Teaching a Project-based Course at a Safe Distance: An Experience Report

Malte Mues
Automated Quality Assurance Group
TU Dortmund
Dortmund, Germany
malte.mues@tu-dortmund.de

Falk Howar
Automated Quality Assurance Group
TU Dortmund
Dortmund, Germany
falk.howar@tu-dortmund.de

***Abstract*—IT security is an important aspect of system design and of quality assurance during the software engineering process. Today, there is a big demand for IT security specialists in job markets around the world. Increased automation of security code reviews is one approach for mitigating the current shortage of IT security professionals. We designed the course "Formal Methods for IT Security" to teach undergraduate students the basics of constraint solving and formal modeling techniques suitable for automation of IT security code reviews in a hands-on format. In this paper, we describe the didactic concept of the course along with the required modifications due to the COVID-19 pandemic. Further, we report our experience from remote teaching the class during the summer term affected by the pandemic. The main pandemic-related challenge we tackled during the course is establishing communication and stimulation of the discussions required for learning in projects without any presence meetings.**

***Index Terms*—Project Based Teaching, Distance Feedback Culture, IT Security Teaching**

I. Introduction

The overall job market for IT jobs is growing intensively. According to a study from Burning Glass Technologies[1] the rate of growth in IT security related job offerings increased 3 times faster between 2013 and 2017 than the rate of growth for IT job offerings in general during this period. As the educational pipeline for training IT security experts does not increase capacity at the same growth rate, the demand for automation skills paired with IT security knowledge increased in the same period by 255% according to Burning Glass Technologies. This observation of the job offering trends aligns well with reports of the supremacy of automated security analysis that supplements and guides human attention during security audits (c.f. [1]–[3]).

In the German summer term 2020, we decided to teach a hands-on course called "Formal Methods for IT Security" at TU Dortmund, as our research interest focuses on formal method-based security analysis and it aligns well with the described trend for security professionals. The offer addresses undergraduate students in the final phase of their bachelor's degree and is worth seven points in the European Credit Transfer System (ECTS). The summer term 2020 is the first instance of this new course, which we planned as a project-based format. Apart from aligning well with our own interests and the current demand for security experts, the main focus of the computer science education at TU Dortmund University is the theoretical foundations of computer science and covers large parts of the theoretical literature for system description and system design. The students have a basic understanding of abstract interpretation techniques, symbolic models of source code execution, and decidability in logic. Given these starting conditions, we decided that a hands-on project demonstrating how these theoretical concepts can be used to build security analysis tools is a good fit. The course should connect the theoretical foundations with their applications in tools.

In this paper, we describe the learning goals of the newly designed course and outline the concept for teaching it along with the modifications required due to the COVID-19 pandemic. We report on our experience on how to establish communication and feedback culture in a distance learning setting. The main challenge we had to address was that course participants did not know each other in advance and never met in person during the course period.

Outline. We first describe the teaching goals and the content of the course in Section II. Next, Section III explains the course design and outlines the used teaching methods. Our experience is summarized in Section IV. Finally, Section V concludes the paper.

II. Teaching Goals

Before we explain how we teach this course, we define what we want to teach. The title "Formal Methods for IT Security" already points out that the course consists of two main topics: Formal methods and IT security. Moreover, the course is an instance of a hands-on project. Every undergrad student has to do one hands-on project (so-called *Fachprojekt*) before graduation and it should be scheduled in one of the last two semesters according to the degree program. The curriculum requires us to include some basic software engineering skills into the course covering general skills from the area of compiling, build management, and source code management. The course should also offer a platform for practicing teamwork and group communication.

[1]https://www.burning-glass.com/wp-content/uploads/recruiting_watchers_cybersecurity_hiring.pdf

IT Security. IT security is a broad term covering many different fields. For this course, we are mainly interested in one particular aspect of IT security: programming errors that might be detected during testing with appropriate source code analysis tools. The main focus of this course is on precise and automated methods for detecting coding errors with a potential impact on IT security. By the end of the course, students should see a potential path to security tests as part of a unit test suite. Before we define this aspect of IT security in greater detail, we will explain what we consider to be the "soft" side of IT security and what we consider the "hard facts".

The soft side of IT security is related to the development and operations processes around Software. The Guide to the Software Engineering Body of Knowledge (SWEBOK) [4] deals with IT security only as part of the process in the main headings listed in the table of contents: The heading of Section 2.7 in Chapter 2 names security as part of the system design process and the heading of Section 17 in Chapter 13 mentions secure software development. The system design part underlines that it is an important activity to design explicitly a system that meets the security requirements. But in the requirements section of the SWEBOK security requirements are condensed to sub aspects of the *Process Quality and Improvement* requirements. The presented lead questions for the elicitation of security requirements cover confidentiality, integrity, and availability (sometimes referred to as the CIA-Triad) of the system. In this framework, the security of a product becomes an aspect of the software engineering process but is not relevant during software development. The safety of software, in contrast, is a major part of the software quality chapter. As a consequence, we see an increased interest in establishing software safety as part of the development process, while IT security is often relegated to security audits at the end of the development cycle or even to operation and is addressed by additional measures, e.g., a web application firewall (WAF) (cf. [5]).

The defense-in-depth principle requires emerging programmers and software engineers to reduce IT security risks in source code (hard facts) to a minimum. For example, an SQL injection cannot occur, if a proper input sanitization and query building mechanism is used. This is a hard fact (done correctly or incorrectly) in the source code base and can be detected during development and system design as it is independent of requirements and runtime settings. The community maintains this kind of knowledge in the Common Weakness Enumeration (CWE)[2] database, and it is time to integrate this knowledge into software quality checks the same way we establish software safety checks. We can observe this trend in the industry driven by global players like AWS (e.g. [6]) or the joint initiative between different stakeholders in the automotive domain defining ISO/SAE 21434 for cybersecurity engineering for road vehicles.

For the course design, we decided to discuss both sides of IT security. Students should become aware of the broad spectrum of IT security topics. We structured this overview using the introduction of Eckert's book "IT-Sicherheit" [7]. In the first half of the semester, we teach existing vulnerabilities and how these can be exploited, e.g., in a Metasploit[3] module. The course also discusses which protection goals of the CIA-Triad are compromised in a successful attack. The second half of the semester focuses on the automated detection of security weaknesses in source code during the development phase: Students should get a basic understanding of the common penetration testing workflow and common penetration testing tools, e.g., Kali Linux [4], Wireshark [5], or nmap [6]. We do not cover every detail of these tools but students should be able to build upon the understanding gained in the course and continue to learn about penetration testing during their professional life.

Formal Methods. We use the term *formal methods* to describe modeling techniques that allow one to precisely reason about source code execution paths in a mathematical model. There are many different formal approaches to the correctness of code and we cannot cover all of them in this course. We focus on a subset suitable for analyzing security weaknesses in source code. Motivated by our own research background, the subset discussed in the course consists of tools based on symbolic execution [8]. We work intensively on a dynamic symbolic execution engine for Java called JDart [9], [10] and have a basic understanding of CBMC [11], a bounded model checker suitable for C programs. Both tools use similar high-level concepts in the analysis, while they implement those concepts quite differently, allowing for a discussion of different technical approaches. We refer the interested reader to the results of SV-Comp 2020 [12] for an impression of other tools and other possible answers to those problems.

Symbolic execution is based on a symbolic encoding of the computation done during code execution, resulting in a logic model of the effects along a single execution path. Joining the models of individual execution paths yields a model of the reachable state space that allows reasoning about variable values in its logic-based encoding. The course explains the basics of the ideas used in this logic-based encoding, as single static assignment forms and how they might be encoded into an SMT problem.

It is explicitly not the goal of the course to cover SMT solver internals or the conversion form SMT problems into SAT problems. Students should create a connection between the abstract theoretical foundations taught as part of the curriculum at TU Dortmund and this potential application. For example, the theory on linear programs, the simplex algorithm, and the 3-SAT problem get a context in which it unleashes analysis power for real-world problems. The course should prepare students to become highly skilled professionals in the area of software analysis tools.

[2]https://cwe.mitre.org

[3]https://metasploit.com
[4]https://www.kali.org
[5]https://www.wireshark.org
[6]https://nmap.org

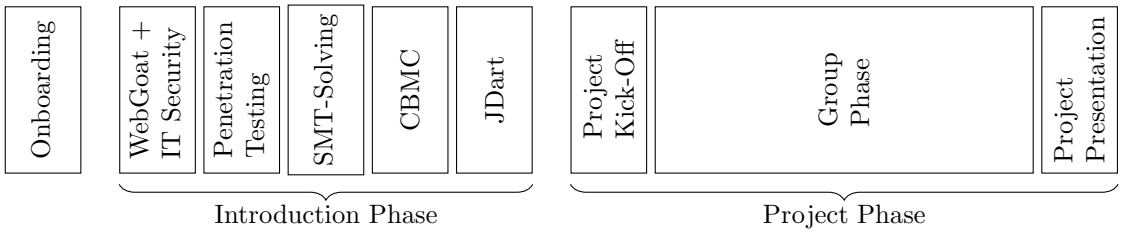

Fig. 1. A representation of the course schedule and structure over the term from the left to the right. Each small block represents one week and the activity scheduled for the week. The group phase consists of 5 weeks.

Software Engineering. Teaching software engineering is complex as it is a multi-dimensional topic. We think software engineering is more than only writing a program or designing a system. Without a question, the design of a system is important and someone has to write it, potentially also a software engineer. But software engineering is more: It is also planning a system, learning to debug, to operate, and to test a system. The goal is to elicit requirements and satisfy customers with the build solution, even if the design violates mainstream trends in certain aspects on purpose because the software engineer knows that it works out better this way. This requires strategies for problem-solving in the same way as it requires experience and theoretical knowledge with software systems.

We cannot teach experience, we can only create settings where students make some initial experiences that empower them to keep experimenting on the job. An important outcome of being able to make experiments that lead to experience is mastering the used tools and programming languages. There is a recent observation made by teaching professionals is that tooling and a required "computing ecosystem literacy" is often missing in modern computer science education[7]. The hands-on project format of this course is one of two places in the undergrad curriculum at TU Dortmund, where tooling might be part of the course. We aim to seize this opportunity with the course design and include tasks that should lead to a better understanding of editors, support tools (e.g. shells), source code versioning, build systems, and virtual machines.

Soft Skills. Non-technical soft skills gain importance for IT professionals. Discussions and blog posts[8] about career paths for software engineers focus on the difference between a junior, a mid-level, and a senior developer: not only technical skills are required to climb up the ladder, but also soft skills regarding communication and team management — a view also expressed by Robert C. Martin in his book *The Clean Coder* [13].

While the undergrad computer science curriculum at TU Dortmund does not cover the basics of communication and team dynamics in an explicit course we believe that we can increase the career prospects of students after the program by creating a protected group setting in which students can gain experience and train their soft skills. For this hands-on project, we aim for a group work setting, where the students are challenged to solve a non-trivial task as a group and present the results together. We leave it to the group for how much support from instructors they ask. They should learn to take responsibility for their learning result and group success. Creating a setting where students have to help each other in a group allows them to develop strategies to tackle challenges together. Everyone gets the chance to bring in his or her skill set and to teach others or learn from them.

III. Course Design

The presented course is designed to fit the shortened German summer term in 2020 spanning 13 weeks due to the COVID-19 pandemic instead of the planned 15 weeks. As it is worth 7 ECTS, we might expect students to invest between 175 and 210 hours, which equals up to two days a week. The number of participants is limited to 20, enabling the instructors to interact closely with the students.

Figure 1 shows the course layout and the planned design in two phases. After one week of onboarding, the first half of the summer term introduces different tools and discusses the meaning of IT security. During the second half, the students run a small project in groups of four students and present them at the end of the term. The term is split into a 13 weeks lecture period hosting all the classes followed by some lecture-free time for labs and exams. The course is designed to align with the lecture period. We identified four challenges to tackle: Forming a learning group, introduction of content, transitioning to the project phase, and the final presentations.

Forming a Learning Group. When we started to plan the course in a distance learning setting, we identified the formation of a learning group as a key challenge to the success of the summer term. Consequently, the design of adequate means of communication with maximal ease of use became one of our main focuses in the short period of preparation.

To make up for the missing possibilities of personal interaction around classes on the campus, we provided a protected chat room for the course on a university-affiliated Matrix chat server. Students could also use the chat server to interact with each other in smaller groups or bilateral chats. Chatting is one cornerstone of basic asynchronous communication. In previous project-based formats, we have learned that chatting works better for this kind of communication between students than

[7]https://missing.csail.mit.edu/about/

[8]https://medium.com/better-programming/the-differences-between-a-junior-mid-level-and-senior-developer-bb2cb2eb000d

E-mails. Next, a platform for collaborative programming and working is required. TU Dortmund University uses Zoom as video conferencing platform for small student group meetings during the pandemic. Supporting video chat and screen sharing, this tool provides a communication channel that allows remote pair or group programming. For lectures, we hosted a Big Blue Button video conference server, which became a standard lecture tool in our group. The communication infrastructure is complemented with a dedicated GitLab instance for teaching and student projects that allows students to host software projects with a git and wiki space. Last but not least, we set up a Moodle room for providing material for the course.

Providing communication infrastructure is only the first step. The students have to be onboarded into the course. Forming a group with an open, mutually supportive, and collaborative attitude out of 20 individuals is always a challenging task at the start of a new term. We have experience in how to do this in presence classes: In didactic courses, we learned methods like a living statistic or a map of commonalities to activate students and trigger discussions between them. When we started plans for the new course design, the initial idea was to use one of those methods along with a barbecue to kick-start conversations between the students. TU Dortmund University has many students who commute to the university for classes but do not live on the campus or in the city. Commute times up to 90 minutes (single distance) are common. As a consequence, student groups are often very heterogeneous and only meet for studying. Given this situation, we cannot assume that students know each other as they are all studying along for several terms nor do we know them from previous courses.

To introduce the students to the group, we set up a wiki in the Moodle room and assigned pairs. The task was to interview the other person and to write a short presentation about the interview partner in this wiki. This way, we pair up every student with at least one other person in the course. The Moodle room contained a welcome message from the instructors to the students as a video on the very top of the Moodle room. It was already available when students got signed into the room. Regarding onboarding to the chat room, we posted a description on how to set up the chat in the Moodle room and expected students to contact us to get invited to the room. We provided all other accounts in upfront and prepared a project space for the course in the GitLab.

Introduction of Content. Following the teaching goals, we identified four thematic blocks of content for which we had to provide input at the beginning: IT security, SMT solving, bounded model checking (in the form of CBMC), and dynamic symbolic execution (in the form of JDart). For the introduction to IT security, we decided to split the content into two sub-blocks. First, we discuss the term IT security and have a detailed look at SQL injection vulnerabilities. Second, we provide an introduction to penetration testing (cf. left of Figure 1).

For the SQL injections, we forked the OWASP WebGoat project[9]. OWASP WebGoat is an educational web application that allows to explain the basics of certain security vulnerabilities in a hands-on fashion. The tool provides instant feedback, whether the student was successful or not in performing an injection attack. Provided instructions and tasks are blended in one interface. This is an effective way to teach knowledge, especially as there is no feedback channel between instructor and student required in this setting as the feedback from the system affirms the student's learning progress in real-time. Our fork of WebGoat is mainly a German translation of the English class for SQL injection attacks and provides some additional course-specific information. We provided the tool as a runnable Docker container in the course registry on the GitLab instance. Apart from learning something about IT security, students had to learn some Docker skills, including registry authentication with Docker.

In the past, capture the flag (CTF) events evolved as a standard for teaching attack vectors on systems (e.g. [14], [15]). As this is the basis of penetration testing, we had planned to host a CTF challenge in presence. To comply with the law, at least in Germany, it is recommended to isolate the teaching system used for the CTF event from the Internet so that it is not possible to attack any real system by accident. Due to the COVID-19 pandemic, however, we could not create a security lab on the campus running a CTF session with an isolated network. Instead, we provided a detailed description, how to set up an isolated network in VirtualBox[10] running multiple virtual machines (VMs) within the network. The task introduced the students to concepts of virtualization and VM images apart from the penetration testing concepts. For the exercise, we used a Kali Linux VM running the penetration testing tools. As targets, we used the training target VMs of the Metasploit framework: Metasploitable 2[11] and Metasploitable 3[12]. For the IT security content, we covered in greater detail how to use the Metasploit framework as well as some other tools of the Kali Linux that might be useful for debugging, e.g., Wireshark, including the legal restrictions when using these tools. The students had to explain for a Metasploit module of their choice how it works and for which part of an attack it might be used for penetration testing. They had to establish a connection to protection goals and potential impact. This is only the starting step for a penetration testing career, but we hope that it might help curious students to continue this learning path on their own.

To introduce the basics of SMT solving and the SMT-Lib format, we used the Z3 tutorial provided by Microsoft Research[13]. Then we provided some tasks from the SV-Comp benchmark repository[14] and discussed a logic-based encoding in the SMT-Lib format that allows us to model the program semantic. For the introduction of bounded model checking

[9]https://owasp.org/www-project-webgoat/
[10]https://www.virtualbox.org
[11]https://docs.rapid7.com/metasploit/metasploitable-2/
[12]https://github.com/rapid7/metasploitable3
[13]https://rise4fun.com/z3/tutorial
[14]https://github.com/sosy-lab/sv-benchmarks

in CBMC and dynamic symbolic execution in JDart we also used tasks form the SV-Comp benchmark repository to get the tools running and demonstrated some features. In a short introductory video, we additionally explained the basics of dynamic tainting.

The instructions and explanations for these course tasks have been provided as videos (between 20 minutes and 60 minutes long). Then students had a week to process the tasks and prepare a short preparation of the results. Once a week, we held a plenary session in which we discussed the results, questions, and learning progress. In addition to the plenary sessions, we offered a 30-minute session for video consultation twice a week. One of the instructors was available in a video chat room and students could join with questions related to the course.

Project Phase. After the introduction into different topics, we planned a phase in which students had to propose projects they want to work on. A project should cover security analysis of a Java project with JDart or a C project with CBMC. The projects should be completed in groups of four. Each group should present their final results at the end in a final plenary session. During the project, the students should collect experience with compiling and packaging projects such as the creation of suitable tests and analysis environments for those projects.

Final Presentation. The presentation of work results is a central part of successful software projects. Originally, we planned a poster session for which every project group designs a poster and explains the project at their poster instead of giving a standard talk with a slide deck. Due to the pandemic, we changed this plan: students had to give an online presentation in Big Blue Button. The pandemic created a situation in which screen sharing knowledge became a central part of the presentation challenge. This does not support the presentation skills in the same way as designing a poster might do, but it still improves the student's skill set in relevant aspects: Holding a presentation online is a chance for practicing presentation skills required in distributed teams.

IV. Lessons Learned

After completing the summer term, we were surprised by how the course worked remotely. The students stayed motivated across major parts of the project and managed to overcome all frustration periods during the project phase. Some colleagues reported unusually high dropout rates during the term as students started highly motivated into the online semester but stopped participation over the term. We have not observed any unusual dropout rates compared to presence courses in the past. The students reached the presented goal for IT security knowledge, made progress in mastering their tools, and presented successful projects using the introduced tools in a video conference. This worked well. In the remainder of this section, we report observations on communication and broach the issue of feedback in remote teaching since we perceived these two points as the main challenges for remote teaching the course. We also report briefly on the effect of the shortened term.

Communication. As described in the previous section, we have invested a lot of planning effort into establishing communication. The partner interview task already demonstrated that this is required as very few students know each other from previous courses at TU Dortmund University. Many students have reached out asking how they might get in contact with their partner. All students are in the Moodle room and Moodle supports a communication format based on messages, so they could have contacted each other within the tool with a single click. But at least our students were not familiar with this communication feature and could not reach out to each other. The partner interview challenged them to overcome actively this communication barrier in the very first week of the term.

In the second week, when we handed out the first exercises, students actively asked whether they are allowed to hand in results as pairs. We observe that many of the initial interview pairs formed a learning tandem for the first half of the term and continued to work as a team. They helped each other out with setting up software and solved the tasks.

Overall, 17 out of 20 students followed the invitation into the chat room. Over time, we could observe that interactions between students in the chatroom evolved around the different tasks they had to process. Many students seem to study during weekends, so they posted their questions, and often someone else in the course had already answered them long before we have even read them. We were delighted to observe this. It shows that it is possible to overcome the distance between students even though everyone is at home and it is possible to establish a form of collaboration that normally evolves in a lab at the university.

During the project phase, we observed that students continued to rely on the provided communication channels to contact instructors but chose Discord as a platform for organizing the work of their groups. We think this is mainly motivated by the fact that students already have been using Discord before the pandemic for gaming and social interaction. Some of the groups invited us into the Discord groups and we were pleased to see that over time not only technical discussions evolved but also informal communication within the group. We believe that this is crucial for establishing a supportive learning environment and for establishing lasting networks of students. Some groups of students are planning to head out together for an informal social activity after a relaxation of the COVID-19 contact restrictions.

Feedback. The chosen format for providing material to the course is a one-way communication channel. We uploaded videos in which we described everything we thought helps tackle the next exercise in addition to some context. Students did not have a chance to ask questions during the video production or influence the level of detail in which we provided information. During consultation hours, students had a chance to ask any remaining questions after they watched the videos or could bring up questions in the discussion of the work results

in the plenum. It was a challenge during video production to balance the level of detail, providing sufficient information while keeping the videos interesting and short.

In general, students reported back that the videos helped to start working on the exercises but sometimes also skipped important information, they had to investigate on their own. Moreover, students liked the videos because it allowed them to learn whenever it did fit their schedule instead of being bound to lecture times. The online semester therefore allowed flexibility in planning for them.

We chose to provide installation instructions for tools in a platform-independent way but demonstrated installation on our main operating system. Students had to transfer to their operating system. This raised many conflicts, which, most of the time, could be solved in a short amount of time, and solutions spread quite well between students in the chat rooms. Implicitly, students learned something about writing technical documentation and the importance of knowledge sharing.

At the end of the term, many of the students provided feedback in a questionnaire that they might have preferred a fixed VM containing all tools we are going to work with. But they also lined out that while such a VM would have saved them a lot of time for this specific course, they learned many tricks in mastering their development environment. As an example, some of them learned to love the power of shell scripts. We are personally grateful for this feedback and are convinced that this is a positive result for the software engineering teaching goals. Based on this observation, we would recommend sticking with partial installation instructions from time to time instead of providing ready-to-use virtual machines. This term showed us that the resulting conflicts are valuable parts of the learning experience. It helped students to take responsibility for their learning progress.

Shortened Term. The decision to shorten the summer term was communicated to us two weeks before the planned start of the course. This left us with four weeks for concept modifications and setting up the new learning experience. Shortening the term to 13 weeks instead of 15 weeks created time pressure affecting the course in two regards. We had less time to explain tooling and we had to reduce the complexity of the projects slightly to make them manageable in the given time frame. Students did not have sufficient time to reach a comfortable baseline with the general tooling (e.g. IDEs, compiler, and git) and CBMC or JDart specifically before the project phase started. As a result, the students did not feel comfortable proposing projects and rejected the task. Instead, we handed out topics for the project phase. The course would have benefited from more time for learning general tool-related skills before the project phase.

To provide a stable and predictive learning environment, we decided to switch to a fully remote course on the same day we were informed about the delayed online semester start. While it was initially unclear whether in-person meetings would become possible in the second half of the term, we ruled out this possibility from the beginning to keep the summer term plannable for us and the students. The students welcomed this decision.

V. Conclusion

In this paper, we present a concept for a hands-on course combining formal methods for IT security analysis with the basics of penetration testing. Due to the COVID-19 pandemic, we had to teach the class completely remotely and present challenges arising from teaching a project-based course fully remotely. We focused on how we established communication in the group and which experience we made with remote only group building. All in all, we report on a positive experience with remote teaching and conclude that online courses might be beneficial for students regarding future career prospects as they strengthen remote work skills such as video conferencing. For the future, we think the proposed course would benefit most if the first phase of knowledge introduction is in the classroom and the second phase of group work is taught at distance.

References

[1] B. Cook, "Formal reasoning about the security of amazon web services," in *International Conference on Computer Aided Verification*. Springer, 2018, pp. 38–47.

[2] P. Godefroid, M. Y. Levin, and D. Molnar, "Sage: whitebox fuzzing for security testing," *Queue*, vol. 10, no. 1, pp. 20–27, 2012.

[3] D. Distefano, M. Fähndrich, F. Logozzo, and P. W. O'Hearn, "Scaling static analyses at facebook," *Commun. ACM*, vol. 62, no. 8, p. 62–70, Jul. 2019. [Online]. Available: https://doi.org/10.1145/3338112

[4] P. Bourque and R. E. Fairley, Eds., *Guide to the Software Engineering Body of Knowledge*, version 3.0 ed. Los Alamitos, CA: IEEE Computer Society, 2014. [Online]. Available: http://www.swebok.org/

[5] P. Byrne, "Application firewalls in a defence-in-depth design," *Network Security*, vol. 2006, no. 9, pp. 9 – 11, 2006. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1353485806704226

[6] B. Cook, K. Khazem, D. Kroening, S. Tasiran, M. Tautschnig, and M. R. Tuttle, "Model checking boot code from aws data centers," *Formal Methods in System Design*, pp. 1–19, 2020.

[7] C. Eckert, *IT-Sicherheit: Konzepte-Verfahren-Protokolle*, 10th ed. De Gruyter, 2018.

[8] J. C. King, "Symbolic execution and program testing," *Communications of the ACM*, vol. 19, no. 7, pp. 385–394, 1976.

[9] M. Mues and F. Howar, "Jdart: Dynamic symbolic execution for java bytecode (competition contribution)," in *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*. Springer, 2020, pp. 398–402.

[10] K. Luckow, M. Dimjašević, D. Giannakopoulou, F. Howar, M. Isberner, T. Kahsai, Z. Rakamarić, and V. Raman, "Jdart: A dynamic symbolic analysis framework," in *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*. Springer, 2016, pp. 442–459.

[11] E. Clarke, D. Kroening, and F. Lerda, "A tool for checking ansi-c programs," in *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*. Springer, 2004, pp. 168–176.

[12] D. Beyer, "Advances in automatic software verification: Sv-comp 2020," in *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*. Springer, 2020, pp. 347–367.

[13] R. C. Martin, *The Clean Coder: A Code of Conduct for Professional Programmers*. Prentice Hall, 2011.

[14] C. Eagle and J. L. Clark, "Capture-the-flag: Learning computer security under fire," NAVAL POSTGRADUATE SCHOOL MONTEREY CA, Tech. Rep., 2004.

[15] V. Ford, A. Siraj, A. Haynes, and E. Brown, "Capture the flag unplugged: An offline cyber competition," in *Proceedings of the 2017 ACM SIGCSE Technical Symposium on Computer Science Education*, 2017, pp. 225–230.