Asking Security Practitioners: Did You Find the Vulnerable (Mis)Configuration?

Richard May
Harz University of Applied Sciences
Wernigerode, Germany
rmay@hs-harz.de

Christian Biermann
msg services gmbh, Harz University of Applied Sciences
Hamburg, Germany
christian.biermann@msg.group

Jacob Krüger
Eindhoven University of Technology
Eindhoven, Netherlands
j.kruger@tue.nl

Thomas Leich
Harz University of Applied Sciences
Wernigerode, Germany
tleich@hs-harz.de

Abstract

With ever evolving software, reliability and quality assurance are facing growing complexity and security issues. Particularly, interconnected and configurable systems are threatened by (mis)configurations that can lead to exploitable vulnerabilities. Unfortunately, there is limited information on how such configuration vulnerabilities occur or how practitioners deal with these. To tackle this gap, we investigated the connections between (mis)configurations, vulnerabilities, and their treatment by conducting a survey with 41 security practitioners who have encountered configuration vulnerabilities in their work. More precisely, our objectives were to understand the causes, prevalence, severity, and treatments of such vulnerabilities. We found that configuration vulnerabilities are prevalent and severe in practice. They primarily stem from dependency issues, outdated software, and inconsistent (cross-)configurations; are typically influenced by human errors; and are either identified during testing or, in the worst case, during deployment and operation. Generally, configuration vulnerabilities are detected due to security incidents or through word-of-mouth, implying that more preventive security management is required—ideally at an early stage and as part of a holistic security-engineering process. Overall, we aim to enhance the understanding of researchers and practitioners regarding current practices related to handling configuration vulnerabilities as well as open challenges.

CCS Concepts

• **Software and its engineering**; • **Security and privacy** → **Vulnerability management**; **Software security engineering**;

Keywords

variability, misconfiguration, vulnerability management, security, practitioners, survey, questionnaire

ACM Reference Format:
Richard May, Christian Biermann, Jacob Krüger, and Thomas Leich. 2025. Asking Security Practitioners: Did You Find the Vulnerable (Mis)Configuration?. In *19th International Working Conference on Variability Modelling of Software-Intensive Systems (VaMoS 2025), February 04–06, 2025, Rennes, France.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3715340.3715439

1 Introduction

Ensuring high-quality and reliable software systems is becoming more and more challenging [76], due to the increasing complexity of such systems [20]. In recent years, security has become one of the most important quality attributes related to system reliability [56, 61, 62]. Concurrently, growing configurability [21], interconnections and dependencies between different features and systems [55], as well as system evolution [42] have increasingly contributed to the emergence of misconfigurations, unwanted feature interactions, unexpected bugs, or even system failures [48, 67, 92]. So, misconfigurations tend to cause security vulnerabilities that can be exploited by malicious attackers, usually leading to major violations of a system's confidentiality, integrity, and availability [44, 45, 63].

Typically, security experts are hired to evaluate software systems and their configurations regarding potential risks for vulnerabilities and malicious exploitation (i.e., attack likelihood and impact) [36, 69]. For example, penetration tests are used to identify and assess specific vulnerabilities [19] or attack trees are modeled to trace and understand attack scenarios [43, 46]. Due to the rising number of vulnerabilities and exploits, the experiences gained and required by security experts is growing fast. Consequently, surveying such experts offers a valuable opportunity to synthesize and benefit from their lessons learned and best practices.

Using this opportunity, we asked security practitioners to share their experiences on vulnerabilities caused by configuring (i.e., *configuration vulnerabilities*). Our research goal was to **understand the connections between configurations and vulnerabilities, 🔍 the main causes of these vulnerabilities, ⚠ how prevalent and severe they are, and, 🛡 how they can be treated.** In more detail, we conducted an online questionnaire among 41 security practitioners who already experienced configuration vulnerabilities. There has already been research studying such vulnerabilities (cf. Section 7), for example, May et al. [55] analyzed developer discussions on vulnerable system configurations. Loureiro [48] studied

VaMoS 2025, Rennes, France
© 2025 Copyright held by the owner/author(s).
ACM ISBN 979-8-4007-1441-2/25/02
https://doi.org/10.1145/3715340.3715439

misconfiguration prevention strategies and Dietrich et al. [22] analyzed the human aspects of vulnerable misconfigurations from an operators' perspective. However, to the best of our knowledge, there is currently no comparable study (i.e., thematic focus, sample size) focusing on practitioners' voices on main causes, prevalence, and common mitigation practices.

In detail, our contributions are:

- A comprehensive and practice-oriented overview of configuration vulnerabilities and their properties.
- A discussion of the main causes, severity, and common practices for effectively treating configuration vulnerabilities.
- An open-access repository containing our questionnaire and the anonymous answers of all participants to enable replications and validations of our study.[1]

Through our findings, we aim to shed light on the connections between (mis)configuring and vulnerabilities from a practical perspective. Our results are intended to help researchers and practitioners, especially in the domains of variability and security, learn from the experience of security experts. Overall, our work bridges the gap between theoretical research and real-world practice on this topic.

2 Background

Next, we introduce key background information, specifically system configuring and vulnerability management.

2.1 System Configuring

Modern software has become increasingly configurable to make systems and applications adaptable to specific use cases and associated stakeholder requirements [2, 7]. Typically, configurable systems rely on a set of features (i.e., user-visible functions) that can be enabled, disabled, and combined to create a variety of customized system variants [34, 71, 77]. System configuring may involve different types of configurations, oriented towards specific areas like software configuration (e.g., application settings) [77, 87], network configuration (e.g., routing protocols) [11, 32], storage configuration (e.g., cloud and database system access patterns) [13, 23, 54], server configuration (e.g., load balancer) [8, 30], and security configuration (e.g. security measures) [14, 16]. Furthermore, these configurations have numerous relations and interdependencies. For example, a security configuration typically influences all other types of configurations, which have to ensure compliance with security standards [78].

Developers usually implement system configurations using certain variability mechanisms, such as preprocessor directives [49, 82], dependency injections [80], or feature-oriented programming [7, 15]. Such mechanisms represent variability at the implementation level, while the organization of features, their relationships, and dependencies are usually documented via feature models—the defacto standard at a conceptual level [18, 66, 79]. From a more technical perspective, configuration files store the allowed configurations at the implementation level (e.g., specifying constraints) [11, 75]. Particularly constraints are essential to document, since certain features may depend on each other while other features cannot co-exist (i.e., they are mutually exclusive) [59, 88]. Referring to the latter, faulty configurations (i.e., misconfigurations) may lead to non-trivial problems, ranging from vulnerabilities over bugs to system failures that might cause fatal consequences (e.g., in safety-critical systems) [55, 56]. Accordingly, verifying system configurations is key to ensure reliable and secure systems [1, 72, 83].

2.2 Vulnerability Management

The reliability of modern software systems is increasingly linked to IT security [35, 61]. IT security refers to the practices and measures used to protect all types of software systems, comprising a variety of ways to ensure confidentiality, integrity, and availability [37, 69, 74]. Security measures or patterns that involve concrete strategies to protect systems (e.g., security policies) are typically oriented towards the mitigation of threats and risks. Threats are events with a potential negative impact on a system [36, 37, 69], such as unwanted feature interactions or configuration errors (i.e., misconfigurations) [92]. Such events are triggered mainly by certain unsecured system conditions, typically in the context of vulnerabilities (i.e., system weaknesses) [36, 37]. Although not every vulnerability is necessarily critical and may be considered as an acceptable risk, exploiting vulnerabilities (e.g., via SQL injections [33]) often results in violated security objectives and in the associated security risks [36, 69]. The actual risk is specified based on the likelihood and impact of potential exploits [36, 38], and is usually listed in incident databases (e.g., National Vulnerability Database, Artificial Intelligence Vulnerability Database) that provide scales to classify and rate vulnerabilities [43, 57, 60].

Detecting and managing vulnerabilities is complex and has become more difficult, due to the increasing number of features and their respective configuration options [67, 81, 90]. To prevent vulnerabilities, a systematic security-management process should be performed during development and maintenance [37, 68]. In this context, features, products, and entire product lines should be systematically tested (i.e., verified) to minimize the risk of security-related incidents [54, 83].

3 Methodology

In the following section, we describe our study's goal, research questions, methodology, and conduct (cf. Figure 1).

3.1 Goal and Research Questions

Our main goal was to understand the relations between configuring and vulnerabilities. This included identifying and discussing properties and practices related to detecting and treating vulnerabilities. To accomplish this goal, we formulated the following three Research Questions (RQs):

RQ$_1$ 🔍 **What are causes for configuration vulnerabilities?**
First, we aimed to identify the primary causes that lead to vulnerabilities in the context of configuring a software system. Specifically, our focus was on collecting insights on the triggers to find common patterns and highlight the most critical causes for configuration vulnerabilities.

RQ$_2$ ⚠ **How prevalent and severe are configuration vulnerabilities in the real world?**
Second, we aimed to determine the frequency and severity of configuration vulnerabilities as experienced by security experts. Our goal was to collect data on how often these

[1] https://doi.org/10.5281/zenodo.14007881

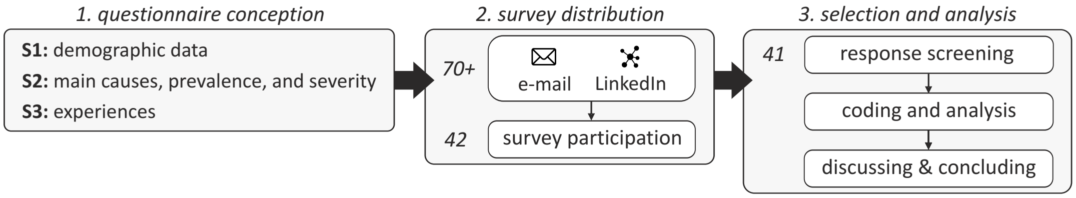

Figure 1: Overview of our research method, with numbers indicating the amount of practitioners involved.

vulnerabilities occur in practice and the impact they can have on systems and operations.

RQ$_3$ 🛡 **What are common practices to treat and prevent configuration vulnerabilities?**
Lastly, we aimed to explore and discuss strategies and practices employed by experts to address and prevent configuration vulnerabilities.

Through our work, we aim to provide a comprehensive overview of the connections between software configuring and vulnerabilities by building on real-world experiences of security experts. Through this overview, we contribute insights for practitioners and researchers that can help to increase their awareness for such vulnerabilities and related practices; providing a helpful means for them to safeguard their software systems and develop new techniques to prevent vulnerabilities.

3.2 Questionnaire Design

We developed our questionnaire based on established guidelines for conducting (online) questionnaires in software engineering [28, 64]. The first and second authors created the questionnaire using Microsoft Forms to build and host the online questionnaire. Iteratively, the third author independently reviewed the questionnaire two times. Besides a general introduction into the topic, we provided a description of how the participants' data was used as well as a consent form. Participants could only take part in the questionnaire if they gave their informed consent and voluntarily. We collected all data in a way that no conclusions could be drawn about the participants or their companies (i.e., anonymized data collection). Furthermore, all data was stored in an encrypted form on a server in Germany.

Overall, our questionnaire involved 20 questions in English and took about 10–15 minutes. The research questions served as basis for the questions and their ordering. In most cases, we used closed-ended questions requiring participants to choose specific items. For these, we typically provided an additional free-text option to add items not listed. Moreover, in some cases, we added the option not to answer a question (i.e., *prefer not to say*) in case participants may have had privacy concerns or constraints by their company. The closed-ended questions involved questions in which we explicitly asked for opinions based on Likert scales (e.g., ranging from *strongly agree* to *strongly disagree*) or number-based scales (e.g., ranging from *1–very rarely* to *10–very frequently*). In addition, we included questions to rank certain items according to their perceived relevance. We further relied on a few open-ended (i.e., free text) questions, typically to obtain more elaborate answers. All but one question were mandatory (i.e., one optional open-ended question for further thoughts on triggers).

Scoping. The target group of our study were IT security practitioners who already experienced configuration vulnerabilities in their work. Besides this background, we did not define any further restrictions. We combined three different distribution channels to promote our online questionnaire, aiming to reach as many participants as possible. These channels included 1) the personal networks of the authors, 2) the LinkedIn network of the first author, and 3) sharing of the invitation through interested participants (i.e., snowballing).

Structure. Our questionnaire involved 20 questions structured into three sections: *demographics* (6 questions), *main causes, prevalence, and severity* (9 questions), and *experiences* (5 questions).

Section 1 (demographics). In the first section, we asked for demographic data, i.e., questions on practical experience, employment, and company. We used this data to assess the responses (e.g., little versus much experience) and to put certain responses into the context of specific participant groups. Precisely, we asked participants to select single-choice options regarding their years of experience (Q_{01}), the country in which they are currently employed (Q_{02}, free text), their company's main industry (Q_{03}), whether the company operates internationally (Q_{04}), and how many employees are in their company (Q_{05}). Finally, they should indicate the area in which they are currently working (Q_{06}, multiple answers). For each question, participants had the opportunity to use a *prefer not to say* option to ensure privacy in addition to anonymizing the data.

Section 2 (main causes, prevalence, and severity). In the second section, we asked for the participants' perceptions about vulnerabilities and configurability in software systems to answer **RQ$_1$** and **RQ$_2$**. First, we asked the participants to describe how vulnerabilities and configuring are connected in their daily work (Q_{07}). After that, we used two questions based on five-level Likert scales to assess the relevance of configuration-related vulnerabilities (Q_{08}) and specific triggers of vulnerabilities (Q_{09}), including an option to add other triggers (Q_{10}, optional free text). The next five questions comprised a ranking of development phases in which vulnerabilities are typically detected (Q_{11}) and an assessment of vulnerability risks (Q_{12}–Q_{15}, ten-level scales). Here, we intentionally oriented the assessment towards established security standards (i.e., ISO/IEC 27000 series [36], NIST Guide for Conducting Risk Assessments [69]), relying on severity, likelihood, impact, and exploits.

Section 3 (experiences). The last section mainly referred to experiences with configuration vulnerabilities. In particular, we asked in which domains the participants experienced the vulnerabilities (Q_{16}, multiple answers), how they became aware of them (Q_{17}, multiple answers), how they fixed (Q_{18}, multiple answers) and prevented (Q_{20}, multiple answers) them, and what the greatest impact of an exploit they experienced was (Q_{19}, free text). We again offered the opportunity to use a *prefer not to say* option. Our objective was to use the data we collected in this section to answer **RQ$_3$** and to complement our answers to **RQ$_1$** and **RQ$_2$**.

3.3 Conduct

We distributed our questionnaire through the channels mentioned above: personal network, LinkedIn, and snowballing. In total, we sent 66 invitations via e-mail to security experts in the authors' personal network. Furthermore, the first author shared a post through their LinkedIn network, with the request to contact us if anyone is interested in participating. Four people reached out via the LinkedIn messenger and we shared a link to the questionnaire with them. We encouraged all participants to share the invitation with other potentially interested experts, which led to an unknown number of further people we reached via this distribution channel. After closing the questionnaire, we received a total of 42 responses.

To analyze the data, we downloaded the Excel spreadsheet created by Microsoft forms. In a first review step, the first author screened the responses and excluded one due to insufficient data quality (e.g., using random characters in mandatory questions to skip them). Consequently, we considered 41 responses for our actual data analysis. The first author then applied open-coding and card sorting to gather recurring patterns and to identify relevant categories with their associated data; particularly for the free-text questions. The first two authors discussed all results through two meetings and iteratively refined them until they reached consensus.

4 Results

In the following, we describe the results of our questionnaire, structured according to the three individual sections.

4.1 Demographics

In Table 1, we present the results related to our participants' demographics. Most of our participants (39 %) have between 6 and 10 years of experience in the area of security (Q_{01}), followed by 29 % with 0 to 5 years, 17 % with 11 to 15 years, and 15 % with more than 15 years. All of our participants are from European countries (Q_{02}), mainly Central-Europe including Switzerland (24 %) and Germany (22 %). Other countries involve, for example, Ukraine (15 %), France, Spain, and Austria (10 % each). The companies for which our participants work (Q_{03}) are quite diverse. For instance, they operate in more traditional IT sectors (34 %), healthcare (20 %), finance (17 %), or manufacturing (15 %). Only 10 % are working on research and education, underpinning the practical orientation of our study. The companies (Q_{04}) typically operate internationally (90 %) rather than nationally (10 %), including (Q_{05}) small- (20 %) to medium-sized companies (53 %) and also larger companies or corporate groups (27 %). Not surprisingly, our participants' employment areas (Q_{06}) mainly relate to development (90 %). Other areas overlap with development, such as research, management, or operations (29 % each). Rarely mentioned other areas include, for example, security consulting.

Table 1: Overview of the responses for demographics (n = 41).

question	answers	responses	
Q_{01}: years of experience	0 – 5	12	29 %
	6 – 10	16	39 %
	11 – 15	7	17 %
	16+	6	15 %
Q_{02}: country	Switzerland	10	24 %
	Germany	9	22 %
	Ukraine	6	15 %
	France	4	10 %
	Spain	4	10 %
	Austria	4	10 %
	Netherlands	3	7 %
	Poland	1	2 %
Q_{03}: main industry	IT	14	34 %
	healthcare	8	20 %
	finance	7	17 %
	manufacturing	6	15 %
	research & education	4	10 %
	other	2	4 %
Q_{04}: internat. operation	yes	37	90 %
	no	4	10 %
Q_{05}: employees	0 – 10	8	20 %
	11 – 100	12	29 %
	101 – 500	10	24 %
	501 – 1000	2	5 %
	1001+	9	22 %
Q_{06}: employment area	development	37	90 %
	research	12	29 %
	management	12	29 %
	operations	12	29 %
	other	3	6 %

4.2 Main Causes, Prevalence, and Severity

The connections between configuration vulnerabilities and our participants' daily work (Q_{07}) are quite diverse. However, we identified five recurring patterns in their responses. Not surprisingly, most participants are typically concerned with more general secure application configuring, such as modeling secure configurations or DevSecOps (54 %). Overall, 24 % have experiences in security-related risk assessments of configurations, 17 % in secure versioning, and 10 % each in dependency checking as well as defensive configuring. We found several more experiences, which were mentioned fewer times, involving variant-richness and its reduction (5 %), countermeasure configuring (2 %), and plugin variety (2 %). Most of our participants strongly agreed (59 %) or agreed (39 %) that configuration vulnerabilities are a relevant topic in practice (Q_{08}). Only one person neither agreed nor disagreed with this statement.

Asking for the relevance of different vulnerability triggers (Q_{09}), our participants mentioned dependencies, outdated software, and inconsistent configurations as most relevant (cf. Figure 2). All other triggers were also supported by participants, but we identified

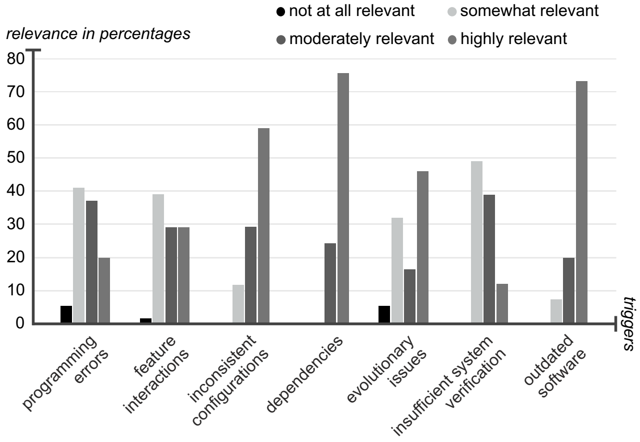

Figure 2: Relevance of vulnerability triggers (Q_{09}).

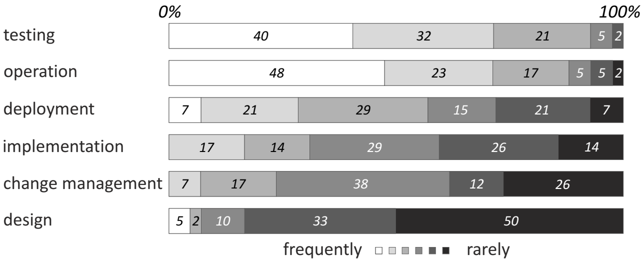

Figure 3: Ranking of the vulnerability occurrence likelihood in different development phases in percentages (Q_{11}).

more disagreements or participants who rated these triggers as only somewhat relevant. This applies in particular to programming errors, feature interactions, evolutionary issues, and insufficient system verification. Interestingly, evolutionary issues highlighted diverging opinions between participants with 46 % considering them highly relevant and 32 % somewhat relevant. Based on the participants' optional notes on triggers (Q_{10}), we found that social engineering is also a relevant trigger (34 %). For example, this relates to compromised development hardware or copying source code with vulnerable configurations from social platforms, such as Stack Overflow. Other triggers that were mentioned include impaired product or process integrity (5 %) and a lack of configuration overview related to configuration complexity (2 %).

According to the participants' ranking, configuration vulnerabilities typically occur or are revealed during testing and operation (Q_{11}, cf. Figure 3). Interestingly, the number of participants who rated operation as most relevant is slightly higher than for testing. In contrast, testing was rated as less relevant by fewer participants than operation. Generally, our participants think that vulnerabilities are less common to occur during the deployment and implementation. Change management and design were usually considered to be the least relevant phases.

As we illustrate in Figure 4, the vulnerability risk ratings (Q_{12}–Q_{15}) with respect to severity (average 7.5), likelihood (average 7.2), impact (average 8), and exploitation (average 7.1) are quite consistent. Regarding severity, most of the practitioners selected 7 (37 %) or 8 (22 %). There is also a great likelihood of configuration vulnerabilities, as implied by 37 % of our participants ranking it as a 7 and 22 % as an 8. Referring to impact, we can see a similar trend (i.e., 34 % refer to 7), with the additional note that 24 % selected a 10. Regarding the likelihood of exploitations, 32 % of our participants referred to an 8 and 29 % to a 7.

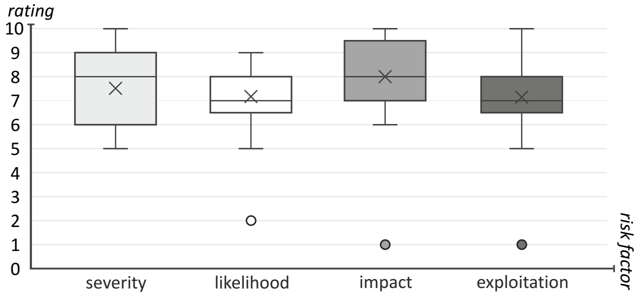

Figure 4: Vulnerability risk rating based on perceived severity, impact, likelihood, and exploitation (Q_{12}–Q_{15}); including interquartile range (boxes), rating range without outliers (outer lines), median (inner lines), average (crosses), and outliers (points) with 0 – lowest rating and 10 – highest rating.

4.3 Experiences

The domains in which our participants experienced configuration vulnerabilities (Q_{16}) are mainly related to server and storage development (93 %), web development (88 %), system and software development (46 %), as well as mobile development (32 %). One participant did not want to answer this question. As we show in Table 2, our participants stated that they usually become aware of configuration vulnerabilities (Q_{17}) through actual security incidents (90 %), word-of-mouth (83 %), regular software testing (76 %), and security scans (73 %). Fewer participants referred to vulnerability disclosures (34 %), new reports (24 %), or penetration testing (15 %).

Typical fixes (Q_{18}) involve configuration changes (85 %), version patches (83 %), risk acceptance (68 %), and implementing countermeasures (49 %). The exploits with the most impact (Q_{19}) vary and heavily depend on whether a participant was involved in the security process of one application or a whole system as well as how many end users were impacted. For example, 29 % of our participants mentioned more than 10,000 impacted users. In contrast, 20 % stated that only more than 100 users were impacted. Interestingly, some participants explicitly mentioned "almost" impacts (5 %) and reputation damage (2 %).

Regarding feasible prevention strategies (Q_{20}), we found (cf. Table 2), not surprisingly, that regularly reviewing and updating configurations as well as conducting regular security scans and/or audits are typical countermeasures (90 % each). Moreover, developing and enforcing security policies and procedures (76 %), integrating security into the development process (51 %), and using configuration management tools (37 %) have been used as prevention strategies by our participants. Interestingly, 12 % of our participants additionally described penetration testing as a prevention strategy.

Table 2: Overview of the awareness triggers (Q_{17}), common fixes (Q_{18}), and prevention strategies (Q_{20}) concerning configuration vulnerabilities.

question	answers	responses	
Q_{17}: awareness	security incident	37	90 %
	word-of-mouth	34	83 %
	software testing	31	76 %
	security scans	30	37 %
	vulnerability disclosure	14	34 %
	news report	10	24 %
	penetration testing	6	15 %
	cyber-threat intelligence	1	2 %
	threat report	1	2 %
Q_{18}: fixes	configuration changes	35	85 %
	patching	34	83 %
	risk acceptance	28	68 %
	countermeasures	20	49 %
	decommission system	1	2 %
	compensating controls	1	2 %
Q_{20}: prevention	regularly reviewing and updating configurations	37	90 %
	regular security scans/audits	37	90 %
	security policies and procedures	31	76 %
	integrating security into the development process	21	51 %
	configuration management tools	15	37 %
	penetration testing	5	12 %
	code reviews	1	2 %

5 Discussion

After describing our results, we now discuss our consequent findings to answer 🔍 **RQ$_1$**, ⚠ **RQ$_2$**, and 🛡 **RQ$_3$**.

5.1 🔍 RQ$_1$: Main Causes

Not surprisingly, our results indicate that configuration vulnerabilities are a relevant topic for practitioners in their daily work, which is further supported by previous research [22, 47, 93]. The responses regarding the main causes for such vulnerabilities point towards dependency issues, outdated software, and inconsistent configurations. We argue that the relevance of dependency issues reflects the challenge of managing complex (cross-)relationships between software systems and their components, with an outdated or insecure dependency easily causing vulnerabilities [55]. Companies may inadvertently introduce configuration vulnerabilities if they fail to regularly update dependencies or thoroughly check the components to integrate for security issues (e.g., when integrating third-party services).

Interestingly, we found that evolutionary issues seem to be a less relevant cause in the perception of practitioners (cf. Figure 2). This is despite evolution and dependency issues being obviously interconnected [3], such as delayed or missing updates [22]. Evolution issues often occur due to configuration drift between development phases, such as implementation, testing, and deployment, or within different branches in the same phase if configurations are not synchronized [25]. Tools that enforce configuration management policies have been suggested to maintain consistency [24], for example, automated dependency management tools to ensure that dependencies are regularly checked for known vulnerabilities [70]. Moreover, the lifecycle phases in which dependencies are integrated seem crucial: typically development and early testing [36, 69]. Ensuring proper dependency management during these phases is key to prevent configuration vulnerabilities in later development phases. Otherwise, the impact of such vulnerabilities can become even worse, as stated by many of our participants who faced such vulnerabilities in the real world (cf. Figure 3).

Other common issues, such as programming errors, social engineering, or feature interactions, were reported fewer times by our participants. In this context, we remark that practitioners may deliberately not report programming errors or social engineering, especially if they were responsible and may feel a certain amount of guilt (i.e., response bias) [22]. So, our findings do not mean that such issues are not relevant. In contrast, human errors are likely to cause misconfigurations [6, 22]. Moreover, the complexity of software systems and their configurable features will likely increase even more in the future. Consequently, we expect even more challenges in handling these complexities, and thus an increased potential for programming errors or unwanted feature interactions causing configuration vulnerabilities [27, 40, 54].

> 🔍 **RQ$_1$: Main Causes:** *Configuration vulnerabilities primarily stem from dependency management issues, outdated software, and inconsistent (cross-)configurations between different software systems and components. Human errors probably have a significant impact in this context.*

5.2 ⚠ RQ$_2$: Prevalence and Severity

Generally, our participants' ratings imply that configuration vulnerabilities are prevalent and severe. Based on our results, we argue that such vulnerabilities are ubiquitous across domains, countries, and company size, further underpinning the need for systematic security engineering and management. More specifically, we can see in Figure 4 that severity, likelihood, impact, and exploitation are all rated in the medium to high ranges (5 to 10). This indicates strong concerns regarding configuration vulnerabilities' potential to cause harm. These insights suggest that current practices, while reasonably effective, may not be sufficiently robust given the constantly evolving threat landscape and refinements of attacks [5].

Our participants' notable agreement on severity implies challenges in maintaining secure and consistent configurations as software complexity increases. Still, there are also a few outliers. These indicate that configuration vulnerabilities do not necessarily have serious consequences in every case. So, it is likely that configuration vulnerabilities are more common than we may anticipate, since they do not have to result in large-scale incidents or exploits [22]. Thus, we argue that configuration vulnerabilities may occur regularly without having critical consequences. Existing research has actually found that such vulnerabilities are often not recognized at all, as it is difficult to detect them; especially if they are silent, and thus do not produce any error messages [94]. If silent vulnerabilities are discovered by malicious actors and actual incidents (i.e., exploits) occur, they usually have a critical impact on the entire software system [47, 93].

Our results further indicate that mistakes in the design, change management, and implementation phases less frequently lead to vulnerabilities compared to the deployment, operation, and test phases. This could suggest that the phases that are primarily related to the design and implementation of the systems are less likely to lead to vulnerable configurations. However, we argue that this is likely a misperception. Most misconfigurations and configuration vulnerabilities originate from these phases [50, 95, 96]. However, they are revealed only during testing in a controlled setting or, in the worst case, during deployment and operation in a less controlled setting with greater impact. In turn, we argue that the awareness for configuration vulnerabilities should be improved already during the design and development of software systems.

> ⚠ **RQ$_2$: Prevalence and Severity:** *Configuration vulnerabilities are acknowledged as prevalent, with high severity and likelihood ratings if detected and exploited. They are typically revealed either in controlled settings (i.e., testing) or less controlled settings with more critical impact (i.e., operation).*

5.3 🛡 RQ$_3$: Treatment and Prevention

Most configuration vulnerabilities our participants experienced were related to server, storage, and web development. This trend is supported by security-related research, which highlights that these domains are particularly prone to vulnerabilities occurring and associated cyber attacks (e.g., cross-site scripting [89]); due to the use of the internet and communication technologies [4, 10, 55]. Surprisingly, more than 83 % of our participants stated that they are aware of vulnerabilities due to security incidents and word-of-mouth between practitioners. This clearly implies more reactive methods being in place regarding security awareness. Consequently, configuration vulnerabilities are apparently often recognized only after they have been exploited or actively discussed within the community (e.g., via Stack Overflow [55]).

Only about half of our participants mentioned that they integrated security into their development process. In contrast, preventive methods through software testing also play an important role (76 %). However, the testing seems less focused on identifying vulnerabilities directly rather than verifying software functionalities. Using security scans or vulnerability disclosures as preventive methods seems underrepresented, or may be not effective enough to identify vulnerabilities at an early stage in the development lifecycle. We argue that the reasons for the limited use of preventive methods may be related to the expensive nature of such methods [9, 22, 91] and the global security workforce gap [17, 41].

Overall, these findings highlight the need to balance reactive and preventive methods, ideally from an early development phase (i.e., design, implementation). For instance, referring to the product-line engineering framework [7], we strongly support the idea to include a security-engineering phase between domain engineering and application engineering [53, 61]. Moreover, as suggested by four participants, defensive configuring may be another feasible strategy at implementation level. For example, this may include defense-in-depth (re)configuration with privilege separation [39].

We can observe a similar trend regarding security fixes. Here, configuration changes and the prompt application of version patches are common reactive methods (each mentioned by more than 83 %). However, these should be supported by preventive methods to anticipate potential vulnerabilities, such as regular threat modeling and automated configuration checks. Accepting risks strategically (68 %) is a short-term, less expensive workaround [31], but clearly highlights the need for long-term strategies. To implement such strategies, enhanced prioritization frameworks and resource allocation can be helpful means.

Surprisingly, there is a quite low percentage of participants that mention the use of configuration management tools in their companies (37 %). We consider such tools as a suitable basis to prevent or identify configuration vulnerabilities. Particularly, we emphasize that there are already proposals in research covering different areas of securely developing highly-configurable software, such as secure product-line engineering [53], vulnerability management [84], and security testing based on feature models [43, 85, 86].

> 🛡 **RQ$_3$: Treatment and Prevention:** *Most practitioners seem to rely on reactive methods like configuration changes to fix configuration vulnerabilities after word-of-mouth or exploitation. Preventive methods like automated (defensive) configuration management are neglected, but should be strengthened in the future to avoid vulnerabilities and exploits from happening.*

6 Threats to Validity

We are aware of potential threats to the internal, external, and construct validity of our work, which we outline next.

Construct Validity. Threats to construct validity refer to the operationalization of variables and measurements (i.e., the questionnaire). The constructs of our questionnaire (e.g., severity of vulnerabilities and their rating) may have lead to misunderstandings among our participants. To mitigate this problem, we based our questions on existing literature and best practices on software security to ensure that the constructs were appropriately operationalized. In particular, the questionnaire and its terms refer to terms, definitions, and scales defined in established security standards, specifically the ISO/IEC 27000 series [36] and the NIST Guide to Conducting Risk Assessments [69]. These are well-established standards security experts are familiar with. Moreover, employing Likert scales (e.g., Q$_{08}$) and closed-ended questions (e.g., Q$_{18}$) may not entirely capture the participants' experiences and perceptions. We mitigated this threat by using open-ended questions (e.g., Q$_{10}$) and free-text options for closed-ended questions (e.g., Q$_{20}$) to allow participants to provide more detailed responses. Nevertheless, we cannot disregard the possibility that participants may have not answered our questions as detailed as possible to save time. We aimed to mitigate this threat by screening all answers at the beginning of the analysis to check whether there were any outliers (i.e., in the responses and the time taken)—leading to one exclusion.

Internal Validity. One threat related to the internal validity is selection bias, as we invited our participants primarily through our personal networks and LinkedIn. As a result, our participants may not represent our target population of software security professionals appropriately, potentially leading to biased results. To address this threat, we carefully identified security experts and sought to broaden our target population by encouraging participants to distribute the questionnaire to additional professionals within their networks. Another threat is response bias: participants may have provided socially desirable answers rather than truthful responses. This issue occurs particularly frequently in the context of sensitive topics [58] including vulnerability management [22]. To address this threat, we included options like *prefer not to say* (e.g., Q_{03}) and ensured that all responses are anonymized (i.e., collecting no personal data) to encourage honest answers. Generally, there may be several threats related to how we interpreted free-text responses (e.g., Q_{10}), which we aimed to mitigate by involving multiple researchers in the analysis process.

External Validity. Again, our participants may not represent the entire population of software security professionals, which limits the generalizability of our results. To address this, we used multiple distribution channels to reach a broader audience and encouraged participants to share the survey. Still, all of our participants are from Europe, and thus their experiences are likely influenced by regional regulations and practices. Moreover, we are aware that a larger number of participants would have strengthened the generalization of our findings. However, we argue that 41 participants and a return rate of 59 % are feasible values to collect and derive reliable results seeing the specific target group of experts we needed to invite [28]. In addition, more than 70 % of our participants have at least six years of experience in the field of security, which increases our confidence in the results.

7 Related Work

Vulnerability Causes and Treatment. There are various works on misconfiguration vulnerability triggers, their exploitation, and how to treat them, in particular related to web and server configuring. For instance, Loureiro [48], Martins et al. [52], and Xu and Zhou [92] surveyed risks and general treatment strategies oriented towards configuration vulnerabilities (e.g., misconfigured web servers). Furthermore, there are several papers presenting tools to detect misconfigurations leading to vulnerabilities. For example, Li et al. [47] proposed their tool ConfVD to identify vulnerabilities caused by SQL injections, while Eshete et al. [26] focused on a tool called Confeagle to detect vulnerabilities in the context of denial-of-service and session-hijacking attacks. Another line of research focuses on exploiting configuration vulnerabilities. For instance, Sulatycki and Fernandez [81] and Haimed et al. [29] present threat patterns to exploit misconfigurations, providing insights in how to build and configure applications more securely. Lastly, researchers focus on evaluations, for instance, Moura et al. [65] reproduced misconfigurations related to DNS services and evaluated their severity. In contrast to such works, we aimed to capture the state-of-practice to identify temporary problems and opportunities for improvements.

Practitioners' Voices. There are only few studies related to practitioners' experiences regarding configuration vulnerabilities. Manfredi et al. [51] studied the usability of security reports related to TLS miconfiguration patching based on a user study with 62 students. As part of their study on misconfigurations in open-source Kubernetes manifests, Rahman et al. [73] conduced interviews with nine developers to validate misconfigurations and obtain insights into the vulnerability detection processes. Bhuiyan et al. [12] presented a survey of 51 developers to identify vulnerability discovery strategies. Interestingly, they found that adapting configurations to trigger misconfigurations is a promising method. Moreover, May et al. [55] analyzed 651 Stack Overflow posts related to configuration vulnerabilities, providing a broader overview of concerns developers face.

The work closest to ours is the one by Dietrich et al. [22], who investigated system operators' perspectives on misconfigurations that impact system security. They combined qualitative interviews with a quantitative survey with more than 200 practitioners. Although they did not involve security practitioners, some of their results are in line with ours (i.e., misconfigurations are a common issue). However, generally Dietrich et al. have another focus, which is much more on business operations, such as setting budget restrictions for incident management. So, although the related work is somewhat similar to ours and may provide partly overlapping findings, we argue that they cover another body of knowledge that is out of our scope. We focus on a different topic based on security experts' opinions, leading to novel, practice-related insights.

8 Conclusion

In this paper, we reported a questionnaire with 41 security experts on configuration vulnerabilities. Our research goal was to understand the main causes, prevalence, severity, and treatments of such vulnerabilities. We found that configuration vulnerabilities are prevalent and severe. They primarily stem from dependency issues, outdated software, and inconsistent (cross-)configurations likely influenced by human errors. Such vulnerabilities are typically identified during testing (i.e., controlled settings) or, in the worst case, during the deployment and operation (i.e., less controlled settings). Overall, we found that there is a clear awareness for configuration vulnerabilities and their impact. However, due to the common use of reactive methods instead of more preventive ones, these vulnerabilities are often discovered too late and lead to a larger negative impact. Note that our results show several threats to validity (cf. Section 6), which might affect the validity of our findings.

Although researchers have proposed methods that partially address preventive methods, the transfer of these methods seems rather limited. Even if the higher costs of using such practices may be difficult to influence, we believe that they are highly valuable and that there are major knowledge gaps regarding their availability as well as use. In our future work, we aim to address these gaps to provide a comprehensive overview of existing methods to mitigate configuration vulnerabilities. For this purpose, we also aim to support the transfer of preventive research methods (e.g., defensive configuring) into practice.

References

[1] I. Abal, C. Brabrand, and A. Wasowski. 2014. 42 variability bugs in the Linux kernel: A qualitative analysis. In *International Conference on Automated Software Engineering (ASE)*. ACM, 421–432.

[2] I. Abal, J. Melo, Ş. Stănciulescu, C. Brabrand, M. Ribeiro, and A. Wąsowski. 2018. Variability bugs in highly configurable systems: A qualitative analysis. *ACM Transactions on Software Engineering and Methodology* 26, 3 (2018), 1–34.

[3] P. Abate, R. Di Cosmo, R. Treinen, and S. Zacchiroli. 2012. Dependency solving: A separate concern in component evolution management. *Journal of Systems and Software* 85, 10 (2012), 2228–2240.

[4] M. Abomhara and G. M. Køien. 2015. Cyber security and the internet of things: Vulnerabilities, threats, intruders and attacks. *Journal of Cyber Security and Mobility* (2015), 65–88.

[5] S. AboulEla, N. Ibrahim, S. Shehmir, A. Yadav, and R. Kashef. 2024. Navigating the cyber threat landscape: An in-depth analysis of attack detection within IoT ecosystems. *AI* 5, 2 (2024), 704–732.

[6] M. Alicea and I. Alsmadi. 2021. Misconfiguration in firewalls and network access controls: Literature review. *Future Internet* 13, 11 (2021), 283–298.

[7] S. Apel, D. Batory, C. Kästner, and G. Saake. 2013. *Feature-oriented software product lines*. Springer.

[8] M. Arlitt and C. Williamson. 2004. Understanding web server configuration issues. *Software: Practice and Experience* 34, 2 (2004), 163–186.

[9] A. Asen, W. Bohmayr, S. Deutscher, M. González, and D. Mkrtchian. 2019. Are you spending enough on cybersecurity? *Boston Consulting Group* (2019).

[10] A. Bamrara. 2015. Evaluating database security and cyber attacks: A relational approach. *The Journal of Internet Banking and Commerce* 20, 2 (2015), 1–16.

[11] R. Beckett, A. Gupta, R. Mahajan, and D. Walker. 2017. A general approach to network configuration verification. In *Conference of the ACM Special Interest Group on Data Communication*. ACM, 155–168.

[12] F. A. Bhuiyan, J. Murphy, P. Morrison, and A. Rahman. 2021. Practitioner perception of vulnerability discovery strategies. In *International Workshop on Engineering and Cybersecurity of Critical Systems (EnCyCriS)*. IEEE, 41–44.

[13] M. Bilal, M. Canini, and R. Rodrigues. 2020. Finding the right cloud configuration for analytics clusters. In *ACM Symposium on Cloud Computing*. ACM, 208–222.

[14] D. Bringhenti, G. Marchetto, R. Sisto, and F. Valenza. 2023. Automation for network security configuration: State of the art and research trends. *Comput. Surveys* 56, 3 (2023), 1–37.

[15] M. Calder, M. Kolberg, E. H. Magill, and S. Reiff-Marganiec. 2003. Feature interaction: A critical review and considered forecast. *Computer Networks* 41, 1 (2003), 115–141.

[16] B. Chung, J. Kim, and Y. Jeon. 2016. On-demand security configuration for IoT devices. In *Conference on Information and Communication Technology Convergence (ICTC)*. IEEE, 1082–1084.

[17] W. Crumpler and J. A. Lewis. 2022. *Cybersecurity workforce gap*. JSTOR.

[18] K. Czarnecki, P. Grünbacher, R. Rabiser, K. Schmid, and A. Wąsowski. 2012. Cool features and tough decisions: A comparison of variability modeling approaches. In *Working Conference on Variability Modelling of Software-Intensive Systems (VaMoS)*. ACM, 173–182.

[19] D. Dalalana Bertoglio and A. F. Zorzo. 2017. Overview and open issues on penetration test. *Journal of the Brazilian Computer Society* 23 (2017), 1–16.

[20] V. Damasiotis, P. Fitsilis, and J. F. O'Kane. 2018. Modeling software development process complexity. *International Journal of Information Technology Project Management* 9, 4 (2018), 17–40.

[21] S. Dass and A. Siami Namin. 2021. Reinforcement learning for generating secure configurations. *Electronics* 10, 19 (2021), 1–19.

[22] C. Dietrich, K. Krombholz, K. Borgolte, and T. Fiebig. 2018. Investigating system operators' perspective on security misconfigurations. In *Conference on Computer and Communications Security (CCS)*. ACM, 1272–1289.

[23] S. Duan, V. Thummala, and S. Babu. 2009. Tuning database configuration parameters with ituned. *VLDB Endowment* 2, 1 (2009), 1246–1257.

[24] C. Ebert, G. Gallardo, J. Hernantes, and N. Serrano. 2016. DevOps. *IEEE Software* 33, 3 (2016), 94–100.

[25] M. M. Emmanuel, M. N. Ibrahim, et al. 2015. Automatic synchronization of common parameters in configuration files. *Journal of Software Engineering and Applications* 8, 04 (2015), 192.

[26] B. Eshete, A. V., K. Weldemariam, and M. Zulkernine. 2013. Confeagle: Automated analysis of configuration vulnerabilities in web applications. In *International Conference on Software Security and Reliability (QRS)*. IEEE, 188–197.

[27] A. M. Gamundani and L. M. Nekare. 2018. A review of new trends in cyber attacks: A zoom into distributed database systems. In *IST-Africa*. IEEE, 1–9.

[28] A. N. Ghazi, K. Petersen, S. V. R. Reddy, and H. Nekkanti. 2018. Survey research in software engineering: Problems and mitigation strategies. *IEEE Access* 7 (2018), 24703–24718.

[29] I. B. Haimed, M. Albahar, and A. Alzubaidi. 2023. Exploiting misconfiguration vulnerabilities in Microsoft's Azure Active Directory for privilege escalation attacks. *Future Internet* 15, 7 (2023), 226.

[30] Z. He, K. Li, and K. Li. 2021. Cost-efficient server configuration and placement for mobile edge computing. *Transactions on Parallel and Distributed Systems* 33, 9 (2021), 2198–2212.

[31] A. Heyerdahl. 2022. Risk assessment without the risk? A controversy about security and risk in Norway. *Journal of Risk Research* 25, 2 (2022), 252–267.

[32] Z. B. Houidi and D. Rossi. 2022. Neural language models for network configuration: Opportunities and reality check. *Computer Communications* 193 (2022), 118–125.

[33] M. Humayun, M. Niazi, N. Z. Jhanjhi, M. Alshayeb, and S. Mahmood. 2020. Cyber security threats and vulnerabilities: A systematic mapping study. *Arabian Journal for Science and Engineering* 45, 4 (2020), 3171–3189.

[34] M. S. Iqbal, R. Krishna, M. A. Javidian, B. Ray, and P. Jamshidi. 2022. Unicorn: Reasoning about configurable system performance through the lens of causality. In *European Conference on Computer Systems (EuroSys)*. ACM, 199–217.

[35] ISO/IEC 25010 2011. *Systems and software engineering – SQuaRE - system and software quality*. Standard. ISO.

[36] ISO/IEC 27000 2018. *Information technology – security techniques – information security management systems*. Standard. ISO.

[37] ISO/IEC 27001 2013. *Information security management systems – requirements*. Standard. ISO.

[38] ISO/IEC 27005 2022. *Information security, cybersecurity and privacy protection – Guidance on managing information security risks*. Standard. ISO.

[39] T. Jaeger. 2016. Configuring software and systems for defense-in-depth. In *Workshop on Automated Decision Making for Active Cyber Defense (SafeConfig)*. ACM, 1–1.

[40] P. Jamshidi, M. Velez, C. Kästner, N. Siegmund, and P. Kawthekar. 2017. Transfer learning for improving model predictions in highly configurable software. In *International Conference on Software Engineering for Adaptive and Self-Managing Systems (SEAMS)*. ACM, 31–41.

[41] C. A. Jordan. 2022. *Exploring the cybersecurity skills gap: A qualitative study of recruitment and retention from a human resource management perspective*. Northcentral University.

[42] J. Jürjens, K. Schneider, J. Bürger, F. P. Viertel, D. Strüber, M. Goedicke, R. Reussner, R. Heinrich, E. Taşpolatoğlu, M. Konersmann, et al. 2019. *Maintaining security in software evolution*. Springer.

[43] A. Kenner, S. Dassow, C. Lausberger, J. Krüger, and T. Leich. 2020. Using variability modeling to support security evaluations: Virtualizing the right attack scenarios. In *Working Conference on Variability Modelling of Software-Intensive Systems (VaMoS)*. ACM, 1–9.

[44] A. Kenner, R. May, J. Krüger, G. Saake, and T. Leich. 2021. Safety, security, and configurable software systems: A systematic mapping study. In *Systems and Software Product Line Conference (SPLC)*. 148–159.

[45] R. A. Khan, S. U. Khan, H. U. Khan, and M. Ilyas. 2021. Systematic mapping study on security approaches in secure software engineering. *IEEE Access* 9 (2021), 19139–19160.

[46] A.-M. Konsta, A. L. Lafuente, B. Spiga, and N. Dragoni. 2024. Survey: Automatic generation of attack trees and attack graphs. *Computers & Security* 137 (2024), 103602.

[47] S. Li, W. Li, X. Liao, S. Peng, S. Zhou, Z. Jia, and T. Wang. 2018. Confvd: System reactions analysis and evaluation through misconfiguration injection. *IEEE Transactions on Reliability* 67, 4 (2018), 1393–1405.

[48] S. Loureiro. 2021. Security misconfigurations and how to prevent them. *Network Security* 2021, 5 (2021), 13–16.

[49] K. Ludwig, J. Krüger, and T. Leich. 2019. Covert and phantom features in annotations: Do they impact variability analysis?. In *Systems and Software Product Line Conference (SPLC)*. ACM, 218–230.

[50] I. Maganha, C. Silva, and L. M. D. F. Ferreira. 2019. The layout design in reconfigurable manufacturing systems: A literature review. *The International Journal of Advanced Manufacturing Technology* 105 (2019), 683–700.

[51] S. Manfredi, M. Ceccato, G. Sciarretta, and S. Ranise. 2021. Do security reports meet usability? Lessons learned from using actionable mitigations for patching tls misconfigurations. In *International Conference on Availability, Reliability and Security (ARES)*. ACM, 1–13.

[52] S. L. Martins, F. M. Cruz, R. P. Araújo, and C. M. R. Silva. 2024. Systematic literature review on security misconfigurations in web applications. *International Journal of Computers and Applications* (2024), 1–13.

[53] R. May, C. Biermann, A. Kenner, J. Krüger, and T. Leich. 2023. A product-line-engineering framework for secure enterprise-resource-planning systems. In *International Conference on ENTERprise Information Systems*. Elsevier, 1–8.

[54] R. May, C. Biermann, J. Krüger, G. Saake, and T. Leich. 2022. A systematic mapping study of security concepts for configurable data storages. In *Systems and Software Product Line Conference (SPLC)*. ACM, 108–119.

[55] R. May, C. Biermann, X. M. Zerweck, K. Ludwig, J. Krüger, and T. Leich. 2024. Vulnerably (mis)configured? Exploring 10 years of developers' Q&As on Stack Overflow. In *Working Conference on Variability Modelling of Software-Intensive Systems (VaMoS)*. ACM, 112–122.

[56] R. May, J. Gautam, C. Sharma, C. Biermann, and T. Leich. 2023. A systematic mapping study on security in configurable safety-critical systems based on product-line concepts. In *International Conference on Software Technologies (ICSOFT)*. SciTePress, 217–224.

[57] R. May, J. Krüger, and T. Leich. 2024. SoK: How artificial-intelligence incidents can jeopardize safety and security. In *International Conference on Availability, Reliability and Security (ARES)*. ACM, 1–12.

[58] A. McCormac, D. Calic, M. Butavicius, K. Parsons, T. Zwaans, M. Pattinson, et al. 2017. A reliable measure of information security awareness and the identification of bias in responses. *Australasian Journal of Information Systems* 21 (2017), 1–12.

[59] J. Meinicke, C.-P. Wong, C. Kästner, T. Thüm, and G. Saake. 2016. On essential configuration complexity: Measuring interactions in highly-configurable systems. In *International Conference on Automated Software Engineering (ASE)*. ACM, 483–494.

[60] P. Mell, K. Scarfone, and S. Romanosky. 2006. Common vulnerability scoring system. *IEEE Security & Privacy* 4, 6 (2006), 85–89.

[61] D. Mellado, E. Fernández-Medina, and M. Piattini. 2010. Security requirements engineering framework for software product lines. *Information and Software Technology* 52, 10 (2010), 1094–1117.

[62] D. Mellado, H. Mouratidis, and E. Fernández-Medina. 2014. Secure tropos framework for software product lines requirements engineering. *Computer Standards & Interfaces* 36, 4 (2014), 711–722.

[63] O. Mesa, R. Vieira, M. Viana, V. H. S. Durelli, E. Cirilo, M. Kalinowski, and C. Lucena. 2018. Understanding vulnerabilities in plugin-based web systems: An exploratory study of Wordpress. In *Systems and Software Product Line Conference (SPLC)*. ACM, 149–159.

[64] J. S. Molléri, K. Petersen, and E. Mendes. 2016. Survey guidelines in software engineering: An annotated review. In *International Symposium on Empirical Software Engineering and Measurement (ESEM)*. ACM, 1–6.

[65] G. C. M. Moura, S. Castro, J. Heidemann, and W. Hardaker. 2021. TsuNAME: Exploiting misconfiguration and vulnerability to DDoS DNS. In *Internet Measurement Conference (IMC)*. ACM, 398–418.

[66] D. Nešić, J. Krüger, S. Stănciulescu, and T. Berger. 2019. Principles of feature modeling. In *Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE)*. ACM, 62–73.

[67] A. Nhlabatsi, R. Laney, and B. Nuseibeh. 2008. Feature interaction: The security threat from within software systems. *Progress in Informatics* 5, 75 (2008), 1.

[68] NIST SP 800-154 2016. *Guide to data-centric system threat modeling.* Standard. National Institute of Standards and Technology.

[69] NIST SP 800-30r1 2012. *Guide for conducting risk assessments.* Standard. National Institute of Standards and Technology.

[70] I. Pashchenko, D.-L. Vu, and F. Massacci. 2020. A qualitative study of dependency management and its security implications. In *Conference on Computer and Communications Security (CCS)*. ACM, 1513–1531.

[71] K. Pohl, G. Böckle, and F. Van Der Linden. 2005. *Software product line engineering: Foundations, principles, and techniques.* Springer.

[72] H. Post and C. Sinz. 2008. Configuration lifting: Verification meets software configuration. In *International Conference on Automated Software Engineering (ASE)*. IEEE, 347–350.

[73] A. Rahman, S. I. Shamim, D. B. Bose, and R. Pandita. 2023. Security misconfigurations in open source kubernetes manifests: An empirical study. *ACM Transactions on Software Engineering and Methodology* 32, 4 (2023), 1–36.

[74] S. Samonas and D. Coss. 2014. The CIA strikes back: Redefining confidentiality, integrity and availability in security. *Journal of Information System Security* 10, 3 (2014).

[75] M. Santolucito, E. Zhai, R. Dhodapkar, A. Shim, and R. Piskac. 2017. Synthesizing configuration file specifications with association rule learning. *ACM on Programming Languages* 1 (2017), 1–20.

[76] A. M. Satpute, J. Priya, J. Mishra, and S. Anilkumar. 2022. Software reliability modelling and application in software development life cycle. *International Journal of Advances and Current Practices in Mobility* 5, 123 (2022), 1577–1584.

[77] M. Sayagh, N. Kerzazi, B. Adams, and F. Petrillo. 2018. Software configuration engineering in practice: Interviews, survey, and systematic literature review. *IEEE Transactions on Software Engineering* 46, 6 (2018), 646–673.

[78] K. Scarfone and P. Mell. 2010. The common configuration scoring system (CCSS): Metrics for software security configuration vulnerabilities. *NIST Interagency Report* 7502 (2010).

[79] I. Schaefer, R. Rabiser, D. Clarke, L. Bettini, D. Benavides, G. Botterweck, A. Pathak, S. Trujillo, and K. Villela. 2012. Software diversity: State of the art and perspectives. *International Journal on Software Tools for Technology Transfer* 14 (2012), 477–495.

[80] M. Seemann and S. van Deursen. 2019. *Dependency injection principles, practices, and patterns.* Simon and Schuster.

[81] R. Sulatycki and E. B. Fernandez. 2015. Two threat patterns that exploit security misconfiguration and sensitive data exposure vulnerabilities. In *European Conference on Pattern Language of Programs*. IEEE, 1–11.

[82] R. Tartler, D. Lohmann, J. Sincero, and W. Schröder-Preikschat. 2011. Feature consistency in compile-time-configurable system software: Facing the Linux 10,000 feature problem. In *European Conference on Computer Systems (EuroSys)*. ACM, 47–60.

[83] T. Thüm, S. Apel, C. Kästner, I. Schaefer, and G. Saake. 2014. A classification and survey of analysis strategies for software product lines. *ACM Computing Surveys* 47, 1 (2014), 1–45.

[84] Á. J. Varela-Vaca, D. Borrego, M. T. Gómez-López, R. M. Gasca, and A. G. Márquez. 2023. Feature models to boost the vulnerability management process. *Journal of Systems and Software* 195 (2023), 1–22 pages.

[85] Á. J. Varela-Vaca, R. M. Gasca, J. A. Carmona-Fombella, and M. T. Gómez-López. 2020. AMADEUS: Towards the autoMAteD secUrity teSting. In *Systems and Software Product Line Conference (SPLC)*. ACM, 1–12.

[86] Á. J. Varela-Vaca, D. G. Rosado, L. E. Sánchez, M. T. Gómez-López, R. M. Gasca, and E. Fernandez-Medina. 2021. CARMEN: A framework for the verification and diagnosis of the specification of security requirements in cyber-physical systems. *Computers in Industry* 132 (2021), 1–14.

[87] S. Wang, B. Luo, W. Shi, and D. Tiwari. 2016. Application configuration selection for energy-efficient execution on multicore systems. *J. Parallel and Distrib. Comput.* 87 (2016), 43–54.

[88] W. Wang, S. Jian, Y. Tan, Q. Wu, and C. Huang. 2022. Representation learning-based network intrusion detection system by capturing explicit and implicit feature interactions. *Computers & Security* 112 (2022), 102537.

[89] S. J. Weamie. 2022. Cross-site scripting attacks and defensive techniques: A comprehensive survey. *International Journal of Communications, Network and System Sciences* 15, 8 (2022), 126–148.

[90] Y. Wei, X. Sun, L. Bo, S. Cao, X. Xia, and B. Li. 2021. A comprehensive study on security bug characteristics. *Journal of Software: Evolution and Process* 33, 10 (2021), e2376.

[91] P. Wooderson and D. Ward. 2017. *Cybersecurity testing and validation.* Technical Report. SAE Technical Paper.

[92] T. Xu and Y. Zhou. 2015. Systems approaches to tackling configuration errors: A survey. *Comput. Surveys* 47, 4 (2015), 1–41.

[93] Z. Yin, X. Ma, J. Zheng, Y. Zhou, L. N. Bairavasundaram, and S. Pasupathy. 2011. An empirical study on configuration errors in commercial and open source systems. In *Symposium on Operating Systems Principles (SOSP)*. ACM, 159–172.

[94] J. Zhang, R. Piskac, E. Zhai, and T. Xu. 2021. Static detection of silent misconfigurations with deep interaction analysis. *Proceedings of the ACM on Programming Languages* 5 (2021), 1–30.

[95] Y. Zhang, H. He, O. Legunsen, S. Li, W. Dong, and T. Xu. 2021. An evolutionary study of configuration design and implementation in cloud systems. *International Conference on Software Engineering (ICSE)*, 188–200.

[96] S. Zhou, X. Liu, S. Li, W. Dong, X. Liao, and Y. Xiong. 2016. Confmapper: Automated variable finding for configuration items in source code. In *2016 IEEE International Conference on Software Quality, Reliability and Security Companion (QRS-C)*. IEEE, 228–235.