TALLINN UNIVERSITY OF TECHNOLOGY
School of Information Technologies
Department of Software Science

IVCM09/18
Ilja Šmarjov 182506IVCM

OWASP SECURE CODING PRACTICES CHECKLIST AND TRAINING: ASSESSMENT OF EFFECTIVENESS IN A TECHNOLOGY COMPANY

Master's Thesis

Supervisor: Olaf Manuel Maennel
PhD
Co-Supervisor: Kristian Kivimägi
MSc

Tallinn 2020

TALLINNA TEHNIKAÜLIKOOL
Infotehnoloogia teaduskond
Arvutisüsteemide instituut

IVCM09/18
Ilja Šmarjov 182506IVCM

OWASP TURVALISE KOODI KIRJUTAMISE KONTROLLNIMEKIRI JA KOOLITUS: TÕHUSUSE HINDAMINE TEHNOLOOGIAETTEVÕTTES

Magistritöö

Juhendaja: Olaf Manuel Maennel
PhD
Kaasjuhendaja: Kristian Kivimägi
MSc

Tallinn 2020

Author's declaration of originality

I hereby certify that I am the sole author of this thesis. All the used materials, the literature and the work of others have been referenced. This thesis has not been presented for examination anywhere else.

Author: Ilja Šmarjov

2020-05-19

Abstract

As more businesses operate online by storing and processing people's data, the topic of web application security is important. Numerous ways to improve web application security are available, but it is often challenging to measure their effectiveness.

This thesis is an exploratory case study, which aims at providing an assessment of the effectiveness of the secure coding practices implemented in the researched global technology company with operations in Estonia. The company operates a private HackerOne bug bounty program and the author used the reported vulnerabilities metrics as the dependent variables during the statistical analysis.

The results of the statistical analysis suggest that the OWASP secure coding practices checklist is an effective method in reducing all vulnerabilities metrics, while the secure coding training only significantly affects the distribution of the reported vulnerabilities. When the two secure coding measures are combined, the overall effectiveness also increases.

The main contribution of this study is the assessment of the effectiveness of the OWASP secure coding practices checklist and secure coding training in reducing the number of vulnerabilities reported in HackerOne, their average severity level and the total bounty payouts. Additionally, the author analyzes the metrics of the private HackerOne bug bounty program. For assessing the effectiveness of the secure coding practices, previous research mainly relied either on the qualitative methods or one-time penetration tests. The novelty of this study is the research method, which examines data over a longer period of time and correlates internal organizational secure development processes with vulnerabilities discovered by the white hat security researchers.

This thesis is written in English and is 63 pages long, including 5 chapters, 5 figures, and 23 tables.

Annotatsioon

Kuna üha rohkem ettevõtteid tegutseb internetis ning haldab inimeste andmeid, on veebirakenduste turvalisuse teema läbi aegade asjakohaseim. Veebirakenduste turvalisuse parandamiseks on palju võimalusi, kuid nende tõhususe mõõtmine on sageli keeruline.

See lõputöö on uurimuslik juhtumianalüüs, mille eesmärk on anda hinnang turvalise koodi kirjutamise meetmete tõhususele Eestis tegutsevas globaalses tehnoloogiaettevõttes. Ettevõttel on privaatne HackerOne'i veahaldusprogramm ja autor kasutas sõltuvate muutujatena raporteeritud turvaaukude mõõdikuid statistilises analüüsis.

Statistilise analüüsi tulemused näitavad, et OWASP turvalise koodi kirjutamise kontrollnimekiri on tõhus meetod kõigi turvaaukude mõõdikute vähendamiseks, samal ajal kui turvalise kodeerimise koolitus mõjutab märkimisväärselt ainult raporteeritud turvaaukude jaotust. Kahe turvalise kodeerimise meetme kombineerimisel suureneb ka üldine tõhusus.

Selle uuringu peamine panus on OWASP turvalise koodi kirjutamise kontrollnimekirja ja väljaõppe tõhususe hindamine HackerOne'is raporteeritud turvaaukude arvu, nende keskmise raskusastme ja väljamaksete vähendamisel. Lisaks analüüsib autor privaatse HackerOne'i veahaldusprogrammi mõõdikuid. Turvalise koodi kirjutamise meetmete tõhususe hindamisel tuginesid varasemad teadusuuringud peamiselt kvalitatiivsetele meetoditele või ühekordsetele sissetungimise testidele. Selle uuringu uudsus seisneb uurimismeetodis, mille abil uuritakse andmeid pikema aja jooksul ja korreleeritakse organisatsiooni sisemised turvalisusele suunatud arendusprotsessid raporteeritud turvaaukudega.

Lõputöö on kirjutatud inglise keeles ning sisaldab teksti 63 leheküljel, 5 peatükki, 5 joonist, 23 tabelit.

List of abbreviations and terms

BBP Bug bounty platform

CTF Capture the flag

ISO International Organization for Standardization

NIST National Institute of Standards and Technology

OWASP The Open Web Application Security Project

VRP Vulnerability rewards program

Table of Contents

List of Figures

List of Tables

1 Introduction

1.1 Motivation

Majority of businesses nowadays use web applications to perform operating activities. Users often store sensitive data in the software and therefore expect vendors to keep their data safe. Security of the product creates trust in its users. Still, data breaches are on a rise [1]. In the 17-month period between 2017 and 2019 Akamai [1] registered around 4 billion alerts, 65% of which were attacks aiming at exfiltrating data using an SQL injection [2]. This means that businesses need to improve their security measures in order to withstand the inevitable web application attacks.

In order to address security issues during software development various measures have been introduced, with a systematic secure development lifecycle being one of the most prominent changes. The secure development lifecycle addresses security at multiple stages [3, 4], instead of relying on security testing when the software is ready.

Secure coding requirements and training are parts of the secure development lifecycle [4] and their importance is highlighted by various international standards (e.g. ISO 27001 [5]). However, no specific guidelines are given about what those requirements and training should contain, and therefore it is up the businesses to decide which security measures to implement and how to train the developers on those measures. Organizations like OWASP [2] developed different secure coding practices in order to help businesses strengthen their security. The OWASP recommendations are now being widely adopted globally (e.g. the number of forks for the Cheat Sheet Series project in GitHub is over 1800 [6]).

Academic research about the use of secure coding checklists and training has been done before, however there is no consensus about the effectiveness of measures. Some authors recommend using the secure coding checklists [7], while others critique this practice [8]. The general notion regarding security training for developers is that it is useful, but measuring its effectiveness is a challenging task. Some papers relied on qualitative data [9, 10]. Even though analyzing survey results is a valid research method, the conclusions are mainly based on people's subjective opinions about the topic and not the measurable

[1] https://akamai.com
[2] https://owasp.org

vulnerabilities being introduced in the code. Other studies conducted simple penetration tests in order to measure the effectiveness of secure coding practices [11, 12]. The author sees it as problematic as well, because such an approach only captures a snapshot at a given time and does not investigate the effects of secure coding practices over a longer period of time.

1.2 Problem statement and contribution

The main goal of this study is to provide an assessment of effectiveness of the secure coding practices implemented in the researched technology company.

In order to achieve the goal, an exploratory case study is conducted on a global technology company with operations Estonia. The company implemented the cyber security training for developers in 2017 and the OWASP secure coding practices checklist [13] in 2018, however the effectiveness of the measures has never been empirically tested. This research aims at addressing this problem.

For measuring the effectiveness the author used the data gathered from the private HackerOne [1] bug bounty program operated by the researched organization. The metrics are:

- the number of submitted vulnerability reports,
- the average severity level of the submitted vulnerability reports,
- the sum of payouts made to the white hat researchers.

Both qualitative and quantitative methods were used in this research. During the data collection phase, semi-structured interviews were conducted with developers and information security team members to verify the author's data collection methods and make sure the data is valid. Quantitative data was used for the statistical analysis, and the author conducted a set of tests (Chi-square, Mann-Whitney U test, linear regression modelling) to assess the effectiveness of the secure coding measures in the researched organization.

Before starting the research, the author had multiple assumptions. First, the author expects that when the secure coding checklist is followed during the development phase,

[1] https://www.hackerone.com

all metrics are reduced. Even if the checklist is generic and not customized for the specific organizational needs, a periodic reminder to think about security when writing code should have a positive effect. Second, the author assumes that the security training for developers might be effective in raising general awareness about security, but does not directly affect the vulnerability metrics in HackerOne.

It is worth mentioning what is outside of the scope of this research. The OWASP secure coding practices checklist fulfilment is measured on a binary scale (either it is filled out or not). The author is not assessing to which extent all 202 requirements of the checklist [13] have been followed by the developers. In other words, data provided in the checklist by the developers is trusted without verification. Additionally, no correlation between the checklist requirements, secure coding training and the types of reported vulnerabilities is analyzed, although that could have provided valuable insights both for the further academic research as well as for other organizations.

The scientific contribution of this study is twofold. First, only around 7% of all vulnerability reports submitted to HackerOne bug bounty platform are public [14]. Since the author had access to a private HackerOne program, a brief analysis of the program metrics is given. Second, the author measures how effective OWASP secure coding practices checklist and the training for developers are in reducing the number of reported vulnerabilities, their severity levels and total payouts to the HackerOne participants. To the author's knowledge, no similar research has been conducted before.

The analysis of this otherwise not publicly available data can be used in other academic studies as well as by companies who are looking for tested ways of improving their web application security.

1.3 Research questions

During this study, the following questions are answered:

1. What characterizes the private HackerOne bug bounty program of the researched organization?
2. Does following the OWASP secure coding practices checklist have a connection with valid HackerOne reports? Specifically, does following the checklist help:

(a) reduce the number of vulnerabilities?

(b) reduce the average severity of the vulnerabilities?

(c) reduce the total payouts to the white hat researchers?

3. Does participating in the Rangeforce training event help developers produce more secure code? Specifically, does completing the training help:

   (a) reduce the number of vulnerabilities?

   (b) reduce the average severity of the vulnerabilities?

   (c) reduce the total payouts to the white hat researchers?

4. Do the OWASP secure coding practices checklist and Rangeforce training compliment each other or should the organization focus its efforts on either of those?

2 Theoretical background

Web application security is a broad topic, and therefore it has been well researched during the recent years. In this study, the author approaches web application security from two sides – internal organizational processes that affect the software security, and black-box security testing done through the emerging markets of bug bounty programs.

During the literature review forward and backward snowballing techniques were used [15]. First, a tentative start set was constructed by using keywords that included *secure development lifecycle, secure coding, OWASP secure coding checklist, security checklist effectiveness, secure coding training effectiveness, assessment of secure programming training, security measures effectiveness, bug bounty, crowdsourced vulnerability discovery and vulnerability reward program*. For the start set generation Google Scholar was used. This helped avoid the possible bias when only looking at specific journals [15]. In addition to Google Scholar, IEEE Xplore database was also actively utilized for enriching the initial start set.

In addition to academic papers, the following chapters also contain grey literature sources. The web application security is not only a theoretical, but a highly practical field of study, and therefore grey literature is essential for a well-rounded literature analysis.

2.1 Secure development practices in a technology company

During the last decade, the importance of security in software has been on a rise. Multiple models for secure software development lifecycle have been developed and researched [16]. Among the famous models are Secure development lifecycle by Microsoft [4], Comprehensive, lightweight application security process by OWASP [17] and Touchpoints introduced by McGraw [18].

Organizations implement different practices in order to add security throughout the software development lifecycle [3]. There are numerous measures that are being included into the security standards and policies inside the organizations [9]:

- threat modelling,
- security code reviews,

- security requirements,
- security tools,
- penetrations testing,
- trainings,
- vulnerability tracking.

The main goal of these security measures is to prevent vulnerabilities in the software. A bug, security bug and vulnerability are often used interchangeably. Ozment suggests that the correct definition for the vulnerability should be "an instance of [a mistake] in the specification, development, or configuration of software such that its execution can violate the [explicit or implicit] security policy" [19]. The important notion here is that a vulnerability is related to security and not just any error in computer systems. Vulnerabilities can be discovered at different stages of the software development lifecycle.

Morrison *et al.* conducted a survey among open-source projects focused on security and found that from a number of different software development security practices the organizations spent significantly less time on following secure coding standards and performing security reviews (which both can be considered checklists in terms of this study) as opposed to doing threat modelling and penetration testing on the application [9]. On the other hand, most of the survey participants claimed that they applied secure coding standards and tracked reported vulnerabilities daily [9]. During their study, Morrison *et al.* used linear regression analysis to identify the relationships between ease of use, effectiveness and training, and usage of different secure coding practices [9]. It was confirmed that training in each practice increases the usage [9].

The Open Web Application Security Project (OWASP) is a non-profit organization that has existed since the end of 2001 and has been helping software engineers improve their code security [1]. OWASP runs multiple open-source projects focused on different topics, with OWASP Top 10 being one of the most popular ones [20]. The top 10 is often used as a signpost security-wise [20]. The list is periodically updated by security experts to reflect the changes happening in software development and ranks most common vulnerabilities found by analyzing the publicly available vulnerabilities from over 100 000 applications [20]. If during the development engineers refer to the OWASP Top 10 list, the basic se-

[1] https://owasp.org/about

curity needs of the application should be covered [11]. However, OWASP encourages organizations to avoid only focusing on the top 10 list, as there are many more vulnerabilities not covered by the list [20]. The latest version of the ranking at the time of writing was released in 2017 and can be seen in Table 1. The next revision of the top 10 list is planned for 2020 [20].

Table 1. OWASP Top 10-2017 risk list [20].

ID	**Risk**
A1	Injection
A2	Broken authentication
A3	Sensitive data exposure
A4	XML external entities (XXE)
A5	Broken access control
A6	Security misconfiguration
A7	Cross-site scripting (XSS)
A8	Insecure deserialization
A9	Using components with known vulnerabilities
A10	Insufficient logging and monitoring

In a study among 5 Norwegian start-ups of different size and industry, it was found that in general security is a concern for organizations and all participants have either heard about or are actively using the OWASP Top 10 list [11]. Some organizations had a very good understanding of their security posture while others did not know if their applications had vulnerabilities [11]. In addition to interviews, the authors of the study also conducted penetration testing on all 5 applications with the goal of verifying if the top 10 vulnerabilities exist in the software [11]. The start-ups that had the most awareness of the OWASP Top 10 project also had fewer vulnerabilities [11].

The problem with a lack of security especially in a start-up environment is the tight competition, which leads to the need to develop faster, not more securely [21]. Also, start-ups and technology companies often practice fast-paced agile development, and following the formal secure development lifecycle is not perceived as very *agile* [21]. This idea has been supported during a survey done by Errata Security, where the biggest reason for not implementing a secure development lifecycle was that it took too much time [22]. Other

reasoning behind not having a formal security process in the development were heavy resource requirements, no knowledge of the methodologies, high cost and not needing it overall [22]. Nicolaysen *et al.* suggest that even when more strict security practices are absent from the agile development, the experience and collective knowledge of developers paired with security training should enable noticing security vulnerabilities during coding [21]. In a more recent study, after interviewing 3 organizations Cruzes *et al.* also found that in an agile setup there is an assumption of security knowledge among developers, but no standardized processes of ensuring security throughout development [7]. For practitioners, authors recommend secure coding training and using secure coding guidelines similar to what OWASP offers [7].

2.2 Secure coding checklist and training

In their paper Gasiba and Lechner mention there is still ambiguity in the secure coding guidelines (or checklists) [23]. The guidelines are either too generic or are programming language specific, although not all languages are represented [23]. If the checklist is abstract, it creates confusion in organizations, because developers do not know how to comply [23]. In the absence of a well-defined language specific secure coding guideline, authors suggest creating one by coming up with a business impact metric, calculating the metric for vulnerabilities from the CVE database, connecting the metrics and vulnerabilities to language-specific rules and finally combining the rules into a checklist [23]. When it comes to making sure that developers are aware of code security best practices, capture the flag style games are appropriate [23], however no assessment of their effectiveness is provided.

Duncan and Whittington discussed the drawbacks of using checklists in cloud computing when complying with standards [8]. According to their study, the standards cannot keep up with the rapidly changing cloud computing and an attempt to force an organization to adjust the systems to follow the complex checklist requirements can, for example, cause some trivial monitoring errors [8]. As a benefit of using checklists in computer science they bring low resource requirements (time, money, senior staff) and suitability for repetitive tasks [8]. Authors also found that the evidence to support the idea that security checklists work is lacking and note that checklists can be effective in some situations, but proper consideration needs to be given to the preparation of specific organizational needs, checklist design and implementation [8]. It should be mentioned that the research

by Duncan and Whittington focused on the use of checklists in terms of compliance with the standards (ISO, NIST, *etc*) and not necessarily for secure coding in web applications [8].

OWASP provides secure coding guidelines and checklists. A more basic one is OWASP secure coding practices checklist [13]. As specified in the accompanying documentation, the checklist is designed to help with general security requirements instead of preventing specific vulnerabilities [13]. This checklist is a good starting point for organizations implementing secure coding guidelines into their software development process [13]. The author created a descriptive table of all requirements from the checklist divided by categories (see Table 2). As can be seen, the biggest focus is on authentication and password management, session management, access control, error handling and logging. Even though this checklist is supposed to be a simple general set of secure coding guidelines, it still has over 200 separate unique requirements [13], which can be a challenge to implement in an agile organization.

Table 2. OWASP secure coding practices checklist categories (composed by the author based on [13]).

Category	**Number of checks**
Input Validation	15
Output Encoding	3
Authentication and Password Management	35
Session Management	21
Access Control	23
Error Handling And Logging	23
Data Protection	10
Communication Security	8
System configuration	16
Database Security	13
File Management	14
Memory management	9
General coding practices	12
Total unique checks	**202**

The topic of security training has been researched from different angles. Some authors

investigated the general employee awareness training [24] and existing theories [25] in behavior research around the topic. Others focused more on the cyber security awareness aspect of training, studying how to increase the developers' secure coding knowledge and skills [26].

Votipka *et al.* did a study on Dropbox developers [26]. They designed and conducted a capture the flag (CTF) competition aimed at teaching attack and defence methods for specific vulnerabilities [26]. The knowledge of participants was measured via a survey as well as quantitatively (by looking at the number of vulnerabilities in the code repository) before and six weeks after the event [26]. CTF participants tended to start approaching development from the perspective of an attacker, which means they started reviewing the security of their solutions more critically [26]. It was also found that those who participated in the training were later less confident in the security of their solutions compared to those who were never exposed to similar training [26]. Being a little under confident never hurts when it comes to security. In general capture the flag training proved to increase security awareness [26]. Trained participants were more proactive in reaching out to the security team with security issues in their code and their solutions were less frequently flagged as potential vulnerabilities during the software development [26].

For any successful hands-on cyber security training many authors agree that the preparation and specifically the design phase is crucial [27, 28]. Too simple challenges do not provide the desired educational value, whereas participants get frustrated and lose interest in too complicated or ambiguous challenges [28]. Artificially difficult CTFs are not effective [28]. So it is important to design a secure coding training considering the level of participants, organization-specific technologies and provide appropriate hints to those who are stuck [28].

Another method of teaching security to developers is using cyber range exercises [29]. The concept is not new, but the popularity of such exercises has grown in the recent years [29]. There are simpler online and on-site versions of the exercises as well as more complex versions used for cyber warfare training in the military and on the government level [29]. Cyber range training usually consists of blue teams on the defense and red teams on the offense (however, red team activity is mostly automated in online training) [29].

2.3 Bug bounty programs and their usefulness

This chapter gives insights into the nature of vulnerability rewards programs (VRP) or bug bounty platforms (BBP) and their usefulness in general.

Zhao *et al.* analyzed two bug bounty platforms – Wooyun in China and HackerOne in the U.S. Wooyun has multiple unique features not found in other platforms [14]. Firstly, white hats can report a vulnerability in any website on the internet (however, mainly Chinese websites are reported) and secondly, if the report is valid, the vulnerability will be made public in 45 days regardless if the vendor provides a patch [14]. In addition to that, for the most part, no monetary reward is provided for a valid vulnerability [14]. One important limitation of the HackerOne analysis is the fact that only around 7% of total reports are publicly disclosed [14]. This signifies that there can be a large number of private programs and the majority of the findings in the public programs are kept private. Zhao *et al.* found that it is important to increase the rewards for valid reported vulnerabilities due to the limited size of white hat community and increased competition for their attention [14]. Findings also show that the number of discovered vulnerabilities is decreasing, signalling increased security of the web applications and therefore more effort required from the white hats to find new vulnerabilities [14]. Additionally, it is suggested that participating organizations should strive to increase the pool of white hats participating in their bug bounty programs in order to ensure vulnerability discovery [14]. Although, the paper does not discuss optimal levels of incentives and number of participants.

Finifter *et al.* studied bug bounty programs by Google Chrome and Mozilla Firefox. The aim was to analyze mature platforms and give guidance to organizations looking to start or improve their programs [30]. The research was done from different angles: software vendor, white hat hacker and vendor's security researcher [30]. The conclusions for a software vendor are promising – the two researched bug bounty programs proved to be effective solutions for finding vulnerabilities and compared to hiring one security engineer cost-wise [30].

In their research, Zhao and Lazska made multiple observations. White hats are motivated by monetary rewards and try to minimize the efforts required for finding vulnerabilities [31]. This economic contradiction leads to bug bounty platforms receiving low numbers of valid reports [31]. Private HackerOne programs have a higher percentage of valid vulnerability reports, being around 50% in 2016 [31]. Authors also concluded that more

white hat researchers does not mean more valid reports [31]. The peak expected utility for the participating organization is reached between 75 and 175 white hats in the bug bounty program [31]. After 175, the number of invalid reports lowers the usefulness of the program [31]. This contradicts the findings of Zhao *et al.* [14].

Ozment tried to test if over time the number of vulnerabilities in a software decreases and used software reliability growth model [32]. He divided the researched period of time into equal intervals and projected identified vulnerabilities into a chart [32]. The initial findings did not provide meaningful insights into the vulnerability reporting trends, however, dividing the period into halves showed a clear downwards trend [32]. In addition, the time-between-failures metric, where software bugs were replaced by security vulnerabilities, was introduced [32]. Correlating time-between-failures (or rather time-between-vulnerabilities) with the researched period divided into halves, showed less vulnerabilities reported in the second half of the study [32].

Not all researchers agree that bug bounty programs are useful in the big picture. Rescorla studied the vulnerabilities introduced in different operating systems and made several controversial conclusions [33]. Firstly, the results of the study show that a software has an infinite number of vulnerabilities and there is no use for white hats to search for them [33]. Over time, the number of vulnerabilities in any given product stays relatively the same [33]. Secondly, publicly disclosing the identified and fixed vulnerabilities gives extra information to malicious actors [33].

During the research by Edmundson *et al.*, 30 software developers were hired to conduct the manual code review and find the vulnerabilities in it [34]. The application was a modified version of open-source Anchor CMS, which had around 3500 lines of code [34]. The researchers found that around 20 code reviewers are needed to find all vulnerabilities in the conducted experiment and adding more reviewers is wasteful [34]. This finding shows that diversity is key when outsourcing vulnerability discovery. It is also noteworthy that the years of experience negatively correlate to the accuracy of finding reports, which sounds counterintuitive [34]. However, more web security reviews done in the past resulted in more vulnerabilities discovered during the experiment [34].

Al-Banna *et al.* conducted a qualitative study on the fears, issues and countermeasures that organization experience regarding crowdsourcing vulnerability discovery using vulnerability reward programs [35]. They interviewed 36 key security professionals from

different organizations and summarized the findings [35]. Fears were expressed by organizations not participating in the VRPs, issues – by organizations who have experience with VRPs and countermeasures for mitigating fears and issues [35]. Typical issues are low quality submissions, high cost of processing submissions and difficulties to maintain participants [35]. Multiple countermeasures were identified [35]:

- learning from the experience;
- purchasing third-party services to conduct validation of vulnerabilities;
- limiting participants by running a private VRP and only inviting white hats with high reputation;
- selective revealing of sensitive information;
- limiting the scope of the program (e.g. domains, vulnerability types);
- adjusting rewards to keep white hats engaged.

Intuitively, limiting participants is an effective method of minimizing low quality submissions and cost associated with processing them. However, this finding is not supported by the research of Zhao *et al.*, who suggest that the number of participants needs to be increased in order to ensure steady inflow of vulnerability reports [14].

2.4 HackerOne case studies

Arkadiy Tetelman was involved in the VRP at Twitter and AirBnB. According to Tetelman, in addition to fixing the reported vulnerabilities, organizations participating in a VRP should also collect and analyze the program metrics: types of vulnerabilities reported, which parts of the application are often vulnerable, which teams introduce vulnerabilities, how quickly vulnerabilities are fixed [36]. So essentially, the organization needs to link externally reported data with internal processes in order to maximize the benefits of the VRP. The data regarding most frequent vulnerability types can help make business decisions in terms of specific tools and custom training aimed at improving engineers' knowledge and skills as well as detecting the vulnerabilities during the software development lifecycle [36]. The VRP metrics (first response time and time it takes to triage, pay out the reward and fix the vulnerability) are essential when determining the overall program health [36]. Tetelman thinks that first response time and how quickly the reward

is paid out are the most crucial metrics when maintaining relationships with white hat researchers [36]. He also suggests that an organization should start with a private program and only make it public when there is a significant number of participants, the internal processes perform well and there is a decrease in the number of incoming reports [36].

The U.S. Government also joined HackerOne in 2016 to benefit from the large pool of white hat researchers [37]. More than 1400 white hats participated in their program, 1189 vulnerabilities were found, of which 138 were previously unknown [37]. The program proved to be very cost effective (around $75000 paid to hackers) and efficient overall [37].

HackerOne did a number of case studies with their participating organizations [38–49]. Majority of the studied organizations run a public vulnerability reward program [39, 40, 42, 45], but some started with a private program first [43, 48, 49]. The main reason for starting private is to test out internal processes for vulnerabilities triage and fix [43]. When it comes to why organizations have a VRP in place, usually it is because internal code reviews and internal/external penetration tests are not scalable and often not effective either [38, 39, 41]. For example, Sumo Logic's penetration tests did not find issues, however during the first 15 days after VRP launch, 12 new valid vulnerabilities were reported [47]. Many organizations find it beneficial, when many professional hackers try to break their applications. Steve Shead, Grand Rounds VP of Infosec and IT, summarizes the advantages of using a vulnerability reward program:

> "you get the benefit of folks looking under rocks you didn't know existed, and they find issues you didn't know you should be looking for." [41]

For maintaining good relationships and high participation with white hat researchers, it is important to have fast response and bounty times, as well as high maximum rewards [38–40]. One important observation from HackerOne use cases is that multiple organizations use HackerOne triage service for initial vulnerability assessment [36, 44]. The main reason behind using this service is to save internal teams' time with filtering out duplicate and invalid submissions [38, 45]. The full summary of case study analysis can be found in Appendix 1.

3 Methodology

This chapter provides an overview about the researched organization, data sources and data collection methods (including potential issues and their mitigation). Additionally, the author discusses the threats to validity and briefly touches on ethical considerations.

3.1 Researched organization overview

The organization chosen for the research operates in a technology sector, has operations in multiple geographical locations, including Estonia, and employs around 300 software engineers. Since 2015 the organization has been running a private HackerOne bug bounty program. In addition to HackerOne, annual penetration tests are also conducted, however a bigger focus is on HackerOne.

In 2017 the researched organization started using Rangeforce [1] as the security training platform for the engineering and infrastructure departments with a heavier focus on software developers. Rangeforce provides a number of different training options, but the one used in this study is an on-site siege, where the teams need to protect small virtual networks with multiple web services against different attack vectors. This training emphasizes teamwork and knowledge sharing between the participants during the hands-on lab. Organization's main goal for using this on-site siege is to raise developers' security awareness about common vulnerabilities, which is assumed to result in developers making fewer security mistakes when writing code. The topics covered in the training are:

- Secure server configuration,
- HTTPS configuration,
- Cookie security,
- SQL injection attacks,
- Cross-site scripting attacks,
- Path traversal attacks.

[1] https://rangeforce.com

Before 2018 the researched organization had a fairly common software development model where each engineering team works on a predefined set of features and services. However, in the middle of 2018 the development model evolved into a framework similar to what Spotify [50] has been using. The main difference is that the new framework enables more internal mobility for software developers, which results in a more focused and faster creation of new product features. With the change, new terminology was introduced. Engineering teams were transformed into "tribes", which often combined multiple teams. Now engineers can choose the business goals to work on and for every new project a dedicated "mission" is formed. Missions consist of people from different departments and tribes. New features and services are only developed during missions. After a mission has finished, the feature or service ends up on a "launchpad", where a group of developers from the same tribe is tasked with maintaining the service and fixing bugs, including vulnerabilities reported via HackerOne.

Before each mission lands on a launchpad, a set of checks need to be completed. In an attempt to further ensure web application security, the OWASP secure coding practices checklist was implemented in 2018 alongside the missions framework. The checklist was used without any modifications from the original version provided by OWASP.

3.2 Research method

The data for this research has been collected from multiple sources – HackerOne, Atlassian Jira (internal ticketing system), Atlassian Confluence (internal knowledge space), a custom internal tool for missions tracking and the Rangeforce training after action reports. The overall timeframe for the research is 2015 - 2020, however the timeframe for Rangeforce data is 2017 - 2020 and for the secure coding checklist 2018 - 2020. The author had direct access to all tools and resources needed for this study.

An exploratory case study is chosen as a research method for this paper. The organization had implemented security training and a secure coding checklist with the assumption that these measures are effective in improving the security of the web application. However, these assumptions have never been empirically tested and the effectiveness of the measures has not been confirmed or refuted. The unit of analysis is a mission, because besides fixing product and security bugs on a launchpad, all development is done during the missions. The author conducted an empirical study using various quantitative statistical analysis tests. Because the study is made in a real-life context, the data might not

be perfect. The problems and their mitigations during data collection are described in 3.5. In general, the author included qualitative data collection methods (semi-structured interviews with different stakeholders) to offset the potential researcher bias and verify data collection methods.

The author analyzed whether secure coding training offered to the engineers and following the secure coding checklist are effective using the following metrics: number of valid vulnerability reports in HackerOne, average severity level of the reports and total bounty paid to the hackers.

Additionally, the author intended to analyze the effectiveness of the secure coding measures and conducted a survey among developers. However, the response rate for the survey was less than 10% of all engineers, and therefore the results cannot be extrapolated to the whole population (engineering department). For this reason the survey is omitted from this thesis.

3.3 Threats to validity

Maturation. Over time developers become more experienced. After a mission has ended, a developer could be tasked with fixing bugs as opposed to working on new projects. If HackerOne researchers find vulnerabilities in the feature that the mission implemented, the developer participates in fixing the vulnerability. Therefore, the skill and knowledge can increase, leading to making fewer mistakes in the future missions, even though the security checklist might not have been followed.

Participant selection. Even though participants have not been selected for the conducted research, developers volunteer to participate in any given mission. Such volunteering means that a particular developer is proactive and eager to learn, which, in turn, might mean lower likelihood of introducing vulnerabilities over time.

Internal validity. The requirements outlined in the OWASP secure coding practices checklist are not completely relevant to each mission. The author identified and removed such missions which were not directly related to the web application or could not have had any valid vulnerability reports in HackerOne within the researched timeframe. These missions included mobile development, website style updates (fonts, etc), proof-of-concept work, research and discovery (without any end product), features that were not released

until after 1 January 2020.

Another typical threat to internal validity is a poor choice of statistical analysis methods [51]. To mitigate this threat, the author researched various statistical methods and their proper use. The main issue with the collected dataset was non-normal distribution, and therefore nonparametric statistical tests were used in hypothesis testing [52].

Because the research has been conducted on a real company, it can be challenging to isolate the effects of explanatory variables on the dependent variables. The changes in the organizational structure, development process, new tools and frameworks could have directly or indirectly affected the security of the web application. The author acknowledges this limitation, but the scope of this study did not allow for a very detailed data collection about all possible influencing factors.

Reliability and external validity. Because this research is done on a unique organization-specific dataset, the results might not be directly comparable to other technology companies. However, the general research methods and statistical analysis used in this thesis could be applied in other academic studies. A sequential confirmatory case study [51] is required to test the findings of this thesis.

3.4 Ethical considerations

The research was conducted based on internal organizational data. The original data points included the names of employees, nicknames of HackerOne participants, internal teams/tribes, features/services and internal and external project titles. Due to this, the majority of data was anonymized and aggregated as much as possible during the analysis. The author held multiple meetings with the head of information security at the researched organization in order to confirm that only the permitted amount of internal information is included in this work.

3.5 Data collection

For this study the data was gathered from multiple separate sources and then combined to the main dataset, which was used throughout the analysis. A combination of manual and programmatic methods were used for data collection and the main dataset creation.

3.5.1 HackerOne dataset

The first dataset was collected from the private HackerOne program. The export was done for the period of 9th June 2015 - 29th December 2019 and contained the following information – id, report title, severity level (low, medium, high, critical), severity score (0 - 10), status, substate, weakness, timestamps (first report, first response, triage, close, bounty awarded), reporter, bounty amount, Jira ticket reference. Some data was missing from the original exported file, so the author first had to fill in the gaps. Every valid vulnerability report has to have a corresponding internal Jira ticket, but 45% of the valid reports did not have one. At first, the author wrote a Python script to automate searching for HackerOne report URLs in Jira and enriching the dataset. Then, the author manually searched for the remaining 10% of missing Jira tickets. Lastly, all tickets were confirmed to be linked to the correct HackerOne report. The descriptive numbers can be seen in Table 3.

Table 3. Iterations of Jira ticket search (composed by the author).

Iteration	**Performed actions**	**Resolved reports missing Jira ticket (value)**	**Resolved reports missing Jira ticket (%)**
0	original export from HackerOne	228	45.15
1	Python script to search for exact match	153	30.30
2	Python script with multiple additional conditions	48	9.50
3	manual inspection and verification	0	0

When submitting a report in HackerOne, the security researcher can choose a vulnerability type (weakness) and severity level. Over 50% of all reports had a missing severity level and 32% lacked vulnerability type. The author consulted with an internal security engineer to help with assigning the missing severity levels and weaknesses. The security engineer was involved with the HackerOne program from the beginning and had the necessary knowledge. Existing severity levels and vulnerability types were also adjusted, if the content of the reported vulnerability did not match the specified type. This step of data collection was time consuming, and even though improving the HackerOne program data quality was not related to the research questions of the study, this activity on its own was a good contribution to the researched organization.

After the dataset was complete, the author enriched it with additional information that would be used in further analysis. Based on reported weaknesses, a corresponding category from the OWASP Top 10-2017 list was added to each report. Not all weaknesses in HackerOne platform are aligned with OWASP Top 10, so the author made modifications. OWASP Top 10-2017 and corresponding HackerOne weaknesses used in this research are presented in Table 4. The mapping is not complete, meaning not all possible vulnerability types are observed in the researched organization's private program. Even though *Cross-Site Request Forgery* is a standalone vulnerability type, it was categorized as *Broken access control* because all reports of this vulnerability were about privilege escalation [20]. Another otherwise standalone type, *Server-Side Request Forgery*, in all HackerOne reports meant missing security headers, which falls under *Security misconfiguration* in this case [20]. Sometimes, under *Code Injection* white hat researchers categorized *HTML injection*, which is an example of *Cross-site scripting* [20].

Additionally, teams responsible for fixing each valid vulnerability were added to the dataset. When the HackerOne program was launched in 2015, the company was not using Jira, so bugs were fixed by telling a responsible person directly or via the internal messaging system. When a Jira ticket did not exist for a specific bug, the author looked at the service that was affected and deduced the team's name from that, because until 2018 each team worked on a specific scope of features/services.

When the missions framework was introduced, in some cases a team's name was simply changed into a tribe name. In other cases, multiple teams were combined into one tribe. The author traced all historical team name changes and mergers and only the present day tribe names were left in the dataset. This process had a set of shortcomings. First, a significant number of people moved between tribes. This means that a drop in reported vulnerabilities per tribe could be simply attributed to some low-performing members moving to another tribe. Second, one particular tribe which exists in 2020 is formed from 4 separate teams. Because of its size, it is expected that a big portion of all historical vulnerabilities is attributed to that tribe. Last, two new teams were formed in 2018 and consequently became tribes. These tribes inherited legacy features that could have newly reported vulnerabilities. Due to this fact, the author decided to attribute vulnerabilities to the teams/tribes where the vulnerability originated.

Lastly, each vulnerability report was tagged with a feature name, where the issue was found. This step was challenging, because the application consists of hundreds of mi-

Table 4. HackerOne vulnerability types mapping to OWASP Top 10-2017 (composed by the author based on [20]).

ID	OWASP Top 10-2017 risk	Corresponding observed HackerOne weakness
A1	Injection	Code Injection, Command Injection - Generic, SQL Injection
A2	Broken authentication	Improper Authentication - Generic
A3	Sensitive data exposure	Information Disclosure, Insufficiently Protected Credentials, Privacy Violation
A4	XML external entities (XXE)	XML External Entities (XXE)
A5	Broken access control	Cross-Site Request Forgery, Forced Browsing, Improper Access Control - Generic, Insecure Direct Object Reference, Privilege Escalation
A6	Security misconfiguration	Information Exposure Through an Error Message, Information Exposure Through Debug Information, Insecure Storage of Sensitive Information, Security Misconfiguration, Server-Side Request Forgery, UI Redressing
A7	Cross-site scripting (XSS)	Code Injection, Cross-site Scripting (XSS) - DOM, Cross-site Scripting (XSS) - Reflected, Cross-site Scripting (XSS) - Stored
A8	Insecure deserialization	Not observed
A9	Using components with known vulnerabilities	Not observed
A10	Insufficient logging and monitoring	Not observed

croservices and it was often not clear where exactly the vulnerability was introduced. This exercise resulted in 153 different categories after the first iteration. To make sure that features are properly connected with vulnerability reports, the author consulted with one of the senior back end developers. As a result of the conversation, the following iteration narrowed the scope to 45 features/services.

3.5.2 Missions, secure coding checklist and Rangeforce training

The second phase of data collection consisted of multiple steps. At first the author did a search in the internal Confluence for the keyword "security checklist" and identified 73 completed secure coding checklists. Out of those, 4 checklists technically existed, but were empty, so the author eliminated them from the list. If the checklist had other noticeable issues (e.g. some categories were not addressed, some requirements still had a "to do" status, everything was marked as "done"), the author contacted the person responsible for filling out the security checklist. If the responsible person confirmed that the checklist was wrongly done or everything was marked as "done" by mistake without analyzing each requirement, such checklists were eliminated from the research.

Next, the author searched the internal tool created for tracking missions and exported data regarding all completed missions before 1 January 2020. This resulted in 153 missions. The list contained: mission name, tribe, mission lead (engineer), developers participating in the mission, product manager, launch date, end date. A mission ID was added to each mission to simplify data correlation during the next steps.

To ensure the data regarding completed secure coding checklists is accurate, the author manually went through each mission's Confluence pages to verify that a completed checklist existed. It was identified that 3 checklists had different naming conventions and were missed by the initial keyword search. Because the author did not have experience with the internal missions tool before, he contacted one of the engineering managers in order to gain better knowledge of the tool and tribes' responsibilities in general. Each mission was labelled with 0 (no) or 1 (yes) based on the following criteria:

- secure coding checklist filled out,
- mission is related to a web application feature.

Because the secure coding checklist used in the organization is mostly about web application security, the author decided to only focus on missions that worked on the application. Mobile development, internal projects and custom tools were outside of scope for this research.

Additionally, the following data was added regarding Rangeforce participation:

- mission lead participated in the Rangeforce training (0/1),
- mission lead's Rangeforce training score,
- each developer's Rangeforce training score,
- date of the Rangeforce training for each person.

In terms of Rangeforce training, the author decided to separate the mission lead from the rest of the developers in a mission. Since the lead helps to make sure that all appropriate checks are done before a mission is completed, the assumption here is that the mission lead's secure coding training might have an impact on making sure that the development team considers security during writing code as well.

As some people attended the training after they finished working on a mission, the data would not be applicable for the analysis. Therefore, such Rangeforce participation was removed from the dataset.

Lastly, the specific web application feature was added for each mission. The decision about the feature was based on the mission scope documented in Confluence. After that, the author sorted missions data first by feature and then by mission end date in the ascending order. For the purpose of correlating data with HackerOne vulnerability reports, the author did the same sorting in the HackerOne spreadsheet.

When HackerOne and missions spreadsheets were completed, it allowed to cross-reference each mission with the reported vulnerabilities. The process of cross-referencing included manually working through the list of missions, looking for the resolved HackerOne reports for the specific feature and tagging the HackerOne reports with the corresponding mission ID. The methodology of assigning a vulnerability report to a mission is shown on Figure 1.

Without having underlying knowledge of the application source code, the scope of each mission and the overall architecture of the application, providing highly reliable cross-reference is challenging. In order to minimize the risk of poor data collection and, consequently, unreliable research results, the author collaborated with the mission leads and the information security team. Semi-structured interviews with 11 mission leads were held to discuss attribution of specific vulnerabilities to missions they were in charge of. This approach helped validate the method the author used for connecting HackerOne vulnerability reports to the features developed during missions and increase the reliability

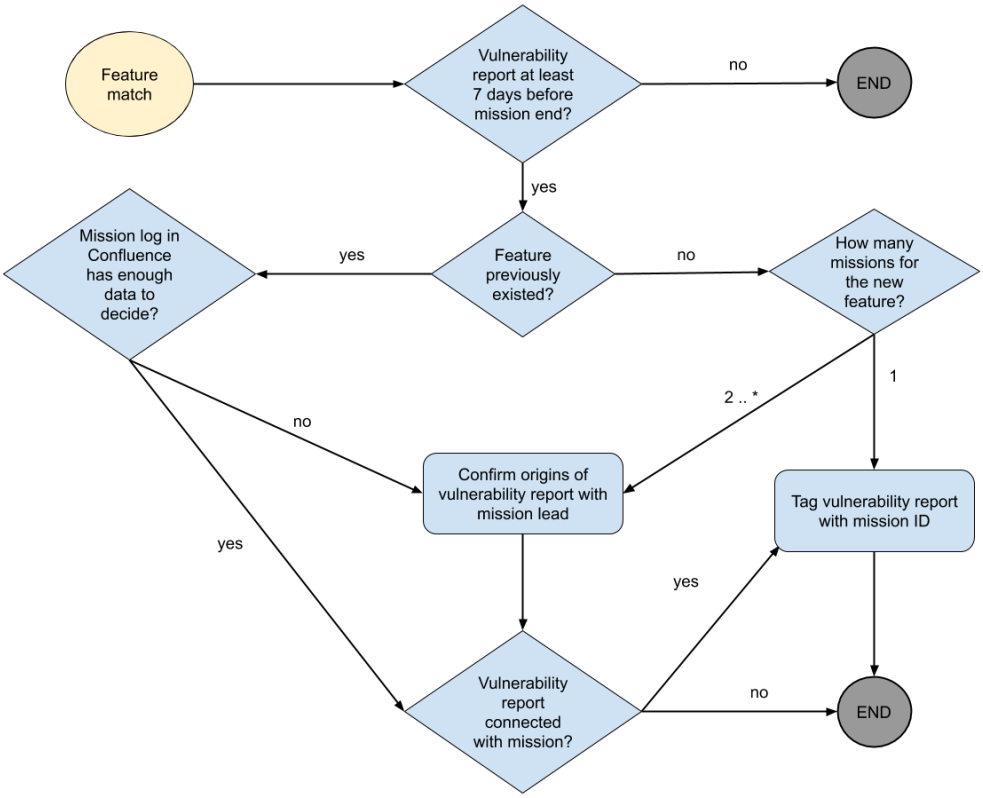

Figure 1. Process of vulnerability attribution to missions (composed by the author).

of the research results. It was discovered that some vulnerabilities were introduced at early stages of development during the proof-of-concept missions. This finding motivated the author to perform one more iteration of investigating each mission to make sure the vulnerabilities were properly attributed. Even though during this step of data collection multiple actions have been taken to decrease the chance of false positives and false negatives when attributing reported vulnerabilities to the internal development missions, it is still possible that errors occurred.

The method of data collection described above resulted in a spreadsheet containing the following columns:

- Mission ID,
- Tribe,
- OWASP secure coding practices checklist (0/1),

- Mission lead participated in Rangeforce training (0/1),
- Mission lead's Rangeforce training score,
- Number of developers total,
- Number of developers with Rangeforce training,
- Number of developers without Rangeforce training,
- Rangeforce training participation ratio,
- Average Rangeforce training score,
- Number of valid reported vulnerabilities,
- Total bounties paid,
- Average severity of valid reported vulnerabilities (1 - 4) [1].

The number of vulnerabilities, total of bounties paid and average severity were calculated using formulas based on the mission ID in the HackerOne vulnerabilities spreadsheet.

The spreadsheet was used in hypothesis testing to answer research questions. The author acknowledges that the Rangeforce-related variables are most likely highly correlated, but keeps them purposefully in the dataset for further analysis.

[1] severity levels were transformed by giving numeric values: low = 1, medium = 2, high = 3, critical = 4

4 Data analysis

4.1 HackerOne program descriptive statistics

In this chapter the author provides descriptive statistics of the private HackerOne program. In total, 1019 vulnerability reports were received between the middle of 2015 and the end of 2019 (see Table 5). Out of those, half were valid reports that got triaged and resolved by the organization. Duplicate reports contributed to a third of all submissions. Such a big number can be explained by the fact that none of the submitted vulnerabilities are made public, and therefore the white hat researchers cannot know what has been reported before.

Table 5. Descriptive statistics of the HackerOne program of the researched organization for the period of 2015-2020 (composed by the author).

	Absolute values	**Percentages**
Total number of submitted reports	1019	100%
- resolved reports	522	51.23%
- duplicate reports	341	33.46%
- informative reports	110	10.79%
- not-applicable reports	46	4.51%
Resolved reports	522	100%
- web application	453	86.78%
- public infrastructure	24	4.60%
- main website/blog	21	4.02%
- internal systems	14	2.68%
- mobile applications	5	0.96%
- outsourced development	5	0.96%

When looking at the breakdown of all valid resolved reports (see Table 5), the vast majority belongs to the web application, whereas less than 1% of valid vulnerabilities were found in the iOS or Android applications combined. There can be multiple explanations to this finding. First, the bounty range for mobile vulnerabilities is between $50 and $600 USD, so the white hat researchers might not be financially motivated. Second, mo-

bile platforms require different knowledge and skills compared to web applications. Since this particular bug bounty program is private and the number of participants is much lower than in the public ones, it might have just happened that not enough mobile experts have been invited.

During the studied period the organization paid $366100 USD in bounties to the white hat researchers. Figure 2 shows a monthly cumulative bounty payout on a logarithmic scale, which is suitable for depicting growth trends. As can be seen, there was a rapid growth in total bounties in the first months of starting the private program, followed by a slow-down of around 6 months. Starting from Q2 of 2016, rapid growth repeated and slowed down in the first half of 2017. The second growth period is explained by the fact that in April 2016 HackerOne implemented a feature for automatic invitations to private programs [53]. This change brought more researchers and as a result, more vulnerabilities were reported.

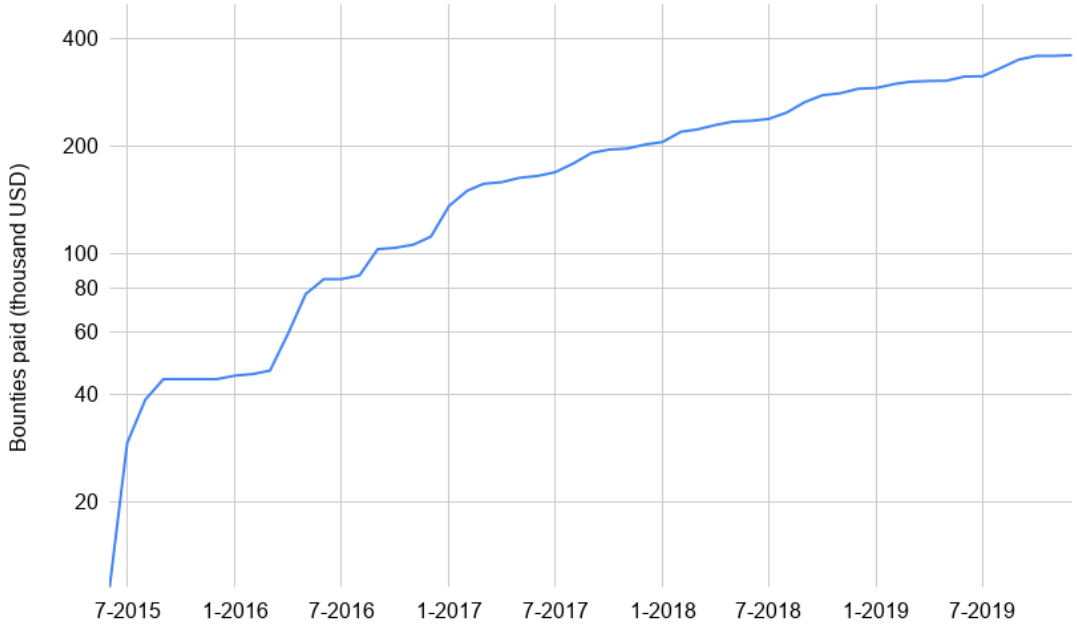

Figure 2. Cumulative total bounties paid in HackerOne (composed by the author).

When looking at the breakdown of resolved vulnerability reports by OWASP Top 10-2017 risk (see Table 6), it is clear that broken access control, cross-site scripting and security misconfiguration form the majority of the reports in terms of the number of submissions and the total bounty paid out. However, if we consider the severity of the vulnerabilities, XML external entities, injection and broken authentication are above average. Higher

severity leads to higher average bounty in those categories as well. It can be hypothesized that higher severity also means higher complexity of the vulnerabilities. In this sense, seeing lower complexity vulnerabilities in the top is logical, because generally bug bounty program participants aim to maximize the returns with less effort [31].

Another noteworthy observation regarding the types of vulnerabilities reported in the private HackerOne program is that different forms of injections, the most common risk according to OWASP Top 10-2017 list (see Table 1), only accounts to 1.5% of all valid reports (see Table 6). Generally, the top 4 risks from OWASP Top 10-2017 contribute to below 12% of the total submissions from white hat researchers. This can be explained by either a good level of security in those categories or lacking knowledge and skills of the bug bounty participants.

Table 6. Valid reported vulnerabilities in HackerOne divided by OWASP top 10-2017 risk for the period of 2015-2020 (composed by the author).

OWASP Top 10-2017 risk	**Number of resolved vulnerabilities**	**Percentage of total vulnerabilities**	**Total bounty**	**Average bounty**	**Average severity**
Broken access control	193	36.97%	150550	780.05	2.12
Cross-site scripting (XSS)	162	31.03%	78450	484.26	2.17
Security misconfiguration	106	20.31%	62750	591.98	2.20
Broken authentication	30	5.75%	39550	1318.33	2.54
Sensitive data exposure	21	4.02%	18300	871.43	2.14
Injection	8	1.53%	8500	1062.50	2.50
XML external entities (XXE)	2	0.38%	8000	4000	4.00
TOTALS	**522**	**100%**	**366100**	**701.34**	**2.19**

The author has visualized the cumulative number of resolved vulnerabilities by different OWASP Top 10-2017 risks on a logarithmic scale in the following figure (see Figure 3). It can be observed that cross-site scripting was the main focus of white hat researchers when the private bug bounty program first started. Generally, the comparably rapid growth in the cumulative reported vulnerabilities across all categories happened until the middle of 2017. The slower growth phase can be explained by either a more secure application over time, which would be supported by previous research [32], or by the fact that a roughly fixed pool of white hat researchers might be lacking diversity [34]. For example,

if the majority of participants are using a method of looking at specific types of low severity vulnerabilities in a large number of bug bounty programs, the more complex critical vulnerabilities will not be discovered.

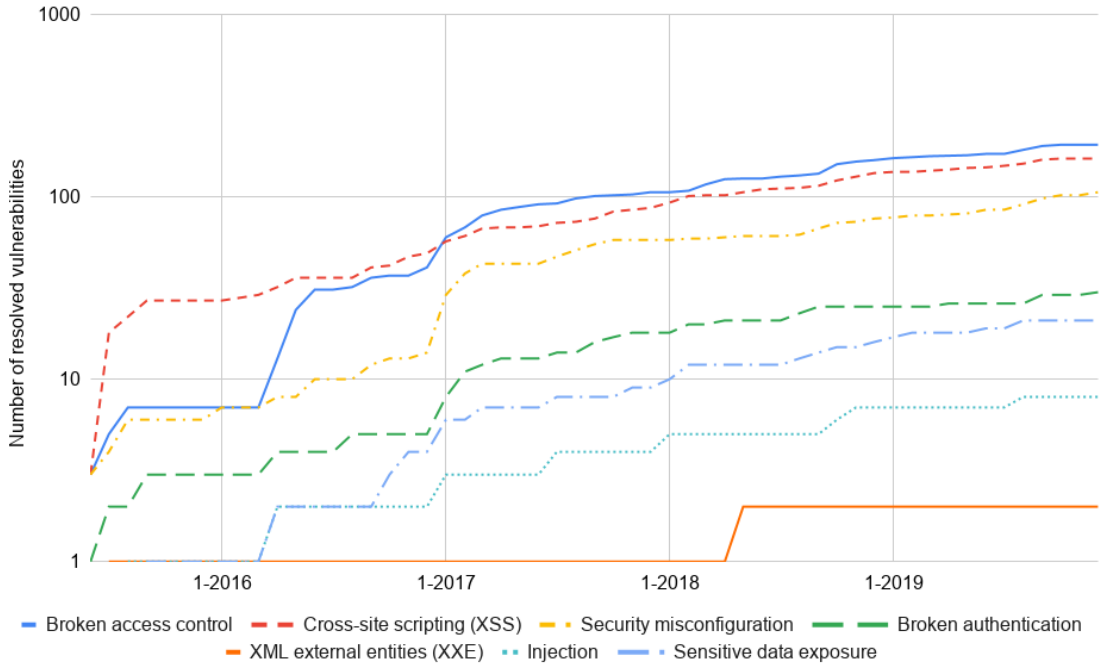

Figure 3. Cumulative number of resolved vulnerabilities divided by OWASP Top 10-2017 risk in the researched organization's HackerOne program (composed by the author).

When analyzing the severity levels of the valid vulnerability reports in HackerOne, the author used a 6-month moving average for each severity level. The moving average helps to smooth the spikes in the monthly data and allows to see a trend. A stacked column chart (see Figure 4) shows a monthly ratio between the four severity levels. From the chart it can be seen that at the start of the HackerOne program the majority of the reported vulnerabilities were considered to be high and medium severity with a period of October 2015 - March 2016 being only high severity reports. This data in combination with data from Figure 3 gives insights into the way how the severity levels of different vulnerability types have been assessed over time. It appears that in the beginning of the researched organization's HackerOne program in 2015, the web application had lots of cross-site scripting and broken access control vulnerabilities that, if exploited by malicious actors, could have caused high impact.

Over time, the proportion of high severity reports decreased significantly, whereas low and medium severity have been trending up. It can be also observed that from the beginning

of 2019 the 6-month moving average ratio of the low and critical severity levels has been in an upward trend. Considering the changes in the secure coding practices that happened

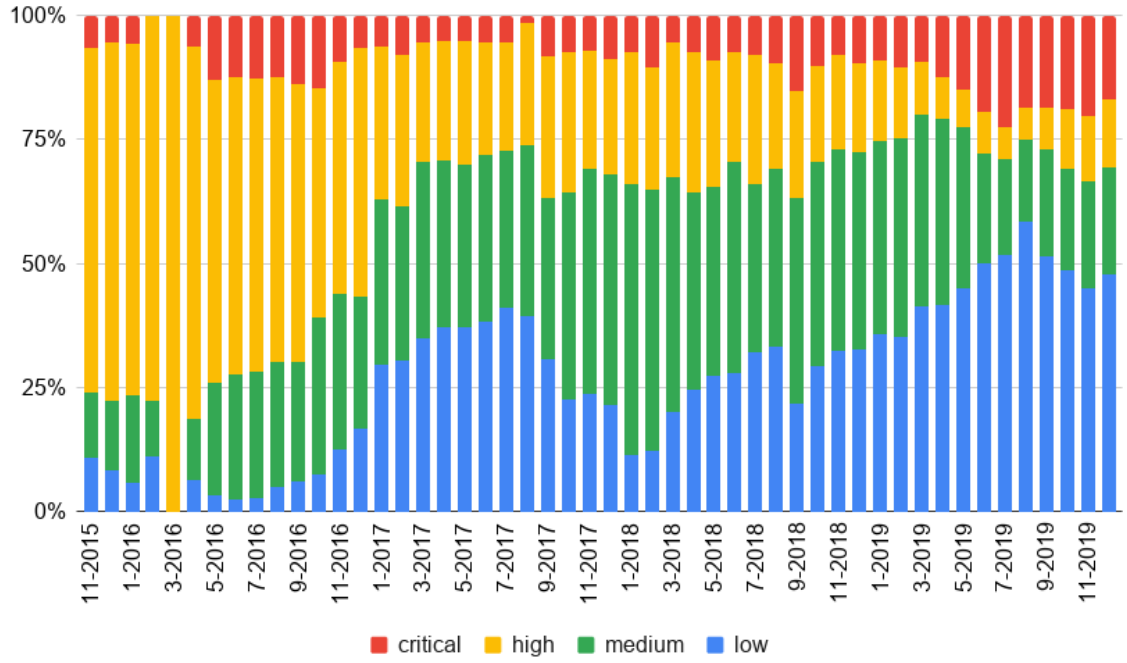

Figure 4. Monthly ratio of 6-month moving average of severity levels (composed by the author).

at the end of 2017 and the middle of 2018 (implementing the Rangeforce training and secure coding checklist), this observation might be hinting that the changes had an effect on the severity levels of the reported vulnerabilities.

4.2 Missions descriptive statistics

In this chapter the author presents an overview of the main dataset used for answering the research questions. Descriptive statistics, normality tests and correlation analysis are performed on the dataset in the preparation for the further data analysis.

The table of all variables from the dataset is shown below (see Table 7). In total there are 7 independent and 3 dependent variables. Independent variables can be divided into 2 groups – Rangeforce training and OWASP secure coding practices checklist.

Out of 153 missions that were completed before 1 January 2020, 78 were directly related to the web application. The rest of the missions worked on the features and tools, which either were not implemented into the web application during the researched timeframe

Table 7. Variables in the missions dataset (composed by the author).

Variable	**Description**	**Type**
DEVS_NO_RF	Number of developers who did not participate in Rangeforce training	Independent
DEVS_DID_RF	Number of developers who participated in Rangeforce training	Independent
RF_RATIO	The ratio of Rangeforce participation	Independent
AVG_RF_SCORE	Average Rangeforce score	Independent
ML_DID_RF	Mission lead participated in Rangeforce training?	Independent
ML_RF_SCORE	Mission lead's Rangeforce score	Independent
CHECKLIST	OWASP secure coding practices checklist filled?	Independent
NUM_BUGS	Total number of resolved vulnerabilities	Dependent
AVG_SEVERITY	Average severity level of vulnerabilities	Dependent
BOUNTY	Total amount of money paid out	Dependent

or were not available to the public (e.g. internal tools). Only the missions related to the web application were used in the further analysis. The author understands the importance of security in any development. However, since the research is done on the effectiveness of the secure coding checklist and security training, which both mainly relate to the web applications, the author decided to narrow the scope to only the web application missions.

Out of 78 missions, the breakdown of checklist and mission lead's participation in Rangeforce training are similar (see Table 8). The number of missions with identified vulnerabilities makes up 32% of all web application related projects.

Table 9 shows the descriptive statistics for the variables excluding categorical ones (ML_DID_RF and CHECKLIST). The statistics were collected for the whole dataset without dividing into categories. As can be seen, each variable has the minimum of 0. In case of dependent variables, this means that no vulnerabilities were reported for a specific mission. In case of Rangeforce scores, 0 means that there was no participation in the training.

On average, 47% of developers in each mission had participated in the Rangeforce train-

Table 8. General statistics of the missions dataset (composed by the author).

Mission indicator	**Value**	**Percentage of total**
Missions total	78	100%
Checklist filled	45	57.69%
Checklist not filled	33	42.31%
Mission lead participated in Rangeforce	50	64.10%
Mission lead did not participate in Rangeforce	28	35.90%
Vulnerabilities found	25	32.05%
Vulnerabilities not found	53	67.95%

ing prior to starting the mission (see Table 9). The Rangeforce training scores were much higher among mission leads as opposed to the whole mission team. The dataset had some outliers, which can be particularly observed in the paid out bounties, where the mean was 653 and maximum sum of bounties of 16900 for one mission.

Table 9. Descriptive statistics of the missions dataset (composed by the author).

	Mean	**Standard deviation**	**Minimum**	**Maximum**
DEVS_NO_RF	2.19	1.478	0	7
DEVS_DID_RF	1.86	1.384	0	8
RF_RATIO	0.47	0.276	0	1
AVG_RF_SCORE	1962.62	1217.788	0	5063.67
ML_RF_SCORE	4525.5	1351.522	0	6386
NUM_BUGS	0.7	1.389	0	8
AVG_SEVERITY	0.68	1.137	0	4
BOUNTY	653.21	2308.206	0	16900

One of the main deciding factors when choosing a correct analysis method is whether data is normally distributed [52]. While preparing data for analysis, the author conducted the Shapiro-Wilk normality test using *JASP* open-source software [1]. The null-hypothesis

[1] https://jasp-stats.org

was that the data is normally distributed and the confidence level was set at 95% (alpha = 0.05). As can be seen from Table 10, only the average Rangeforce score can be considered normally distributed (p-value = 0.06), and the rest of the researched data are not normally distributed (p-value < alpha). Based on this, for data analysis the author decided to utilize non-parametric tests, because they do not assume normal distribution of the sample data [52].

Table 10. Tests for normality in the missions dataset (composed by the author).

	Shapiro-Wilk W	**p-value**
DEVS_NO_RF	0.909	< 0.001
DEVS_DID_RF	0.845	< 0.001
RF_RATIO	0.944	0.002
AVG_RF_SCORE	0.970	0.060
ML_RF_SCORE	0.947	0.025
NUM_BUGS	0.569	< 0.001
AVG_SEVERITY	0.659	< 0.001
BOUNTY	0.310	< 0.001

In addition to checking for normal distribution, the author performs correlation tests to identify potential relations in the dataset. If any of the independent variables are strongly positively correlated, then their effect on the dependent variables will be similar, making it hard to distinguish the separate effect of each variable. Kendall's Tau correlation test was used because it does not assume normal distribution of data [54]. The null-hypothesis was that the data is not correlated and alternative hypothesis – data is correlated. The author wanted to see the general correlation, and therefore a two-tailed version of the test was performed.

The results of the two-tailed Kendall's Tau correlation test can be seen in Figure 5. Darker color on the gradient scale represents stronger correlation. Each cell in the matrix contains Kendall's Tau B value, followed by a number of asterisks in case the significant correlation exists. The significance level is 95%. As expected, there is a very strong correlation between all dependent variables and between most of the Rangeforce related independent variables.

When looking at the connection between dependent and independent variables, the se-

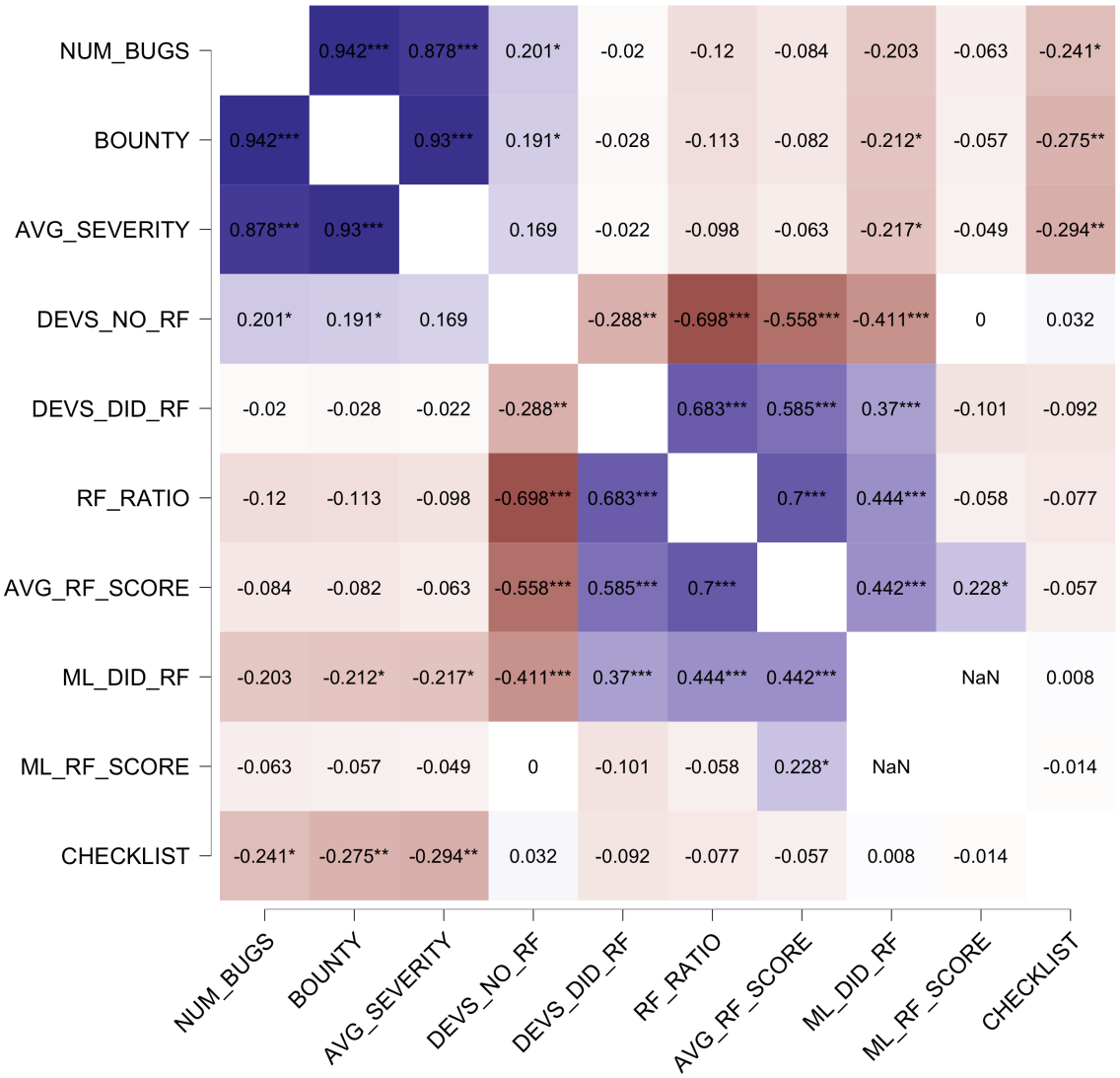

Figure 5. Kendall's Tau correlation matrix for missions dataset variables (composed by the author).

cure coding checklist is significantly negatively correlated with number of vulnerabilities, average severity and total bounties paid. The number of developers who did not get Rangeforce training is positively correlated with the number of vulnerabilities and total bounties paid. A separate categorical variable which checks if the mission lead participated in a Rangeforce training is negatively correlated with the average severity level of vulnerabilities and total bounties paid. None of the Rangeforce training score variables are significantly correlated with the dependent variables. This means the data supports an idea that how well a developer performs during the on-site siege training compared to others is not as important as simply participating and being exposed to web application

security and different common vulnerabilities.

Another observation from Kendall's Tau correlation matrix is that there is no significant connection between Rangeforce training and following the secure coding checklist requirements. This finding contradicts previous research [9] and might be related to the errors in data collection.

4.3 Answering research questions

Chi-square test is used to help address the research questions whether following the OWASP secure coding practices checklist and going through the Rangeforce training have a connection with valid HackerOne reports. The test analyzes the difference between frequencies of the dependent variable and is appropriate for categorical data [55]. To perform the test, the author created a 2x2 matrix where rows contain categorical data (groups) and columns – dependent variables.

The hypotheses are set as follows:

1. Does following the OWASP secure coding practices checklist have a connection with valid HackerOne reports?
   (a) $H0_{CV}$: there is no significant difference in the frequency distribution of vulnerabilities between the missions that filled out the secure coding checklist and those that did not.
   (b) $H1_{CV}$: there is a significant difference in the frequency distribution of vulnerabilities between the missions that filled out the secure coding checklist and those that did not.
2. Does mission lead's participation in the Rangeforce training have a connection with valid HackerOne reports?
   (a) $H0_{MLV}$: there is no significant difference in the frequency distribution of vulnerabilities between the missions depending on whether the mission lead participated in the Rangeforce training or not.
   (b) $H1_{MLV}$: there is a significant difference in the frequency distribution of vulnerabilities between the missions depending on whether the mission lead participated in the Rangeforce training or not.

To address the first question the author created a frequency distribution of vulnerabilities, which can be found in Table 11. The missions that followed the checklist requirements introduced vulnerabilities less frequently than when the checklist was not followed.

Table 11. Observed frequency matrix between checklist and vulnerabilities variables (composed by the author).

	0 vulnerabilities	**1+ vulnerabilities**
checklist+	35	10
checklist-	18	15

To test whether the difference in frequencies is statistically significant, a Chi-square test was conducted. As seen in Table 12, the p-value is below the significance level (p = 0.0298). This means that in terms of vulnerability distribution, there is a statistical significance between the missions that followed the OWASP secure coding practices checklist and those that did not. So we can reject the null-hypothesis $H0_{CV}$ and accept the alternative hypothesis $H1_{CV}$. However, making conclusions solely based on the p-value is not enough, so the author also checks the effect size for the variables [56]. Cramer's V is commonly used when describing the strength of association in Chi-square test [55]. Additionally, odds ratio is calculated for more intuitive interpretation of the test results [57].

Table 12. Chi-square test statistics for checklist and vulnerabilities variables (composed by the author).

Significance level	0.05
df	1
χ^2	4.718
Critical value	3.841
p-value	**0.0298**
Cramer's V	**0.246**

Cramer's V value is 0.246 for this particular test. Interpreting this value is not straightforward, as there is no one standard for the effect size scale. Some sources state that Cramer's V values between 0.15 and 0.25 signal strong and values over 0.25 – very strong association between the variables [54]. Others specify values between 0.20 and 0.30 to be moderate, and above 0.30 to be strong association [58]. In this paper the author considers 0.246 to mean moderate association between the checklist variable and vulnerability

distribution.

Additionally, the author calculated the odds of receiving at least 1 valid vulnerability report through HackerOne depending on whether the secure coding checklist was filled out during the mission. The results of odds ratio are included in Table 13. The ratio means that the odds of receiving a valid vulnerability report is 2.92 times higher if the secure coding checklist was not followed during the mission. Even though the strength of association is not strong, almost 3 times higher likelihood of introducing a vulnerability is an important finding. Over a large number of missions, simply following the requirements of the checklist can save an organization resources on triaging, fixing and paying for the vulnerabilities.

Table 13. The odds ratio for introducing at least 1 vulnerability depending on security checklist variable (composed by the author).

	1+ vulnerabilities
checklist-	0.83
checklist+	0.29
odds ratio	**2.92**

The author conducted another Chi-square test to address the second question. The results suggest that the variables are dependent (p = 0.0417), which means that receiving 0 or at least 1 valid vulnerability report in HackerOne is affected by a mission lead's participation in the Rangeforce training (see Tables 14 and 15). We can reject the null-hypothesis $H0_{MLV}$ and accept the alternative hypothesis $H1_{MLV}$.

Table 14. Observed frequency matrix between mission lead's Rangeforce participation and vulnerabilities variables (composed by the author).

	0 vulnerabilities	**1+ vulnerabilities**
ML_RF+	38	12
ML_RF-	15	13

Cramer's V value is 0.231, and therefore the strength of association can be considered moderate. According to the odds ratio, the odds of receiving a valid vulnerability report is 2.74 times higher if the mission lead did not participate in a Rangeforce siege (see Table 15).

Table 15. Chi-square test statistics for mission lead's Rangeforce participation and vulnerabilities variables (composed by the author).

Significance level	0.05
df	1
χ^2	4.146
Critical value	3.841
p-value	**0.0417**
Cramer's V	**0.231**
Odds ratio	**2.74**

In order to test whether there is a difference in the distribution of total bounty payouts and average vulnerability severity between the missions that followed the OWASP secure coding practices checklist and those who did not, the author conducted a Mann-Whitney U test. It is a non-parametric test that is suitable for small sample sizes [59]. It works by transforming values into their ranks and making calculations using the ranks [59]. This helps eliminate the effects of outliers in the dataset [59].

The hypotheses are set as follows:

1. Does following the OWASP secure coding practices checklist have a connection with the total payouts to the white hat researchers for the reported vulnerabilities?
   (a) $H0_{CB}$: there is no significant difference in the total payouts to the white hat researchers for the reported vulnerabilities between the missions that filled out the secure coding checklist and those that did not.
   (b) $H1_{CB}$: there is a significant difference in the total payouts to the white hat researchers for the reported vulnerabilities between the missions that filled out the secure coding checklist and those that did not.
2. Does following the OWASP secure coding practices checklist have a connection with the average severity level of the reported vulnerabilities?
   (a) $H0_{CS}$: there is no significant difference in the average severity level of the reported vulnerabilities between the missions that filled out the secure coding checklist and those that did not.
   (b) $H1_{CS}$: there is a significant difference in the average severity level of the re-

ported vulnerabilities between the missions that filled out the secure coding checklist and those that did not.

The initial dataset was modified for this test. Out of 78 missions, only 25 resulted in at least 1 reported vulnerability, and for this reason, the author removed the zeros from the dependent variable set. Logically, 0 vulnerability reports is a positive result, however, when conducting rank sum tests, an overwhelming number of zeros disproportionally inflates the sum of ranks and therefore significantly affects the results.

To conduct the test, the data is sorted in the ascending order and each data point is given a rank [59]. In case of ties, an average rank is assigned to each data point [59]. When the ranks are summed, the U statistics are calculated using the Equation 1, where n_x and n_y are sample sizes and R_x is the sum of ranks for sample x [59].

$$U_x = n_x * n_y + \frac{n_x * (n_x + 1)}{2} - R_x \tag{1}$$

Equation 1. Mann-Whitney U statistic calculation formula [59].

After calculating the U_x and U_y, the smaller U statistic is compared to a value from the table of critical values. If the U statistic is smaller than the critical value, null-hypothesis is rejected and an alternative hypothesis is accepted.

The total bounty and average severity ranks have been sorted in the ascending order and are presented in Table 16. *Checklist-* and *checklist+* are corresponding to missing or filled out secure coding checklists in a mission. It can already be observed that in both cases the ranks are higher within the groups which did not follow the checklist.

The results of Mann-Whitney U test can be seen in Table 17. At 95% confidence level the calculated U_1 in both tests is smaller than the critical U of 39 (p = 0.013 for bounty and p = 0.001 for severity). This means the data supports the idea that not following the secure coding checklist during the mission results in significantly different amounts of money paid out to the security researchers at HackerOne and the average severity of reported vulnerabilities. We can reject both null-hypotheses $H0_{CB}$ and $H0_{CS}$, and accept the alternative hypotheses $H1_{CB}$ and $H1_{CS}$.

The effect size of the relationship is moderate (r = 0.446) for the total bounty and strong (r = 0.606) for the average severity (see Table 17). Because the rank sums for the group

Table 16. Ranks distribution for bounty and average severity dependent variables split between checklist groups (composed by the author).

Total bounty ranks		**Average severity ranks**	
checklist-	**checklist+**	**checklist-**	**checklist+**
2.5	2.5	3	3
8.5	2.5	10	3
8.5	2.5	13.5	3
8.5	5	13.5	3
8.5	6	13.5	6.5
12.5	11	13.5	6.5
14	12.5	13.5	8.5
18	15	17	8.5
19	16	18	13.5
20	17	19	20.5
21		21	
22		22	
23		23	
24		24.5	
25		24.5	

that did not follow the checklist is much higher in both tests, we can conclude that the researched organization pays hackers significantly more for those missions and the reported vulnerabilities are of significantly higher severity.

The strong relationship between following the OWASP secure coding practices checklist and lower severity of reported vulnerabilities is an important finding. This means that even though the checklist cannot completely eliminate the security mistakes made during the missions, it can lower their severity and therefore the potential impact on customers' data.

A Mann-Whitney U test is also performed to analyze if there is a meaningful difference in the total bounty and average severity of reported vulnerabilities depending on whether the mission lead participated in a Rangeforce training.

Table 17. Mann-Whitney U test results for bounty and average severity dependent variables split between checklist groups (composed by the author).

	Total bounty	**Average severity**
Checklist- rank sum	235	249
Checklist+ rank sum	90	76
n_1	15	15
n_2	10	10
$\boldsymbol{U_1}$	**35**	**21**
U_2	115	129
Critical U at $n_1 = 15$ and $n_2 = 10$	39	39
Mean	75	75
Standard deviation (corrected for ties)	17.955	17.822
z-score	-2.228	-3.030
p-value	**0.013**	**0.001**
r	**0.446**	**0.606**

The author set the following hypotheses:

1. Does mission lead's participation in the Rangeforce training have a connection with the total payouts to the white hat researchers for the reported vulnerabilities?
   (a) $H0_{MLB}$: there is no significant difference in the total payouts to the white hat researchers for the reported vulnerabilities between the missions depending on whether the mission lead participated in the Rangeforce training or not.
   (b) $H1_{MLB}$: there is a significant difference in the total payouts to the white hat researchers for the reported vulnerabilities between the missions depending on whether the mission lead participated in the Rangeforce training or not.
2. Does mission lead's participation in the Rangeforce training have a connection with the average severity level of the reported vulnerabilities?
   (a) $H0_{MLS}$: there is no significant difference in the average severity level of the reported vulnerabilities between the missions depending on whether the mission

lead participated in the Rangeforce training or not.

(b) $H1_{MLS}$: there is a significant difference in the average severity level of the reported vulnerabilities between the missions depending on whether the mission lead participated in the Rangeforce training or not.

The ranks in the ascending order are presented in Table 18 below.

Table 18. Ranks distribution for bounty and average severity dependent variables split between mission lead Rangeforce participation groups (composed by the author).

Total bounty ranks		**Average severity ranks**	
ML_RF-	**ML_RF+**	**ML_RF-**	**ML_RF+**
2.5	2.5	3	3
5	2.5	3	3
6	2.5	6.5	3
8.5	8.5	6.5	8.5
8.5	8.5	8.5	10
12.5	11	13.5	13.5
14	12.5	13.5	13.5
15	17	13.5	13.5
16	18	18	17
19	22	19	20.5
20	24	20.5	22
21	25	24.5	23
23		24.5	

In both tests the calculated U statistic is greater than the critical value ($U_1 > 41$) at 95% significance level (see Table 19). This leads us to conclude that the mission lead's participation in the Rangeforce training prior to leading a mission has no association with the amount of money that would be paid to HackerOne participants and the average severity of reported vulnerabilities. We can accept both null-hypotheses $H0_{MLB}$ and $H0_{MLS}$, and reject the alternative hypotheses $H1_{MLB}$ and $H1_{MLS}$.

For the last phase of the data analysis, the author conducted a set of tests to assess the combined effects of the OWASP secure coding practices checklist and developers par-

Table 19. Mann-Whitney U test results for bounty and average severity dependent variables split between mission lead Rangeforce participation groups (composed by the author).

	Total bounty	**Average severity**
ML_RF- rank sum	235	249
ML_RF+ rank sum	90	76
n_1	13	13
n_2	12	12
$\boldsymbol{U_1}$	**76**	**72.5**
U_2	80	83.5
Critical U at $n_1 = 13$ and $n_2 = 12$	41	41

ticipating in the Rangeforce training on the number of reported vulnerabilities and their average severity. The author is not analyzing the effect on the total bounty paid, because previous tests showed that generally BOUNTY dependent variable is correlated with the number of vulnerabilities and their severity, and follows the same patterns.

JASP was used to perform linear regression analysis. The stepwise linear regression modelling was used in all following tests and only the coefficients with confidence index above 95% ($p < 0.05$) were accepted into the final models. All tests contain a null-model H_0 and an alternative model H_1. All null-models included only the CHECKLIST independent variable, because it showed the biggest descriptive power in the Chi-square and Mann-Whitney U tests.

The first stepwise linear regression modelled the relationship between independent variables and the number of reported vulnerabilities. During the regression the only significant independent variables included in the final model were checklist, number of developers who participated in Rangeforce training and the ratio of Rangeforce participation.

Only the OWASP secure coding practices checklist as a predictor explains 5.9% of the variances in the number of reported vulnerabilities for a given mission and is statistically significant ($p = 0.019$) (see Table 20). However, when in addition to following the checklist team members have also participated in the Rangeforce training, the explanatory power of the model increases to 22%. The model one is highly significant ($p < 0.001$).

Table 20. Model summary - number of vulnerabilities (composed by the author).

Model	R	R^2	Adjusted R^2	RMSE	p
H_0	0.266	0.071	0.059	1.348	0.019
H_1	0.500	0.250	0.220	1.227	< 0.001

The unstandardized coefficients in the alternative model H_1 (see Table 21) show that when a secure coding checklist is followed, the number of vulnerabilities is reduced by 0.63. The assumption that the number of developers who had the Rangeforce training decreases the number of vulnerabilities is not supported by this model. Each additional developer on a mission, who had Rangeforce training increases the number of security bugs by 0.582. However, the higher the ratio of developers with Rangeforce training, the fewer vulnerabilities are introduced. The meaning of this finding is controversial, but could be explained intuitively by the fact that the more people work on a project, the more code is produced, and therefore more potential vulnerabilities can arise regardless of how well people are trained security-wise.

Table 21. Linear regression model coefficients for the number of reported vulnerabilities in HackerOne (composed by the author).

Model		Unstandardized	Standard Error	Standardized	t	p
H_0	(Intercept)	1.121	0.235		4.778	< 0.001
	CHECKLIST	-0.743	0.309	-0.266	-2.406	0.019
H_1	(Intercept)	1.209	0.332		3.642	< 0.001
	CHECKLIST	-0.634	0.284	-0.227	-2.234	0.028
	DEVS_DID_RF	0.582	0.144	0.580	4.050	< 0.001
	RF_RATIO	-2.622	0.717	-0.521	-3.657	< 0.001

The author also analyzed the effects of independent variables on the average severity of the reported vulnerabilities using linear regression. Null-model H_0 had the secure coding checklist as the independent variable and the alternative model H_1 had one additional variable – number of developers without the Rangeforce training. During the stepwise regression modelling all other independent variables were not significant enough to increase the explanatory power of the null-model. As can be seen in Table 22, both models are highly significant. Just following the checklist explains 12,6% of the variance in the average severity of the reported vulnerabilities. When we add the number of people with-

out the security training, the explanatory power increases to 16,4%.

Table 22. Model summary - average severity (composed by the author).

Model	R	R^2	Adjusted R^2	RMSE	p
H_0	0.370	0.137	0.126	1.063	< 0.001
H_1	0.431	0.186	0.164	1.039	< 0.001

When the checklist is filled out, the average severity is reduced by 0.853, and every additional developer without the Rangeforce training increases the severity by 0.167 (see Table 23).

Table 23. Linear regression model coefficients for the average severity level of the reported vulnerabilities in HackerOne (composed by the author).

Model		Unstandardized	Standard Error	Standardized	t	p
H_0	(Intercept)	1.172	0.185		6.336	< 0.001
	CHECKLIST	-0.846	0.244	-0.370	-3.475	< 0.001
H_1	(Intercept)	0.813	0.250		3.250	0.002
	CHECKLIST	-0.858	0.238	-0.375	-3.598	< 0.001
	DEVS_NO_RF	0.167	0.080	0.217	2.080	0.041

The findings of the statistical tests conducted in this chapter confirm the effectiveness of the combined secure coding practices implemented in the researched organization.

5 Conclusions and further research

This thesis gave an assessment of the effectiveness of two secure coding measures – OWASP secure coding practices checklist and Rangeforce training – implemented in a global technology company with operations in Estonia. The combination of qualitative and quantitative research methods were used throughout this study. During data collection the author held semi-structured interviews with internal stakeholders (developers, mission leads, information security team staff) in order to gather input and validate the data collection methods used. When testing hypotheses, multiple statistical analysis methods were used.

The OWASP secure coding checklist on its own proved effective. There is a moderate connection between filling out the checklist during the development phase and the distribution of vulnerabilities reported in HackerOne. This means that the likelihood of HackerOne participants finding a new vulnerability is 2.92 times higher if the checklist was not followed. There is also a significant difference in the average severity of reported vulnerabilities and the total bounty payouts depending on whether the checklist was filled out. Based on the results of Mann-Whitney U test, the author can conclude that following the checklist results in significantly lower bounties and severity of vulnerabilities. Thus, the first assumption of the author was confirmed.

The effectiveness of the Rangeforce training, on the other hand, was not as evident as of the secure coding checklist. The author tested whether the mission lead's participation in the training prior to leading a mission had any connection to the number of reported vulnerabilities, their average severity and the total bounty payouts. Similarly to the secure coding checklist, mission lead's Rangeforce training showed a moderate connection with the distribution of vulnerabilities reported in HackerOne. The odds ratio is slightly lower than of the checklist. When a mission is led by a developer who had Rangeforce training, the likelihood of introducing a vulnerability that would be later found in HackerOne is 2.74 times lower than if the training was not completed. Unlike the secure coding checklist, mission lead's Rangeforce participation did not have a significant connection with the average severity of reported vulnerabilities and the total bounty payouts.

Additionally, the author tested the combined effect of following the OWASP secure coding practices checklist and going through the Rangeforce training on the vulnerability metrics in HackerOne using stepwise linear regression modelling. Only following the re-

quirements specified in the secure coding checklist describes 5.9% of the variance in the number of reported vulnerabilities and 12.6% of the variance in the average severity of those vulnerabilities. However, when combined with the overall Rangeforce training participation, the explanatory power of the model increased to 22% and 16.4%. The results of the linear regression modelling confirm the idea that both secure coding measures are more effective when used together.

The author has multiple ideas for further research. First, the results of this research cannot be directly applied to other similar companies, because numerous additional factors could influence the vulnerabilities reported in the bug bounty program. For example, Thus, the results need to be validated in a sequential confirmatory case study on another technology company that does software development in a similar way.

Second, it would be interesting to conduct a more in-depth research about which types of vulnerabilities the checklist helps to mitigate. Such study would help organizations to make decisions regarding software development lifecycle. If the organization receives mostly certain types of vulnerabilities during the security assessment and the checklist does not efficiently address those vulnerabilities, additional secure coding measures would need to be implemented. Such measures could include vulnerability-specific secure coding training for developers, adding automated vulnerability-specific tests and modifying the checklist with more appropriate requirements.

Lastly, finding out whether the secure coding training has a significant connection with any specific OWASP Top 10 vulnerability types would be also beneficial.

References

[1] *2020 SonicWall Cyber Threat Report*. SonicWall Inc. 2020. URL: https://www.sonicwall.com/resources/2020-cyber-threat-report-pdf (visited on Apr. 15, 2020).

[2] E. Shuster et al. "Web Attacks and Gaming Abuse Report". In: *State of the Internet / Security* 5.3 (2019-06), p. 27. URL: https://www.akamai.com/us/en/multimedia/documents/state-of-the-internet/soti-security-web-attacks-and-gaming-abuse-report-2019.pdf (visited on Apr. 22, 2020).

[3] J. Viega and G. McGraw. *Building Secure Software: How to Avoid Security Problems the Right Way (Paperback) (Addison-Wesley Professional Computing Series)*. 1st. Addison-Wesley Professional, 2011. ISBN: 0321774957.

[4] M. Howard and S. Lipner. *The security development lifecycle*. Vol. 8. Microsoft Press Redmond, 2006.

[5] *ISO 27001 - Annex A.14: System Acquisition, Development & Maintenance*. ISMS.online. URL: https://www.isms.online/iso-27001/annex-a-14-system-acquisition-development-and-maintenance (visited on Apr. 22, 2020).

[6] *The OWASP Cheat Sheet Series GitHub repository*. GitHub. URL: https://github.com/OWASP/CheatSheetSeries (visited on Apr. 27, 2020).

[7] D. S. Cruzes et al. "How is Security Testing Done in Agile Teams? A Cross-Case Analysis of Four Software Teams". In: *Agile Processes in Software Engineering and Extreme Programming*. Ed. by H. Baumeister, H. Lichter, and M. Riebisch. Cham: Springer International Publishing, 2017, pp. 201–216. ISBN: 978-3-319-57633-6.

[8] B. Duncan and M. Whittington. "Reflecting on Whether Checklists Can Tick the Box for Cloud Security". In: *Proceedings of the International Conference on Cloud Computing Technology and Science, CloudCom* 2015 (2014-12). DOI: 10.1109/CloudCom.2014.165.

[9] P. Morrison, B. Smith, and L. Williams. "Surveying Security Practice Adherence in Software Development". In: *Proceedings of the Hot Topics in Science of Security: Symposium and Bootcamp*. HoTSoS. Hanover, MD, USA: Association for Computing Machinery, 2017, pp. 85–94. ISBN: 9781450352741. DOI: 10.1145/3055305.3055312.

[10] B. Subedi et al. "Secure paradigm for web application development". In: *15th RoEduNet Conference: Networking in Education and Research*. 2016, pp. 1–6. DOI: 10.1109/RoEduNet.2016.7753243.

[11] H. Sohoel, M. Jaatun, and C. Boyd. "OWASP Top 10 - Do Startups Care?" In: *2018 International Conference on Cyber Security and Protection of Digital Services (Cyber Security)*. 2018-06, pp. 1–8. DOI: 10.1109/CyberSecPODS.2018.8560666.

[12] T. Vieira and C. Serrao. “Web Applications Security and Vulnerability Analysis Financial Web Applications Security Audit – A Case Study”. In: *International Journal of Innovative Business Strategies* 2 (2016-12). DOI: 10.20533/ijibs.2046.3626.2016.0014.

[13] *OWASP Secure Coding Practices Quick Reference Guide*. The OWASP Foundation. 2010. URL: https://owasp.org/www-pdf-archive/OWASP_SCP_Quick_Reference_Guide_v2.pdf (visited on Jan. 25, 2020).

[14] M. Zhao, J. Grossklags, and P. Liu. “An Empirical Study of Web Vulnerability Discovery Ecosystems”. In: *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*. CCS ’15. Denver, Colorado, USA: Association for Computing Machinery, 2015, pp. 1105–1117. ISBN: 9781450338325. DOI: 10.1145/2810103.2813704.

[15] C. Wohlin. “Guidelines for Snowballing in Systematic Literature Studies and a Replication in Software Engineering”. In: *Proceedings of the 18th International Conference on Evaluation and Assessment in Software Engineering*. EASE ’14. London, England, United Kingdom: Association for Computing Machinery, 2014. ISBN: 9781450324762. DOI: 10.1145/2601248.2601268.

[16] B. De Win et al. “On the secure software development process: CLASP, SDL and Touchpoints compared”. In: *Information and software technology* 51.7 (2009), pp. 1152–1171.

[17] J. Viega and J. Epstein. *The CLASP Application Security Process*. Secure Software, Inc. 2005. URL: https://cwe.mitre.org/documents/sources/TheCLASPApplicationSecurityProcess.pdf (visited on Mar. 3, 2020).

[18] G. McGraw. “Software security”. In: *IEEE Security Privacy* 2.2 (2004), pp. 80–83.

[19] A. Ozment. “Improving Vulnerability Discovery Models”. In: *Proceedings of the 2007 ACM Workshop on Quality of Protection*. QoP ’07. Alexandria, Virginia, USA: Association for Computing Machinery, 2007, pp. 6–11. ISBN: 9781595938855. DOI: 10.1145/1314257.1314261.

[20] *OWASP Top Ten*. The OWASP Foundation. URL: https://owasp.org/www-project-top-ten (visited on Jan. 25, 2020).

[21] T. Nicolaysen et al. “Agile Software Development: The Straight and Narrow Path to Secure Software?” In: *International Journal of Secure Software Engineering (IJSSE)* 1 (2010-01), pp. 71–85. DOI: 10.4018/jsse.2010070105.

[22] D. Geer. “Are Companies Actually Using Secure Development Life Cycles?” In: *Computer* 43 (2010-07), pp. 12–16. DOI: 10.1109/MC.2010.159.

[23] T. E. Gasiba and U. Lechner. “Raising Secure Coding Awareness for Software Developers in the Industry”. In: *2019 IEEE 27th International Requirements Engineering Conference Workshops (REW)*. 2019, pp. 141–143.

[24] B. Khan et al. “Effectiveness of information security awareness methods based on psychological theories”. In: *African journal of business management* 5 (2011-10). DOI: 10.5897/AJBM11.067.

[25] B. Lebek et al. “Information security awareness and behavior: A theory-based literature review”. In: *Management Research Review* 37 (2014-11), pp. 1049–1092. DOI: 10.1108/MRR-04-2013-0085.

[26] D. Votipka et al. “Toward a Field Study on the Impact of Hacking Competitions on Secure Development”. In: 2018.

[27] T. Gasiba et al. “On the Requirements for Serious Games Geared Towards Software Developers in the Industry”. In: 2019-09, pp. 286–296. DOI: 10.1109/RE.2019.00038.

[28] K. Chung and J. Cohen. “Learning Obstacles in the Capture The Flag Model”. In: *2014 USENIX Summit on Gaming, Games, and Gamification in Security Education (3GSE 14)*. San Diego, CA: USENIX Association, 2014-08. URL: https://www.usenix.org/conference/3gse14/summit-program/presentation/chung.

[29] J. Vykopal et al. “Lessons learned from complex hands-on defence exercises in a cyber range”. In: 2017-10, pp. 1–8. DOI: 10.1109/FIE.2017.8190713.

[30] M. Finifter, D. Akhawe, and D. Wagner. “An Empirical Study of Vulnerability Rewards Programs”. In: *Proceedings of the 22nd USENIX Conference on Security*. SEC’13. Washington, D.C.: USENIX Association, 2013, pp. 273–288. ISBN: 9781931971034.

[31] M. Zhao, A. Laszka, and J. Grossklags. “Devising Effective Policies for Bug-Bounty Platforms and Security Vulnerability Discovery”. In: *Journal of Information Policy* 7 (2017), pp. 372–418. ISSN: 23815892, 21583897. URL: http://www.jstor.org/stable/10.5325/jinfopoli.7.2017.0372.

[32] A. Ozment and S. E. Schechter. “Milk or Wine: Does Software Security Improve with Age?” In: *USENIX Security Symposium*. Vol. 6. 2006.

[33] E. Rescorla. “Is finding security holes a good idea?” In: *IEEE Security Privacy* 3.1 (2005), pp. 14–19. ISSN: 1558-4046. DOI: 10.1109/MSP.2005.17.

[34] A. Edmundson et al. “An Empirical Study on the Effectiveness of Security Code Review”. In: *Engineering Secure Software and Systems*. Ed. by J. Jürjens, B. Livshits, and R. Scandariato. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 197–212. ISBN: 978-3-642-36563-8.

[35] M. Al-Banna et al. “Friendly Hackers to the Rescue: How Organizations Perceive Crowdsourced Vulnerability Discovery”. In: *22nd Pacific Asia Conference on Information Systems, PACIS 2018, Yokohama, Japan, June 26-30, 2018*. 2018, p. 230. URL: https://aisel.aisnet.org/pacis2018/230.

[36] A. Tetelman. *Data driven bug bounty*. Accessed on: November 5th, 2019. 2019. URL: https://programanalys.is/blog/data-driven-bug-bounty.

[37] A. T. Chatfield and C. G. Reddick. “Cybersecurity Innovation in Government: A Case Study of U.S. Pentagon’s Vulnerability Reward Program”. In: *Proceedings of the 18th Annual International Conference on Digital Government Research*. dg.o ’17. Staten Island, NY, USA: Association for Computing Machinery, 2017, pp. 64–73. ISBN: 9781450353175. DOI: 10.1145/3085228.3085233.

[38] *The About You Bug Bounty Story*. HackerOne. 2019. URL: https://www.hackerone.com/sites/default/files/2019-04/about-you-case-study.pdf (visited on Nov. 20, 2019).

[39] *Fanduel’s Liam Somerville on prioritising researchers as an extension of the security team*. HackerOne. 2019. URL: https://www.hackerone.com/blog/FanDuels-Liam-Somerville-Prioritising-Researchers-Extension-Security-Team (visited on Nov. 20, 2019).

[40] K. Wang. *Inside the GitLab public bug bounty program*. GitLab blog. 2019. URL: https://about.gitlab.com/blog/2019/04/29/inside-the-gitlab-public-bug-bounty-program (visited on Nov. 20, 2019).

[41] *Grand Rounds VP Infosec: achieving SOC 2 type II compliance with hacker-powered security*. HackerOne. 2019. URL: https://www.hackerone.com/blog/grand-rounds-vp-infosec-achieving-soc-2-type-ii-compliance-hacker-powered-security (visited on Nov. 20, 2019).

[42] *Towards the $50,000 bounty - improving SDLC with bug bounties*. HackerOne. URL: https://www.hackerone.com/resources/localtapiola-case-study (visited on Nov. 20, 2019).

[43] *ownCloud and HackerOne collaborate for better security*. ownCloud. URL: https://oc.owncloud.com/rs/038-KRL-592/images/HackerOne_CaseStudy_ownCloud.pdf (visited on Nov. 21, 2019).

[44] *Program insights from the PayPal security team*. HackerOne. 2019. URL: https://www.hackerone.com/blog/Program-Insights-QA-PayPal-Security-Team (visited on Nov. 21, 2019).

[45] *Priceline launches public bug bounty program: Q&A with Matt Southworth*. HackerOne. 2019. URL: https://www.hackerone.com/sites/default/files/2019-07/priceline-case-study.pdf (visited on Nov. 21, 2019).

[46] *Qualcomm’s Alex Gantman on bug bounties*. HackerOne. 2017. URL: https://www.hackerone.com/blog/qualcomms-alex-gantman-on-bug-bounties (visited on Nov. 21, 2019).

[47] *Sumo Logic uses hacker-powered pen tests for security and compliance*. HackerOne. 2018. URL: https://www.hackerone.com/blog/Sumo-Logic-Looks-Hacker-Powered-Pen-Testing-Security-and-Compliance (visited on Nov. 21, 2019).

[48] *Upserve Resolves Over 85 Bugs in Two Years Thanks to Hackers*. HackerOne. 2019. URL: https://www.hackerone.com/sites/default/files/2019-09/H1-851_Upserve.pdf (visited on Nov. 21, 2019).

[49] *Q&A with Wordpress security team lead, Aaron Campbell*. HackerOne. 2017. URL: https://www.hackerone.com/blog/Q-and-A-with-Wordpress-security-team-lead-aaron-campbell (visited on Nov. 21, 2019).

[50] A. Salameh and J. M. Bass. "Spotify Tailoring for Promoting Effectiveness in Cross-Functional Autonomous Squads". In: *Agile Processes in Software Engineering and Extreme Programming – Workshops*. Ed. by Rashina Hoda. Cham: Springer International Publishing, 2019, pp. 20–28. ISBN: 978-3-030-30126-2.

[51] S. Easterbrook et al. "Selecting Empirical Methods for Software Engineering Research". In: *Guide to Advanced Empirical Software Engineering*. Ed. by F. Shull, J. Singer, and D. I. K. Sjøberg. London: Springer London, 2008, pp. 285–311. ISBN: 978-1-84800-044-5. DOI: 10.1007/978-1-84800-044-5_11.

[52] D. J. Sheskin. *Handbook of Parametric and Nonparametric Statistical Procedures*. 4th ed. Chapman & Hall/CRC, 2007. ISBN: 1584888148.

[53] *Changelog - Automatic Invitations for Private Programs*. HackerOne. URL: https://docs.hackerone.com/changelog/#automatic-invitations-for-private-programs (visited on Apr. 29, 2020).

[54] H. Akoglu. "User's guide to correlation coefficients". In: *Turkish Journal of Emergency Medicine* 18 (2018-08). DOI: 10.1016/j.tjem.2018.08.001.

[55] M. L. McHugh. "The Chi-square test of independence". In: *Biochemia Medica* 23 (2013), pp. 143–149.

[56] J.W. Kotrlik, H.A. Williams, and M.K. Jabor. "Reporting and Interpreting Effect Size in Quantitative Agricultural Education Research". In: *Journal of Agricultural Education* 52 (2011-03), pp. 132–142. DOI: 10.5032/jae.2011.01132.

[57] A. Field. *Discovering Statistics Using IBM SPSS Statistics*. 4th. Sage Publications Ltd., 2013. ISBN: 1446249182.

[58] T. Marchant-Shapiro. *Statistics for political analysis: Understanding the numbers*. SAGE Publications, 2015. ISBN: 9781452258652. DOI: 10.4135/9781483395418.

[59] N. Nachar. "The Mann-Whitney U: A Test for Assessing Whether Two Independent Samples Come from the Same Distribution". In: *Tutorials in Quantitative Methods for Psychology* 4 (2008-03). DOI: 10.20982/tqmp.04.1.p013.

Appendix 1 – HackerOne case studies analysis (compiled by the author based on the cited sources)

Organization name	**HackerOne program type**	**Reason for a VRP**	**Key findings**
ABOUT YOU	private	Internal security engineers and periodic penetration tests are not effective in a scaling organization [38]	Maintain fast resolution time; White hat researcher communication is important [38]
Fanduel	public	Small security team and periodic penetration tests are not scalable [39]	Treat white hat researchers well and build relationships with them [39]
GitLab	public	Impossible to manually assess code security; Code scanning is not effective [40]	Quick response, triage and fix times; Reports of patched vulnerabilities are made public 30 days after fixing the issue [40]
Grand Rounds	not specified	Regular penetration testing is predictable and therefore less effective [41]	Organizations should outsource security testing to 3rd parties (VRP) in order to get unbiased findings [41]
LocalTapiola	public	Inefficient and lengthy software development and security assessment process [42]	High maximum reward amount results in more reports; Having developers pay for the vulnerabilities in their code motivates to make less mistakes [42]

Organization name	**HackerOne program type**	**Reason for a VRP**	**Key findings**
ownCloud	private => public	Penetration tests did not find vulnerabilities [43]	Launched private program first and opened public program after internal processes were optimized; Increasing rewards to stimulate white hats' motivation vulnerabilities are reported over 21 times faster than before a VRP [43]
Paypal	public	Large pool of white hat researchers for more extensive application testing [44]	Remote code execution (RCE), server-side and database attacks get the most attention from Paypal; A clearly stated impact in the vulnerability report is important [44]
Priceline	public	Extend the overall cybersecurity strategy [45]	When new attacks are introduced, they are usually reflected in the vulnerability reports. This helps stay ahead of malicious actors; RCE, logic flaws and mobile application vulnerabilities deserve more attention from the security team [45]
Qualcomm	private	Extend the overall cybersecurity strategy [46]	Private program mitigates unnecessary noise in submissions; White hat researchers like to work with people, so maintaining direct relationships is important; Vulnerability reports should be included in the feedback loop. This helps improve the software development lifecycle [46]
Sumo Logic	private	Penetration tests did not find vulnerabilities [47]	12 vulnerabilities missed during the penetration tests found in the first 15 days after VRP launch [47]

Organization name	**HackerOne program type**	**Reason for a VRP**	**Key findings**
Upserve	private => public	Needed a new perspective on vulnerability finding; Internal security reviews are not efficient [48]	Started with a private program and 7 months later switched to public; Public disclosure of vulnerabilities is useful for white hats and the organization itself; Internal stakeholder support is crucial when fixing reported vulnerabilities [48]
Wordpress	private => public	Large pool of white hat researchers for more extensive application testing [49]	Private program was used to make sure internal security team optimized triage and patch processes; Public program increased number of reports and the team had to prioritize based on potential impact [49]