Tampere University

Miina Rautakorpi

USABILITY PROBLEMS IN VULNERABILITY REPORTS

Bachelor's thesis
Faculty of Information Technology and Communication Sciences
October 2022

ABSTRACT

Miina Rautakorpi: Usability problems in vulnerability reports
Bachelor's thesis
Tampere University
Information Technology
October 2022

Security is important part of information technology industry. Cyber attacks are common and often happen because of vulnerabilities that were already known before the attack happens. Automated and continuous process of tracking vulnerabilities is important to prevent these security breaches. In DevSecOps process, developers are expected to find vulnerabilities early in the product's development cycle. One way to achieve this is to use vulnerability scanners and vulnerability reports produced for developers. The meaning of the vulnerability report is to show the user a statement of application's vulnerabilities and weaknesses.

This study investigates what are the common usability problems in vulnerability reports. The reports are often unclear, not taken seriously, have unnecessary or missing information, and are divided between many different scanners. Usability problems lead to misunderstandings and unsecure applications.

As a result, possible solutions to these usability problems are presented. To prevent the problems the design of the vulnerability report needs to be planned carefully. Designer should plan the interface with usability in mind, combine different scanner results to get an overview, and not make any assumptions on developer's knowledge. The data needs to be minimized to necessary information and the contents of the report should consider the needs of the developer.

Preventing usability problems in vulnerability reports requires lot of time and resources from the organization. However, usable vulnerability reports enable effortless vulnerability management systems and reliable applications.

Keywords: vulnerability report, vulnerability scanner, usability, developer, DevSecOps, information security

TIIVISTELMÄ

Miina Rautakorpi: Käytettävyysongelmat haavoittuvuusraporteissa
Kandidaatintyö
Tampereen yliopisto
Tietotekniikka
Lokakuu 2022

Tietoturva on tärkeä osa tietotekniikkateollisuutta. Useimmat kyberhyökkäykset ovat peräisin haavoittuvuuksista ja virheistä, jotka olivat tiedossa jo ennen hyökkäystä. Automaattinen ja jatkuva prosessi haavoittuvuuksien seuraamiseen on tärkeää, jotta nämä hyökkäykset voidaan estää. DevSecOps-prosessissa oletetaan, että ohjelmistokehittäjät tunnistavat haavoittuvuudet jo aikaisessa vaiheessa tuotteen kehityssykliä. Yksi tapa saavuttaa tämä on käyttää haavoittuvuusskannerien muodostamia haavoittuvuusraportteja, jotka ovat tuotettu juuri ohjelmistokehittäjille. Haavoittuvuusraportin tavoitteena on havainnollistaa käyttäjälle ohjelmiston tila löydettyjen haavoittuvuuksien perusteella.

Tässä tutkielmassa käydään läpi, mitä käytettävyysongelmia haavoittuvuusraporteissa yleensä on. Raportit ovat usein epäselviä, niitä ei oteta vakavasti, niissä on turhaa tai puuttuvaa tietoa ja ne ovat jakaantuneet useaan eri paikkaan. Käytettävyysongelmat johtavat väärinkäsityksiin ja suojaamattomiin ohjelmistoihin.

Tuloksena esitetään mahdollisia ratkaisuja näihin käytettävyysongelmiin. Jotta ongelmia voidaan välttää, haavoittuvuusraportin muoto tulee suunnitella huolellisesti. Suunnittelijan tulisi suunnitella käyttöliittymä käytettävyys mielessään, yhdistää eri skannereiden tulokset yleiskatsauksen saamiseksi, eikä tehdä mitään olettamuksia ohjelmistokehittäjien osaamisesta etukäteen. Datan tulisi sisältää vain tarpeellista tietoa ja raportin sisältö tulisi suunnitella ohjelmistokehittäjien tarpeiden mukaan.

Käytettävyysongelmien ehkäiseminen haavoittuvuusraporteissa vaatii paljon aikaa ja resursseja yritykseltä. Käytettävyydeltään hyvät haavoittuvuusraportit kuitenkin mahdollistavat vaivattoman haavoittuvuusprosessin ja luotettavat ohjelmistot.

Avainsanat: haavoittuvuusraportti, haavoittuvuusskanneri, ohjelmistokehittäjä, käytettävyys, DevSecOps, tietoturva

CONTENTS

1. INTRODUCTION

Security in the IT industry is important factor for both service providers and users. Failures in security can be problematic or even catastrophic for information systems. Most of the cyber attacks are results of vulnerabilities or failures that were already known before the attack happens. That is why it is important to track vulnerabilities and possible risks early and as a continuous process. (Karabašević et al., 2018)

DevOps, from words development and operations, is a crucial part of many organization's strategy. It aims to improve collaboration between programmers and operators, reduce development life cycle and upgrade the quality of the software. DevSecOps is DevOps process with security included. In DevSecOps, it is important to address security problems early in the product's development cycle. It is done with continuous testing, deploying and validating the software. When bugs and vulnerabilities are found early, the possible cost is usually smaller and the recovery is fast. (Akbar et al., 2022) This kind of DevSecOps process requires ways for the developers to find the vulnerabilities. There needs to be a fluent way to scan the code and see the security problems early in the process. One way to accomplish this is to use vulnerability scanners. However, based on a survey, many currently available vulnerability scanners are hard to operate or the solutions provided for vulnerabilities are too difficult to understand (Yoshimoto et al., 2005).

Vulnerability scanners are used to analyze the application. Scanners produce vulnerability reports, which are meant to show the user the state of the application based on found vulnerabilities. Research for vulnerability scanners is active, but there is not much research made about scanner results (Reynolds et al., 2021). The study about vulnerability reports is still essential and needed since the user acts based on vulnerability reports. It is important to understand what the report should include so that it would be the most useful for the specific user group and the possible organization as well. Nembhard et al. express that many developers are sceptical of using code analyzing tools. It is mainly because unusable interface design and poor presentations of vulnerability warnings. Vulnerability scanners are often focused on the technical aspects, leaving usability neglected or forgotten. (Nembhard and Carvalho, 2019) This study investigates what are the common usability problems in vulnerability reports. Possible solutions to these problems are also presented.

After the introduction, this study will continue with theory. Basic concepts like vulnerability, vulnerability scanner and vulnerability report are explained. Example of a vulnerability report is shown. Chapter 3 is about usability problems in vulnerability reports. Problems include unclear reports, security as a secondary concern, unnecessary information, missing information and many different scanners. Chapter 4 is about solutions for problems presented in chapter 3. Solutions include more understandable reports, how to define necessary information, visual analytic tool called VULNUS, and suggestions on interface design. Chapter 5 is the conclusion of the study.

2. THEORY

2.1 Vulnerability and vulnerability management systems

Dowd et al. present that software vulnerabilities are some kind of weaknesses in the system. Vulnerabilities are a way for an attacker to get inside the system. Attacker could for example want to steal some sensitive data, destroy the system or take control of the system. Vulnerabilities are usually software bugs that have security issues. However, this is not the case every time. Software can work exactly how it is supposed without any bugs, but still have a vulnerability for example with access rights. (Dowd et al., 2007)

Vulnerability management systems are important for manufacturers and organizations. Manually fixing and finding vulnerabilities is difficult and expensive (Zulkarneev and Kozlov, 2021). The system should be automated and a continuous process, because new vulnerabilities might come up daily. For example, new code containing vulnerabilities might be written and added to the system or new versions of libraries could have come up, leaving the old versions vulnerable. Attackers are trying to figure new ways to breach the systems continuously, so keeping security defences up-to-date and secure is a challenging process that requires a lot of attention. One way to automatically control the vulnerabilities is to use vulnerability scanners. Vulnerability scans form vulnerability reports, that can be used to view the current state of the software.

2.2 Vulnerability scanners

Vulnerability scanners are a way to collect information about the state of server operating systems and network devices (Karabašević et al., 2018). Scanner's idea is automatically analyze the application. For example, scan can be configured to begin every time there is changes in version control on the main branch.

Every vulnerability scanner has its own analysis method. The analysis can be categorized to be either static analysis or dynamic analysis. Static analysis means that application's data is parsed into data structures. It does not notice change over time and only analyses the current application as it is without running it. Dynamic analysis is the opposite from static. It is made while the application is running and usually includes inputs and outputs from the user interface. A problem with static analysis is that it usually shows a lot of

false positives. False positive in vulnerability scanners means that a vulnerability is found, but with further inspecting it is noticed not to be actually relevant. Static analysis spots a vulnerability, when scanning result is technically incorrect. However, some technically incorrect parts might not be used at run time at all, which causes the finding to be false positive. (Reynolds et al., 2021)

The result of a vulnerability scan could be viewed as a vulnerability report. For example vulnerability scanners Anchore and SonarQube make static analyses and form vulnerability reports based on them. Example of a dynamic vulnerability scanner would be Tenable.sc. Dynamic scanners also form reports.

2.3 Vulnerability reports

Vulnerability report is a statement of application's vulnerabilities. Usually vulnerability report is made by a vulnerability scanner. The goal of the report is to show the user the state of the application based on found vulnerabilities. The report may for example include how many vulnerabilities were found, what severities they were or which parts of the code were unsecure. Severity is usually low, medium, high or critical regarding how dangerous the vulnerability is. There may also be different graphs or visualizations to help the user notice the overview.

Vulnerability reports are needed to identify, prioritize and remediate vulnerabilities (Chapple, 2018). The organization can make its own strategy considering who will view the reports and be responsible to act based on them. The reports should be different based on who is meant to be reading them. One option would be to send vulnerability reports to management and they could choose which vulnerabilities or actions are worth of putting time and effort into them. Other option is to send reports straight to security architect, who might have more knowledge regarding vulnerabilities. The architect could for example know better, which vulnerabilities require immediate attention and which are false positives. However, concept of "shift-to-left" is getting more popular in organizations. It shifts more responsibility to the developers. The idea is that the issues are found and quality risks are removed earlier in the cycle (Nader-Rezvani, 2018). This is a way to fasten the process, create better customer experience and optimize cost (Nader-Rezvani, 2018). Developers make the code and therefore in the end they will be the ones fixing the vulnerabilities as well. If someone else views the reports, the information would take a while to return back to the developers. Management could have good ideas about prioritization and security architect could know specific details about security, but this study inspects the vulnerability reports from developer's point of view.

There is an example of a vulnerability report in figure 2.1. At the top of the report user can see some general numbers, like the total number of CVEs, exploitable CVEs, new CVEs in the last week, and overall number of devices and exposed devices. CVE (Common

Vulnerabilities and Exposures) is an international number to identify the common vulnerabilities. CVE number is often defined with Common Vulnerability Scoring System, CVSS (Angelini et al., 2019). The report also includes two lists, vulnerabilities ordered by largest number of exposed devices and just exposed devices. There is one table about the number of CVEs organized by severity and exploitability level. Two figures contain number of vulnerabilities by their severity level, and number of CVEs and exposed devices by vulnerability's severity level.

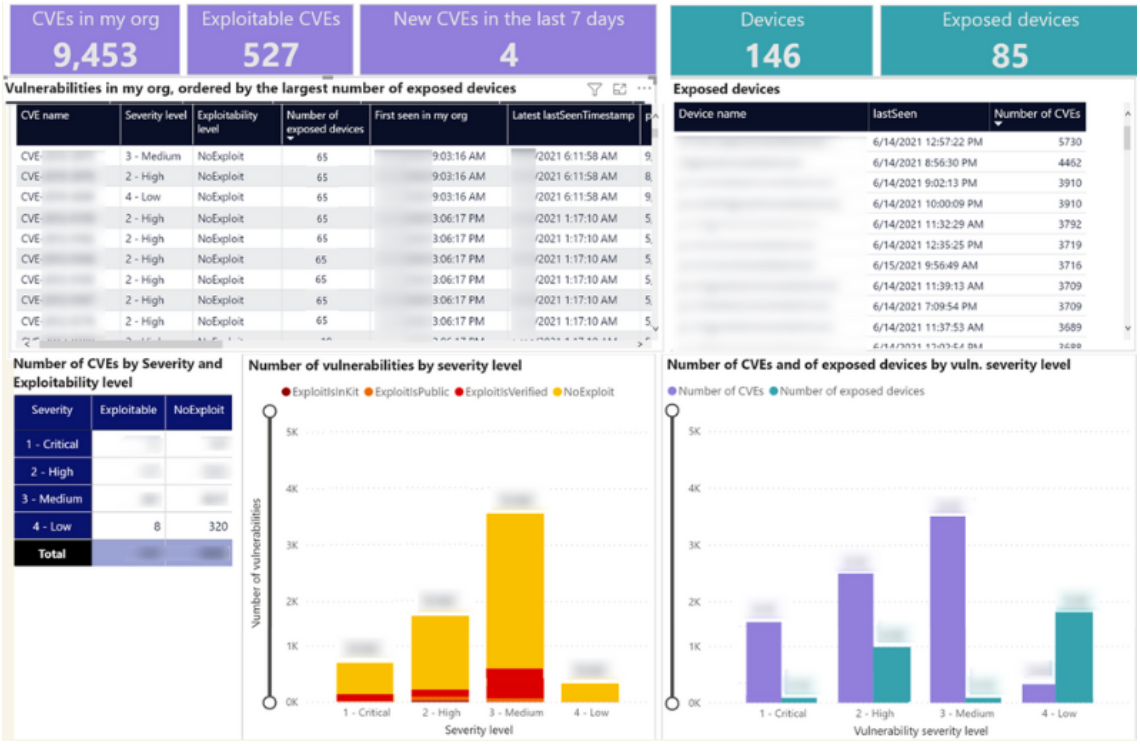

Figure 2.1. *Example of a vulnerability report (Kischel, 2021)*

All vulnerability reports are not so informative as the example in figure 2.1. Reports can vary a lot and this study will be focused into common usability problems the vulnerability reports may have and solutions to these problems.

3. USABILITY PROBLEMS IN VULNERABILITY REPORTS

3.1 Unclear reports

Researches have shown that a common reason for security tools' usability problems is that the developers making the tools assume the users to know or understand more than they actually do. Developers can be expected to have high level of technical knowledge and therefore it is easy to assume that they also have knowledge on security matters. (Acar et al., 2016) This concludes, that vulnerability reports indicated to developers might be too complicated and hard to understand. When a user is reading something he necessarily does not understand, it is easy to give up and move to a next task without resolving the problem. When important parts of the report are not understood, it might lead to misunderstandings and incorrect assumptions. If no-one acts based on reports, or acts based on minor understanding, the vulnerabilities pile up and the reliability of the program worsens. In worst case scenario the acts or not acting at all leads to security breaches and cyber attacks.

Angelini et al. present that most of the visual information in cyber security commercial tools is simple, and does not support more abstract visualizations. In this context, simple visualizations would be for example bar and pie charts. They are commonly used in vulnerability reports. More abstract visualizations like treemaps would be needed when managing large networks. They are hierarchical structures that are presented with nested rectangles (Sondag et al., 2018). Treemaps help to bring up all the important aspects and relations, that would be missing in simple charts. (Angelini et al., 2019) However, too complicated visualizations might lead to unclear reports. Simple visualizations can be easily understood with common knowledge, for example without the need to understand the relationships between nodes in treemaps. Abstract visualizations are better for presenting large and complicated networks, but they can easily lead to unclear report, where user is expected to understand complicated structures.

3.2 Security as a secondary concern

One usability problem is that security is often a secondary concern for developers (Acar et al., 2016). Security itself does not show directly in organization's success or profit. Meaning that security work is often invisible and happens in the background. That is why security work is easy to ignore or work around (Acar et al., 2016).

However, security should be one of the top priorities in organizations. Many applications handle sensitive information, which needs to be kept safe. Security breaches could have enormous effects, like a loss of trust or reputation from customers and public. Managing these breaches is consuming and expensive for the organization. Some customers can have their own vulnerability scanning methods and if they are not happy with the product's safety, they might buy the product from somewhere else. When ignoring security matters is easy, it can easily lead to poor vulnerability management.

3.3 Unnecessary information

Sometimes it is hard to find meaningful information from vulnerability report, since there can be too much irrelevant data. As Acar et al. present, continuous adding of more data and security advice to vulnerability report is not a viable solution to make the report more usable (Acar et al., 2016). Big amounts of data can overwhelm users and make it harder to spot the vulnerabilities and errors that actually matter and need to be fixed.

Static analysis tools produce large amounts of false positives (Thomas, 2015). A way to detect these false positives is that a security expert goes through all the vulnerabilities and whitelists the ones that do not count. Whitelisting means, that the vulnerability is allowed and it does not effect scans or safety, based on whitelister's opinion and knowledge. This approach is slow and dangerous. Some vulnerabilities might be whitelisted by accident or without valid knowledge.

Vulnerability triage is a process of ruling out false positives. It aims to estimate the criticality of the issues and prioritize them. Unfortunately this process needs to be made manually by a security expert and takes a lot of time and effort. (Reynolds et al., 2021)

False positives compromise the overview of the report. It is hard to spot relevant information if only some of the vulnerabilities are valid. Not all developers can be masters in security, but in this approach, they would still be expected to know details about specific vulnerabilities. In a better approach, there should be verification that the vulnerabilities in the report are actually correct (Thomas, 2015).

3.4 Missing information

Different scanners find different kind of vulnerabilities. Predicting missing information is hard, because it is unclear how well each scanner performs with different types of vulnerabilities. Different scanners can be tested with benchmarks that are specifically designed for this purpose. The tests show how much variation there is between different scanner results. Testing starts with vulnerable code, and when the same code is run with different scanners, the results can be compared with each other. Benchmarks can also be used to test how the scanner spots false positives. Famous benchmarks are for example OWASP and WAVSEP, which both include several different test cases with most common vulnerabilities found from applications. (Lavens et al., 2022)

Lavens et al. conducted a study where they tested several vulnerability scanners and gave them a score based on true positives, false positives, true negatives, and false negatives. They wanted to find out which scanners were most effective to spot certain vulnerability types. As a result, dynamic scanners only detected small amount of vulnerabilities compared to static scanners, but the amount of false positives was significantly smaller. Static scanners gave user feedback and location of the vulnerability, but dynamic scanners only generated an exploit. Injection vulnerabilities like cross-site scripting and SQL injection were mostly found by dynamic scanners. Static scanners were better in catching configuration issues, like weak cryptography and trust boundary violation. (Lavens et al., 2022) Dynamic scanners seem to miss more information and static scanners seem to have more unnecessary information in them.

3.5 Many different scanners

As explained in the chapter 2.2, vulnerability scanners can be separated into static and dynamic analysis methods. However, also static and dynamic analysis can be separated into smaller sections.

Static scanners can be divided into vulnerability scanners and static code analyzers. Vulnerability scanners, for example Anchore, Grype and Black Duck, investigate third party software versions and compare the vulnerabilities found to newest available versions. Anchore and Black Duck also produce software Bill of Material (BOM). BOM is a listing of the primary data that supports the information technology of complex products (Wang et al., 2022). Static code analyzers, like SonarQube and Coverity, examine the code and find coding errors or dangerous implementations that can cause vulnerabilities.

Dynamic scanners can be separated into remote and local scans. Remote scans only run tests to interfaces that can be accessed from outside the system, without authentication or certificate details. Local scan runs the tests from inside the system. It means that the scanner is given needed authentication and certificates, so that the scanner is able to

test all possible actions. These actions may for example include getting into the web user interface or being able to connect and fetch data from API.

To get a decent view of application's safety, different scans are needed for different stages. For example, dynamic analysis scanners are needed for port scanning, vulnerability scanning, API scanning and robustness testing. One scanner is not enough to fulfill all security regulations. All the scanners produce their own vulnerability reports, which concludes that there is vulnerability reports scattered in many different places. Vulnerability report's goal is to show the user the state of the application, but this way none of the reports give satisfying overviews. Instead, user is expected to know how to find, combine and understand several different reports together.

4. SOLUTIONS TO USABILITY PROBLEMS

4.1 Understandable reports

When designing a vulnerability report, there should not be any assumptions about user's, in this case developer's, knowledge. The report should be made in a way that all developers can understand it without specific knowledge on security matters. Tool developers should consider the needs of the developer and focus to usability and the main goals developers have considering the report.

Vulnerability reports are better understood, if there is not too much data in them. It is good to minimize the possible vulnerabilities by removing the user from the loop (Acar et al., 2016). User should have as little responsibilities considering safety as possible. Developer can for example make the updates automatic, force browsers to use HTTPS and choose secure defaults (Acar et al., 2016). However, sometimes it is not possible to make secure choices for user's behalf. In those times developers should use opinionated design, where users are encouraged to make secure choices without the need to actually understand the situation (Acar et al., 2016). Minimizing the overall of vulnerabilities helps with decent sized vulnerability reports where the amount of information is not overwhelming. Other option to decrease the amount of data is to prioritize available data and only show some of it in vulnerability reports.

One way to make more understandable reports is to educate developers in security matters. They should all understand simple security terms and the vulnerability management plan. As presented in chapter 3.2, security is often a secondary concern for developers. The organizations should make security more visible part of the work, so it would be more routine standard and not an extra burden.

4.2 Necessary information

Reynold et al. conducted a survey about the most important aspects in a vulnerability analysis system. Users were asked about their experience, goals and tasks with vulnerability scanners. There were 18 participants and seven of them were software developers and 11 were security experts. The study focused into attributes in a vulnerability report and in a single vulnerability. Visual and interactive aspects were taken into account as

well. Participants told that important things in the reports were high and medium severity vulnerabilities, and metadata. Each vulnerability needed to include category, title and description. Visualizations of overview, most relevant data and different app versions that had new vulnerabilities were wanted. Interactively, the ability to search and filter, exporting the vulnerabilities to other systems and viewing the vulnerabilities in the code were the most important. Based on this survey, the most necessary aspects in a vulnerability report are overview and the ability to find relevant vulnerabilities that require fixing. (Reynolds et al., 2021)

Too much information often leads to unclear reports. Other problem with unnecessary information was to handle the amount of false positives. Predefined rules that scanners have can not fully determine the accuracy of a vulnerability (Gowda et al., 2018). Gowda et al. present a machine learning solution, where Decision tree classifier and Random forest classifier algorithms are built into scanners. Decision tree uses observations from the item and based on item's target value formats conclusions. Random forest produces a class, which is individual tree's class mode or mean prediction. Both algorithms spotted over 90 percent of the false positives. (Gowda et al., 2018)

4.3 A specific tool: VULNUS

Angelini et al. introduce a solution called VULNUS (VULNerabilities visUal aSsessment), which is a visual analytic tool that helps developers to understand the network status and vulnerability spreading visually. VULNUS presents a visualization about the role of each vulnerability in possible attack paths with other features like CVSS scores. Most often used CVSS score is Overall score. However, with Overall score, some details from other scores are left out. VULNUS aims to get the details by visually encoding all the scores, relying on the Target Environment score, which aims to define the dangerousness of a vulnerability. The goal of VULNUS is to increase situational awareness and help detecting and prioritizing the vulnerability fixes. (Angelini et al., 2019)

Visually VULNUS uses the structure of the classical User Centered Design approach, but users are taken into more active role with the design process. Parts of the requirement collection included semi-structured interviews with the key users, direct observation of the operators handling the vulnerabilities, and comparative study of the potential competitors. As a result, sub-networks were ordered by the ascending number of vulnerabilities and colours were used to implicate for example the dangerousness. A scaling option for colour blind people was added. User was able to for example select elements in different ways, get more data from certain vulnerability by mouse-overing, and easily compare multiple vulnerabilities with each other. (Angelini et al., 2019)

VULNUS is one possible solution when planning the visualization of a vulnerability report. However, the solution is new and does not have any external studies made from it. There

are no benchmarks available for systems like VULNUS, and therefore some evaluations have not been made at all (Angelini et al., 2019). The studies Angelini et al. organized were mostly involved with security experts (Angelini et al., 2019), so developer's point of view for the reports was left out. To conclude, VULNUS is a work in progress, not a ready solution to be used yet. In the future it might be used in many vulnerability reports.

4.4 Interface design

Interface design has a huge impact on usability of the tool. Developers designing vulnerability scanners should take a user-centered approach. (Nembhard and Carvalho, 2019) Scanners are used and interpreted by people, so people should also be the first priority of developers.

Developers of vulnerability scanners can follow ten best design principles to ensure user-centered approach. Firstly, there must be an obvious start, so that user knows how to start interacting with the tool. The tool must have clear reverse and consistent logic, which helps user to identify logical patterns between actions. The interface of the tool should be designed with familiar words, images and conventions. Users should receive feedback from actions they do, know their location in the interface and be able to perform similar actions with little navigation. The tool should be open to adaptation for users to modify the interface based on their frequent actions. If user comes across insurmountable problems, some kind of support source should be provided. The purpose of the interface should be to access the content. By following these principles interacting with the scanner should be more productive and require less effort from the user. (Nembhard and Carvalho, 2019)

Vulnerability reports can be interactive, or just pictures or texts about software's state. Most of the design principles are configured for interactive interfaces, but some of them can be adapted to vulnerability reports without interactive actions. For example, first design principle was that user must know how to start interacting with the tool. When user sees a report, he mentally organizes the elements to understand the meaning behind them (Pang et al., 2016). Vulnerability report should be presented in a way user's eye can follow a certain path from element to element. The path should start with most critical findings and end with less important data. Designer can create the path with for example images, color, size, space and position (Pang et al., 2016). This kind of design requires a lot of psychological knowledge from the designer.

5. CONCLUSION

It is important to track vulnerabilities and risks early, as a continuous process. Tracking can be made with vulnerability scanners that analyze the application and produce vulnerability reports. In this study, common usability problems in the vulnerability reports and solutions to them were presented.

Common reason for usability problems is that developers making the tool often expect users to understand more than they actually do. Unclear reports lead to misunderstandings, incorrect assumptions and giving up. Abstract visualizations might be too hard to understand, but simple visualizations might not be informative enough. Security is also often a secondary concern for developers. Security itself does not show directly in organization's profit, so it is easy to ignore or work around. Many vulnerability reports include unnecessary information, like overwhelming amounts of data or big amounts of false positives. However, it is common that there is missing information as well. It is complicated to balance between the right amount of data. Different scanners are needed to find different types of vulnerabilities. All scanners produce their own vulnerability reports, so it is impossible to get a satisfying overview from one report at the moment.

Ways to make vulnerability reports more understandable are minimizing the amount of possible vulnerabilities from the beginning, not make any assumptions on user's knowledge or train users to understand reports. The vulnerability reports are better understood if there is not too much data in them. Solution to handle false positives with machine learning was presented. Reports should be designed with the goal in mind and not make them too complicated. VULNUS is a visual analytic tool in progress, that helps to increase situational awareness, and detect and prioritize vulnerability fixes. Regarding interface design, designers of the vulnerability reports should try to follow ten best design principles presented in chapter 4.4.

It would be beneficial to have one main vulnerability report that gathers data from each scanner's vulnerability reports. This report would provide an overview, which takes more than one scanner results into account. These kind of overview reports are not publicly available in the markets at the moment. Based on the survey presented in 4.2, the most necessary aspects in vulnerability reports are the overview and the ability to find relevant vulnerabilities. It proves that an overview report would be beneficial, since one vulnera-

bility scanner can not provide a decent overview of the whole application, as explained in chapter 3.5.

Designing vulnerability reports and correcting their usability mistakes is not easy, and requires lot of time and resources. However, the work is necessary. Usable vulnerability reports enable effortless vulnerability management systems and reliable applications.

REFERENCES

Acar, Y., Fahl, S., & Mazurek, M. L. (2016). You are not your developer, either: A research agenda for usable security and privacy research beyond end users. *2016 IEEE Cybersecurity Development (SecDev)*, 3–8. https://doi.org/10.1109/SecDev.2016.013

Akbar, M. A., Smolander, K., Mahmood, S., & Alsanad, A. (2022). Toward successful devsecops in software development organizations: A decision-making framework. *Information and Software Technology*, *147*, 106894. https://doi.org/https://doi.org/10.1016/j.infsof.2022.106894

Angelini, M., Blasilli, G., Catarci, T., Lenti, S., & Santucci, G. (2019). Vulnus: Visual vulnerability analysis for network security. *IEEE Transactions on Visualization and Computer Graphics*, *25*(1), 183–192. https://doi.org/10.1109/TVCG.2018.2865028

Chapple, a., Mike. (2018). *Vulnerability scanning* (1st edition). Sybex.

Dowd, M., McDonald, J., & Schuh, J. (2007). *The art of software security assessment : Identifying and preventing software vulnerabilities* (1st edition). Addison-Wesley.

Gowda, S., Prajapati, D., Singh, R., & Gadre, S. S. (2018). False positive analysis of software vulnerabilities using machine learning. *2018 IEEE International Conference on Cloud Computing in Emerging Markets (CCEM)*, 3–6. https://doi.org/10.1109/CCEM.2018.00010

Karabašević, D., Stanujkić, D., Brzaković, M., Maksimović, M., & Jevtić, M. (2018). Importance of vulnerability scanners for improving security and protection of the web servers. *BizInfo*, *9*(1), 19–29.

Kischel, K. (2021). New threat vulnerability management apis - create reports, automate, integrate. *TechCommunity*. https://techcommunity.microsoft.com/t5/microsoft-defender-vulnerability/new-threat-amp-vulnerability-management-apis-create-reports/ba-p/2445813

Lavens, E., Philippaerts, P., & Joosen, W. (2022). A quantitative assessment of the detection performance of web vulnerability scanners. *Proceedings of the 17th International Conference on Availability, Reliability and Security*. https://doi.org/10.1145/3538969.3544416

Nader-Rezvani, N. (2018). Agile quality test strategy: Shift left. *An executive's guide to software quality in an agile organization* (pp. 121–138). Apress.

Nembhard, F., & Carvalho, M. (2019). The impact of interface design on the usability of code analyzers. *2019 SoutheastCon*, 1–6. https://doi.org/10.1109/SoutheastCon42311.2019.9020339

Pang, X., Cao, Y., Lau, R., & Chan, A. (2016). Directing user attention via visual flow on web designs. *ACM Transactions on Graphics*, *35*(6), 1–11.

Reynolds, S. L., Mertz, T., Arzt, S., & Kohlhammer, J. (2021). User-centered design of visualizations for software vulnerability reports. *2021 IEEE Symposium on Visualization for Cyber Security (VizSec)*, 68–78. https://doi.org/10.1109/VizSec53666.2021.00013

Sondag, M., Speckmann, B., & Verbeek, K. (2018). Stable treemaps via local moves. *IEEE Transactions on Visualization and Computer Graphics*, *24*(1), 729–738. https://doi.org/10.1109/TVCG.2017.2745140

Thomas, T. (2015). Exploring the usability and effectiveness of interactive annotation and code review for the detection of security vulnerabilities. *2015 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*, 295–296. https://doi.org/10.1109/VLHCC.2015.7357234

Wang, Y., Ren, W., Zhang, C., & Zhao, X. (2022). Bill of material consistency reconstruction method for complex products driven by digital twin. *International Journal of Advanced Manufacturing Technology*, *120*(1-2), 185–202.

Yoshimoto, M., Bista, B., & Takata, T. (2005). Development of security scanner with high portability and usability. *19th International Conference on Advanced Information Networking and Applications (AINA'05) Volume 1 (AINA papers)*, *2*, 407–410 vol.2. https://doi.org/10.1109/AINA.2005.163

Zulkarneev, I., & Kozlov, A. (2021). New approaches of multi-agent vulnerability scanning process. *2021 Ural Symposium on Biomedical Engineering, Radioelectronics and Information Technology (USBEREIT)*, 0488–0490. https://doi.org/10.1109/USBEREIT51232.2021.9455061