the software. It should also be noted − as pointed out in the previous section - that the success of the four lab attacks in Table 4 does not necessarily mean that the same attacks would be certain to succeed in a real-world exam situation, where there would be a combination of several tools involved, plus human invigilators to oversee the candidates. But some of the attacks do not require much visibly suspicious behavior by the examinee, so could be assumed hard to spot by invigilators.

5 Related Work

Dawson [8] presents five attacks against BYOD e-exams, whereof 4 were tried with various e-exam tools and found successful with at least one tool each. Some of the attacks tried out in our work are inspired by his proposals, especially the key injector attack and the Skype call attack. Dawson, however, does not present any modeling approach or other systematic approach to get from requirements to a test plan.

Cota et al. [16] proposed a framework, RACOON, which is a semi-automatic approach to configure accountability mechanisms (e.g. logging, auditing, monitoring) and reputation mechanisms on the P2P systems. The accountability mechanism helps to monitor cheating whereas the reputation mechanism helps to punish in case of cheating. The paper also discussed the approach to find cheating in the systems through game based simulations using game theory. Although the approach discussed in their paper useful to find cheating in digital exams, the details of penetration tests were not provided in the paper, which is the main consideration for our paper.

Table 4. Tests completed in the project [14] so far

Attack	Result	Description
Inject notes into exam software with USB key injector	Success	We saved a text on a rubber ducky USB and the string was injected into the web page open in SEB
Run SEB on a virtual machine	Fail	When initiating SEB, a pop-up window appears, stating that SEB has detected a virtual machine and will not work
Run SEB on a remote computer	Success	We managed to control SEB from a remote computer, while using SEB
Use clipboard to import notes into exam software	Fail	We were not able to right click or use CTRL + P to paste the clipboard content into SEB
Get assistance by being accessed from a remote computer	Success	We managed to control and access an SEB exam environment from a remote computer
Get assistance by sharing desktop	Fail	Neither Google Hangout nor Skype showed SEB with remote desktop, when it was initiated
Get assistance by communicating with audio/video	Success	Both examinee and assistant can hear each other and use their microphones. The assistant can also see the examinee on camera during a video conversation, but the examinees only sees the SEB environment

Wang et al. [17] present an approach to security testing based on threat models. Using UML sequence diagrams, there is some similarity with our approach (especially the misuse sequence diagrams), but the approach of Wang et al. is more formal, aiming to support automatic generation of test cases, while our approach aims to support brainstorming of test cases that will be performed manually. Other approaches aiming for partly automated generation of test cases from various types of models can be found in [18, 19], and a review of various model-based security testing techniques can be found in [20]. Agile security testing, proposed in [21], uses abuse stories or misuse cases as a starting point, thus having some resemblance with our approach, and in [22] it is further discussed how this can be fit into Scrum. These approaches have some similarities with ours in the initial part, having misuse cases as a possible starting point. Our approach however lacks the connection to agile/Scrum and does not make any assumption about the process, and instead proposes the choice of several different modeling representations, depending on what is found most fitting in the situation.

6 Conclusions and Further Work

This paper has proposed an approach to using models as a basis for brainstorming possible attacks and developing these into penetration tests. It must be admitted that the validation is so far limited, with only 8 lab tests executed so far. Future work in the investigation about e-exams would be to include a broader range of tests, including real-world. Indeed, real-world testing could also be applied to traditional pencil and paper exams, for instance to create a benchmark to establish if cheating is easier with e-exams than with traditional paper exams, which − although often intuitively assumed − need not be the case [23]. Since paper exams are not 100 % secure against cheating either, e-exams may be preferred even in spite of weaknesses, if they are found to have advantages in other respects [8, 24].

For the validation of the proposed method, future work could include experiments to investigate whether people come up with more or better penetration tests if using these modeling languages than if using other approaches (either completely ad hoc, some of those presented in related work, or other modeling approaches like for instance goal-oriented models). It would also be interesting to see if a top-down or bottom-up process to attack brainstorming is the most effective, as well as whether brainstorming is most effective in groups or individually.

References

1. Barmi, Z.A., Ebrahimi, A.H., Feldt, R.: Alignment of requirements specification and testing: a systematic mapping study. In: 2011 IEEE Fourth International Conference on Software Testing, Verification and Validation Workshops (ICSTW). IEEE (2011)
2. Unterkalmsteiner, M., Feldt, R., Gorschek, T.: A taxonomy for requirements engineering and software test alignment. ACM Trans. Soft. Eng. Method. (TOSEM) **23**(2), 16 (2014)
3. Talukder, A.K., et al. Security-aware software development life cycle (SaSDLC) - processes and tools. In: IFIP International Conference on Wireless and Optical Communications Networks, WOCN 2009 (2009)
4. Arkin, B., Stender, S., McGraw, G.: Software penetration testing. IEEE Secur. Priv. **1**, 84–87 (2005)
5. Palmer, C.C.: Ethical hacking. IBM Syst. J. **40**(3), 769–780 (2001)
6. McDermott, J.P., Attack net penetration testing. In: Proceedings of the 2000 Workshop on New Security Paradigms, pp. 15–21. ACM: Ballycotton, County Cork, Ireland (2000)
7. Karpati, P., Opdahl, A., Sindre, G.: HARM: hacker attack representation method. In: Cordeiro, J., Virvou, M., Shishkov, B. (eds.) Software and Data Technologies, pp. 156–175. Springer, Heidelberg (2013)
8. Dawson, P., Five ways to hack and cheat with bring-your-own-device electronic examinations. Br. J. Educ. Technol. (2015). http://onlinelibrary.wiley.com/doi/10.1111/bjet.12246/epdf
9. Frankl, G., Schartner, P., Zebedin, G.: Secure online exams using students' devices. In: 2012 IEEE Global Engineering Education Conference (EDUCON). IEEE (2012)
10. Katta, V., Karpati, P., Opdahl, A.L., Raspotnig, C., Sindre, G.: Comparing two techniques for intrusion visualization. In: van Bommel, P., Hoppenbrouwers, S., Overbeek, S., Proper, E., Barjis, J. (eds.) PoEM 2010. LNBIP, vol. 68, pp. 1–15. Springer, Heidelberg (2010)

11. Karpati, P., Sindre, G., Opdahl, A.L.: Visualizing cyber attacks with misuse case maps. In: Wieringa, R., Persson, A. (eds.) REFSQ 2010. LNCS, vol. 6182, pp. 262–275. Springer, Heidelberg (2010)
12. Karpati, P., Opdahl, A.L., Sindre, G.: Investigating security threats in architectural context: Experimental evaluations of misuse case maps. J. Syst. Soft. **104**, 90–111 (2015)
13. Amyot, D., et al.: Generating scenarios from use case map specifications. QSIC **3**, 108–115 (2003)
14. Søgaard, T.M.: Cheating Threats in Digital BYOD Exams: A Preliminary Investigation. NTNU, Trondheim (2015)
15. Schneider, D.: Safe exam browser 2.0 how to (Install, Configure, Deploy and Use SEB 2.0) (2014). http://safeexambrowser.org/presentations/HowTo_SEB2.0.pdf
16. Cota, G.L., et al.: A framework for the design configuration of accountable selfish-resilient peer-to-peer systems. In: 2015 IEEE 34th Symposium on Reliable Distributed Systems (SRDS). IEEE (2015)
17. Wang, L., Wong, E., Xu, D.: A threat model driven approach for security testing. In: Proceedings of the Third International Workshop on Software Engineering for Secure Systems. IEEE Computer Society (2007)
18. Xu, D., et al.: Automated security test generation with formal threat models. IEEE Trans. Dependable Secure Comput. **9**(4), 526–540 (2012)
19. Marback, A., et al.: A threat model-based approach to security testing. Soft. Pract. Experience **43**(2), 241–258 (2013)
20. Schieferdecker, I., Grossmann, J., Schneider, M.: Model-based security testing (2012). arXiv preprint arXiv:1202.6118
21. Tappenden, A., et al.: Agile security testing of web-based systems via httpunit. In: Proceedings of the Agile Conference, 2005. IEEE (2005)
22. Erdogan, G., Meland, P.H., Mathieson, D.: Security testing in agile web application development - a case study using the EAST methodology. In: Sillitti, A., Martin, A., Wang, X., Whitworth, E. (eds.) XP 2010. LNBIP, vol. 48, pp. 14–27. Springer, Heidelberg (2010)
23. Sindre, G., Vegendla, A.: E-exams versus paper-based exams: a comparative analysis of security threats and countermeasures. In: Norwegian Information Security Conference (NISK 2015). Bibsys OJS: Ålesund (2015)
24. Sindre, G., Vegendla, A.: E-exams and exam process improvement. In: UDIT 2015. Bibsys OJS: Ålesund (2015)