An Exploratory Study on the Engineering of Security Features

Kevin Hermann*, Sven Peldszus*, Jan-Philipp Steghöfer†, Thorsten Berger*‡
*Ruhr University Bochum, Germany †XITASO GmbH, Germany ‡Chalmers | University of Gothenburg, Sweden

***Abstract*—Software security is of utmost importance for most software systems. Developers must systematically select, plan, design, implement, and especially, maintain and evolve security features—functionalities to mitigate attacks or protect personal data such as cryptography or access control—to ensure the security of their software. Although security features are usually available in libraries, integrating security features requires writing and maintaining additional security-critical code. While there have been studies on the use of such libraries, surprisingly little is known about how developers engineer security features, how they select what security features to implement and which ones may require custom implementation, and the implications for maintenance. As a result, we currently rely on assumptions that are largely based on common sense or individual examples. However, to provide them with effective solutions, researchers need hard empirical data to understand what practitioners need and how they view security—data that we currently lack. To fill this gap, we contribute an exploratory study with 26 knowledgeable industrial participants. We study how security features of software systems are selected and engineered in practice, what their code-level characteristics are, and what challenges practitioners face. Based on the empirical data gathered, we provide insights into engineering practices and validate four common assumptions.**

***Index Terms*—Security Feature, Software Security, Secure Software Development, Security by Design, Developer Study**

I. Introduction

Considering security in every development phase is a critical [1] yet challenging task. To secure a software system, developers must engineer its security features. Security features, such as encryption or access control, address security concerns by mitigating an attack or protecting assets, such as personal data, to prevent malicious actions of an attacker [1]. Security features must be selected according to the system's security objectives, for example, according to the CIA triad [2] or the Parkerian Hexad [3]. Following the literature, the engineering of security features should start with the identification of security requirements [4], [5], [6], continue with secure design [7], [8], [9], [10] and implementation [11], and end with validation and verification of the realized features [12], [13], [14], [15].

The research community invested substantial effort in developing security technologies and engineering techniques to keep pace with adversarial actors who continuously discover new vulnerabilities. Most developers, however, are not security experts [16], [17], [11]. While libraries are available for a wide range of security features, developers still have to write a notable amount of security-critical code to configure and integrate them [18], [19], [20], which often leads to insecure implementation of security features despite the use of security libraries [21]. Further difficulties arise when developers must implement custom security features for which no library is available.

Research so far has focused on individual aspects of the development of security features in isolation [20], [22], [23], [11], [21], [24], [19], such as investigating the usability of cryptographic APIs and its impact on security in controlled experiments [21]. *However, we still lack a holistic understanding of how developers engineer security features in practice and what the code-level characteristics of security features are.* The absence of hard empirical data has lead to *individual, anecdotal views on the engineering of security features in software projects*, and *research commonly makes assumptions that might not always be accurate*, as a recent study shows [16].

To improve software security and base research on valid assumptions, we need a better understanding of how practitioners engineer security features. In particular, we need to know on which basis security features are selected and why some are not implemented. Furthermore, we require an understanding of how security features are engineered in practice, the challenges involved, as well as the characteristics of security features. We identified the following research questions to address this gap:

RQ$_1$: What is the developer perspective on security features, and what influences security feature selection?

RQ$_2$: How is the engineering of security features embedded into the software development lifecycle?

RQ$_3$: What are code-level characteristics of security features?

RQ$_4$: What challenges arise in engineering security features?

We present an exploratory study using semi-structured interviews with 26 industry experts. We provide the interview guide, aggregated data, and the evaluation scripts to derive conclusions in our replication packge [25]. The interviews provide insights into practices for engineering security features in software systems. We reason about four common assumptions in security research, combining insights from our interviews and observations from existing studies.

We confirm several commonly held assumptions while revealing significant room for improvement. In particular, we confirm that developers lack knowledge related to detailed security concepts, but are still able to realize security features on a practical level. As a result, many developers struggle with secure configuration of security libraries. Contrary to popular belief, security-by-design techniques are used in practice, but mainly in regulated domains. In general, however, most companies try to follow security-by-design principles, but are forced to prioritize system functionality over security features.

1558-1225/25/$31.00 ©2025 IEEE
DOI 10.1109/ICSE55347.2025.00184

II. Background and Related Work

Outside of the security domain, several definitions of "feature" focus on the distinction between functionalities [26], [27] and products [28], or customer visibility [29], [30] and value [31]. Related to this, *Security features* provide functionalities that address security issues by preventing attacks on a software system [1]. They either realize non-functional security requirements [12] or functional ones [32], e.g., to handle authentication, access control, cryptography, and other aspects of software security [33]. To this end, security features aim to resist, detect or recover from attacks to a software system [34], or increase its resistance, tolerance, or resilience against them [35].

To realize security features, Schumacher et al. [36] list security patterns for different stages of the software development lifecycle such as validation, threat assessment, or risk assessment. An interview study on security testing revealed that practitioners recommend forming dedicated and specialized teams to handle security testing [15]. Oyetoyan et al. [22] compared the security engineering of two agile organizations, finding that knowledge transfer is essential to increase security but must be actively approached. One of the main challenges is the significant difference in granularity at which security features can be considered [37], i.e., abstract planning of which information needs to be protected by which type of security feature, as opposed to detailed security feature characteristics such as their configuration. While previous work investigated the extent of implemented security features in open source projects through a keyword search [38], no study investigated the code-level characteristics of security features, such as scattering [39] or tangling [40].

Ryan et al. [16] performed a systematic mapping study to identify common assumptions in security research containing contradictory findings. Unlike our study, they do not generate new empirical data, but systematically relate existing data to identify misconceptions, serving as a primary motivation for our study. In 2009, Werlinger et al. [41] identified human, organizational, and technological challenges of IT security management related to security experience, prioritization, and tools. Our study reports on which methods practitioners employ to overcome such challenges, and which challenges remain. Klivan et al. [42] found that developers of open source projects rarely communicate their security practices to other contributors, but expect them to utilize sensible ones. However, it is unclear if this also applies to companies as well.

III. Methodology

We first selected assumptions security researchers make about security engineering in practice, followed by interviews with practitioners to gain insights on the engineering process.

A. Selection of Assumptions

To identify common assumptions in security research, we contacted 39 security researchers from 14 institutions and asked them about *"assumptions about security in practice they typically encounter in their research."* We received responses both in discussion and in writing from 22 researchers from nine institutions in various fields. We grouped similar responses and triangulated them with related work to derive and detail the assumptions about security in practice. We then formulated research questions to validate these assumptions.

TABLE I: Interview participants and their characteristics

ID	Role	Experience	Domain
I1	Software Architect	5 years	Logistics
I2	Project Manager	8 years	Quantum Computing
I3	Software Developer	5 years	Antivirus
I4	Project Manager	22 years	Automotive Security
I5	Security Engineer	4 years	Insurance, Public Sector
I6	Software Developer	7 years	Real Estate
I7	Security Engineer	8 years	Insurance, Banking, Retail
I8	Software Developer	7 years	Logistics
I9	Software Developer	10 years	Insurance
I10	Software Architect	6 years	Medical, Automotive, Robotics
I11	Software Developer	5 years	Systems Management
I12[1]	Security Engineer	6 years	Automotive Security
I13[1]	Security Engineer	4 years	Automotive Security
I14[1]	Security Engineer	6 years	Automotive Security
I15	Software Developer	11 years	Medical, Automotive
I16	Software Developer	8 years	Manufacturing
I17	Software Developer	4 years	Medical, Insurance
I18	Software Developer	5 years	Logistics
I19	Software Architect	9 years	Banking
I20	Software Developer	15 years	Cloud Computing
I21	Project Manager	20 years	Cloud Computing
I22	Software Architect	8 years	Insurance
I23	Software Developer	10 years	Machine Learning, Cloud Computing
I24	Software Architect	15 years	Infrastructure, Aerospace, Defense
I25	Software Developer	7 years	Manufacturing
I26	Software Developer	3 years	Manufacturing

[1] I12–I14 were part of one focus group

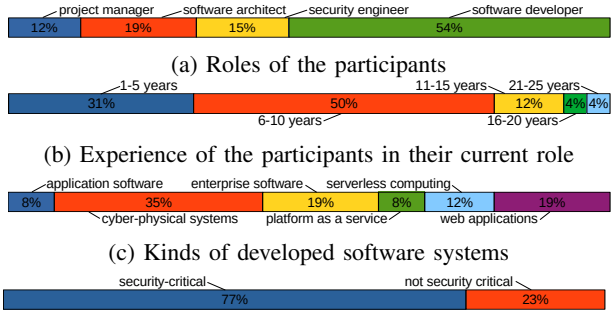
B. Interview Study

We conducted semi-structured interviews [43], [44] with domain experts to obtain insights about engineering security features.

Recruitment. We recruited developers, software architects, security engineers, and project managers involved in at least one phase of the engineering process of security features. We recruited participants through our personal networks, and by asking computer science associations and participants for referrals until reaching saturation [45]. We treated saturation as being able to understand the meaning of each code in the coding process [46]. After interviewing 17 participants, we reached saturation, which is consistent with the experience of Hennink et al. [46] of 16 to 24 interviews. To mitigate the risk of saturation by chance, we conducted further interviews.

Participants. Table I and Fig. 1 summarize the participants. We interviewed 26 professionals from 15 companies covering projects across various domains. Seven participants work as consultants for external companies. Six participants have experience from multiple domains. Some companies work on a single product, others build customized software. All interviewed project managers have a strong background in software engineering, while one has a security background. On average, participants have 8.4 years of experience in their role, ranging from 3 to 22 years. Participants work on diverse types of software systems, with cyber-physical systems, enterprise software, and web applications being the most prominent. We conducted one interview as a focus group with three participants.

Interview Design. Four researchers iteratively drafted the interview questions by proposing various questions according



(a) Roles of the participants

(b) Experience of the participants in their current role

(c) Kinds of developed software systems

(d) Criticality of the developed systems

Fig. 1: Overview of participants for the interviews

Introduction

1. Background (10 questions): What is your professional background? What kind of Software do you develop?

We provided our definition of security features

2. Security Features (1 question): What security features do you develop?

3. Characteristics of security features (5 questions): How large are security features in your codebase? How do security features spread over it?

4. Selection of security features (5 questions): How do you choose what security features to engineer? On what basis do you prioritize security features?

5. Engineering of security features (10 questions): Which security features are implemented through libraries? How do you test security features?

6. Challenges related to security features (8 questions): What are the challenges of engineering security features? In which security features do you detect the most bugs?

Outro

Fig. 2: Outline of the interview guide with sample questions.

to the research questions. The questions were then thoroughly discussed among the researchers, merged, and refined multiple times to form the interview guide. Figure 2 shows the resulting outline of our interview guide with sample questions.

(1) We began by asking the interviewees about their professional background, the software they develop, and the importance of security in their projects. **(2)** We then explained, that "a security feature is a feature implementing a specific kind of security mechanism to mitigate an attack or protect an asset such as personal data.", and asked them to list security features important to them, **(3)** as well as their code-level characteristics. Then, we inquired about how they handle and communicate security features at each development stage, focusing on **(4)** selection, prioritization, **(5)** design, implementation, testing, and maintenance. **(6)** Finally, we discussed challenges in engineering security features, and ways to overcome them.

Analysis. We recorded the interviews with consent of the participants. The first author transcribed the recordings using

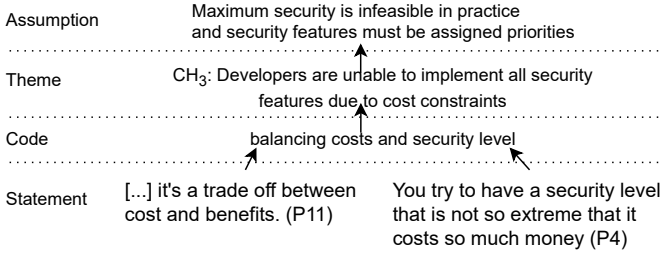


Fig. 3: Illustration of the coding process

Whisper [47], run locally, and manually corrected inconsistencies between transcripts and recordings. Interviews ranged in length from 34 to 95 minutes. We stored all recordings and transcripts only on computers within our institution, only the authors had access to them, and we deleted them after analysis.

We applied open coding [48] to extract themes from the transcribed interviews using MAXQDA [49]. Figure 3 illustrates the coding process. The first author coded the transcripts to extract key information from the answers to the interview questions. We then merged codes to a theme, which we mapped to the assumptions. To ensure consistency, the first and second author discussed the code book in detail.

For each theme, we indicate the number of interviews in which it was substantiated or contradicted and in which it did not come up (**X agree**; **Y disagree**; **Z not mentioned**). We also investigated the influence of roles or domains on each derived theme. We discuss these explicitly where observed.

Replication Package. We provide our interview guide as well as our aggregated results in our replication package [25].

IV. Common Assumptions in Security Research

We identified 4 common assumptions that researchers frequently make and count how often they occurred in their responses.

Assumption 1: Ordinary developers lack knowledge to securely engineer security features (10 occurences)

Many academics make assumptions about developers' security knowledge and ability to perform certain tasks, e.g., security planning or risk assessment. On the one hand, developers are rarely considered to be security experts and, therefore, need tools to help them plan and implement security features [17], [11]. On the other hand, developers are assumed to be capable of performing complex security tasks related to security feature engineering [50], [51]. Such conflicting views on the assumption that *"all developers know how to code securely"* is also reflected in previous work [16]. However, we still lack a detailed understanding of developers' actual security knowledge.

Assumption 2: (Model-based) security-by-design techniques are not used in practice (7 occurences)

Various model-based security-by-design techniques have been proposed [52], e.g., threat modeling based on data flow diagrams [8], [53], [54] or various extensions to design languages such as UMLsec [7]. The literature frequently reports benefits of the practical application of model-based security-by-design techniques [55], [56], [57], [58]. Yet, researchers frequently acknowledge that the adoption of such techniques requires models that may not be used in practice [59], [54], [60], [61]. A recent empirical study [52] shows that most design-time security languages extend design models, and therefore, assume their use. On the other hand, Gorschek et al. show that design models are rarely used in practice [62].

Assumption 3: Security libraries are complicated to use, resulting in many vulnerabilities (5 occurences)

Developers have access to a large set of libraries that include security features. However, complex cryptographic APIs are commonly assumed to be a main reason for insecure code.

Developers often perceive these libraries as too low-level and to require a lot of manual implementation to be useful [23]. As a result, they use security libraries in insecure ways, leading to security issues [21]. To counteract this, tools such as CogniCrypt [18] analyze the use of cryptographic APIs to detect insecure usages. While studies draw a concrete picture for cryptographic APIs, their insights are often generalized. However, we lack concrete insights on what generally challenges practitioners when using security features provided by libraries.

Assumption 4: Maximum security is infeasible in practice and security features must be assigned priorities (7 occurences)

It is often assumed that organizations will make security a top priority and apply research results [16]. In practice, however, resource constraints often force organizations to make difficult trade-offs between security and functionality. Since complete security is neither feasible nor desirable due to these constraints, prioritization between security features seems unavoidable [63], but has not been assessed empirically.

V. Selection of Security Features and Influencing Factors (RQ_1)

First, we focus on what participants consider a security feature and how security features are selected in practice.

SSF_1: Security features are generally perceived as important, but less important than functionality
22 agree; 4 disagree; 0 not mentioned

Participants view security features as important for their projects, but customers often favor functional features. This creates challenges in prioritizing security features over functional ones, particularly with limited resources.

Six interviewees report frequently discussing the need to assign a higher priority to security features to mitigate costs when vulnerabilities are exploited with customers. Customers, however, rarely take these security risks seriously, neglecting the security of the system which can cause vulnerabilities.

Seven participants mention customers not seeing security features as providing business value, thus prioritizing functional features over them: *"functionality, performance, security. That's the order."* (I6, Developer, Real Estate). Two interviewees explain that in experimental domains, e.g., quantum computing, providing functionality is the prerequisite to securing it.

SSF_2: Only few security features are of immediate concern
16 agree; 6 disagree; 4 not mentioned

Participants mentioned important security features, including cryptography, authentication, and authorization, while features such as validation and logging are less prominent.

Table II lists the security features mentioned in at least one interview, while also relating them to the OWASP Top 10 vulnerabilities. We gathered the security features mentioned in step 3 of our interview guide in Fig. 2. We also considered security features mentioned at a later stage of the interview, separating them from the initial response to identify potential patterns. This distinction provides an indication that developers are immediately concerned with the security features *authentication*, *authorization*, and *cryptography*.

TABLE II: Security features mentioned more than once

Security Feature	Times Named			Relation to OWASP Top 10
	total	in step 3	later	
Authentication	18	**14**	4	A07:Authentication
Cryptography	17	**12**	5	A02:Cryptography
Authorization	16	**10**	6	A01:Broken Access Control
Validation	13	5	**8**	A03:Injection, A08:Data Integrity
Logging	8	2	**6**	A09:Security Logging
Security Monitoring	7	**6**	1	A09:Security Monitoring
Session Management	3	1	**2**	A01:Access Control, A07:Authentication

While participants state to *"definitely have authentication/authorization services in all of our software"* (I7, Security Engineer, Insurance, Banking, Retail), they perceive cryptography as for domains that handle personal information, such as real estate or banking, but less so in other domains such as logistics: *"Encryption actually is a very rarely used part in the applications I'm working with [...] For example, if an external HTTPS API was used, then that's encryption, but you actually never get in touch with that because you just import the API library and do the API request and what it does internally, you never care about as a developer"* (I8, Developer, Logistics).

Data validation, *"is always required at the system boundaries"* (I1, Architect, Logistics) when systems receive input, especially in security patterns such as distrustful decomposition or zero trust architecture. Similarly, eight participants highlight the importance of logging, e.g., of login attempts, accesses, or transactions, for accountability, i.e., to detect and understand security incidents. Developers also implement log filtering to ensure that sensitive data is not leaked through logs.

Security monitoring to detect attacks early—reducing cost for mitigation and recovery—and session management are less frequently mentioned. Some participants considered architectural patterns or refer to development activities as security features, which are not features within the system itself, such as firewalls around the company's intranet or information flow control, to prevent secrets in publicly accessible variables or objects.

Six vulnerabilities of the OWASP Top 10 can be directly related to the mentioned security features, although *Broken Access Control*, as well as *Cryptography and Authentication Failures* are of more immediate concern. Note that the remaining four vulnerabilities, such as Insecure Design, are not directly related to a security feature, but loosely related to multiple ones.

SSF_3: Security feature selection is based on experience
16 agree; 0 disagree; 10 not mentioned

The selection of security features varies, ranging from unstructured to well-structured processes for capturing threats. The selection of security features is always based on experience, either from past projects or observations of competitors.

Participants report, feature selection *"is based on years of working with customers and knowing what's needed"* (I21, Architect, Banking). Four participants mention, when not structuring security engineering processes, companies select security features based on those from other systems.

When security features stem from dedicated processes, such as threat modeling (cf. $EPSF_3$), concrete security features are still discussed and selected based on experience, as noted in two interviews. Required structured risk assessment processes in regulated domains [64] are mentioned by three participants.

Finally, seven participants state that development teams discuss which security features are needed in team discussions and workshops which include developers and security experts.

SSF_4: Developers perceive security as an extensive domain and therefore specialize in certain security features
15 agree; 2 disagree; 9 not mentioned

Developers specialize in specific security features, as knowing all details about every security feature is infeasible, and act as "lighthouses" to help others in understanding these features.

More than half of the interviewees claim that it is infeasible to learn about every single security feature, *"because you can't be an expert in all things"* (I26, Developer, Manufacturing). Instead, they focus on specific security features for which they become experts. To reduce overhead, companies sometimes offer training or workshops on various security topics. Typically, developers are intrinsically motivated to learn about security, becoming experts (often called "lighthouses") who assist other developers when working with specific security aspects.

SSF_5: Customers are a main driver when companies need to assign priorities to security features
18 agree; 3 disagree; 5 not mentioned

Except for absolutely necessary security features that must always be implemented, customers are a primary factor in decisions on extending them. Explicitly requested features that benefit most customers receive the highest priority.

Once a baseline of security features is implemented (cf. $EPSF_4$), the priority shifts to what benefits most customers or is actively requested, as revealed in fourteen interviews. Two participants mentioned that internal discussions and the impact on multiple customers help assign priorities to these features.

One interviewee explains that once security features have been selected, they are often realized in a logical order based on their interdependencies: *"if you don't have authentication, authorization doesn't make sense. You can enable authorization, but disable authentication. So I can just tell the system, 'hi, my name is [name]' and it trusts me. But obviously, in that case, I can circumvent authorization checks."* (I21, Manager, Cloud).

Security feature selection is driven by experience, customer demand, and requirements from regulations. Few organizations follow structured processes to select security features, relying on customers to provide requirements and prioritization. However, they rarely provide these and favor the implementation of functional features. The most pressing security features are authentication and authorization. Validation and logging is common, in particular for architectural planning and accountability. To handle all these security features, developers become experts in a small subset of them.
— Selection of Security Features (RQ1) —

VI. Engineering Security Features (RQ_2)

In RQ_1, we covered factors that influence the selection of security features. Now, we focus on how security features embedded into software development processes.

$EPSF_1$: Projects usually lack initial security requirements
19 agree; 0 disagree; 7 not mentioned

Projects in unregulated domains rely on customers to provide security requirements or attack scenarios. They are rarely able to provide these, however, impeding security feature selection. Laws and regulations in certain domains are an important source of security requirements at project start.

Many participants reported that customers, as non-experts, do not request specific security features. Instead, *"They just have the idea that they have to protect their asset"* (I4, Manager, Automotive), as described in five interviews. This makes planning difficult due to often unclear concrete attack scenarios. For example, one interviewee (I16, Developer, Manufacturing) mentions a customer who *"wanted security"* for a medical device *"but were unclear of which actual attack scenarios"* exist, as the device neither communicates nor stores data over a network and is only accessible by a doctor. In such a case, determining attack scenarios is challenging as neither data nor the system itself is at risks.

Only participants working in domains regulated by laws and standards, such as automotive or banking, report starting projects with a well-defined list of required security features. There, initial requirements of security features often stem from relevant laws and standards. Since these requirements must be fulfilled for compliance, developers must assign a higher priority to these security features than in unregulated domains. This reasoning was also provided by participants working in several domains with different levels of security criticality.

$EPSF_2$: Companies try to follow security-by-design, but only rarely implement specific processes
25 agree; 1 disagree; 0 not mentioned

Companies consider security as early as possible, but most do not follow a security-by-design process. Still, processes are needed to address previously neglected security concerns.

All but one participant report that their companies follow some form of security-by-design process. Most companies consider security features early in the development process, which is perceived as more effective than addressing them at the end. Nine interviewees explicitly state that delaying security considerations increases the risk of problems during implementation.

Six participants state that security is often discussed alongside functional features. Of these, three emphasize the necessity of frequent discussion of security features while realizing functional features, while the other three suggest adding security features into sprint backlogs alongside functional features. One participant (I23, Developer, ML, Cloud) mentions that they do *"sprints, that are just security focused"* to realize or update security features that have been neglected for a longer period.

Limited by cost, not every company is able to prioritize security features over functional features early on, as reported

in ten interviews. Still, these companies build systems that implement a minimal set of required security features.

EPSF$_3$: Threat modeling is widely used in practice
17 agree; 8 disagree; 1 not mentioned

Many companies employ structured processes based on threat modeling to identify threats to their systems and plan appropriate countermeasures. However, the extent and used methodology varies between the individual companies.

Threat modeling was identified as the primary security design process, in which developers *"evaluate which threats are there"* (I7, Security Engineer, Insurance, Banking, Retail) determine necessary security features, as described by 17 interviewees. Threat modeling is also often employed in later development stages when requirements change. However, threat modeling is often approached in a lightweight way rather than using model-based approaches such as STRIDE [8]. Still, formal model-based threat modeling techniques are mentioned by five participants working in regulated domains such as the automotive or medical industry. The identified threats play an important role in selecting security features (cf. SSF$_3$).

When participants strongly rely on external service providers such as cloud services to handle their security-critical infrastructure, they limit their threat modeling to identify whether the provided security features are sufficient. In these cases, they inherit the threats from these service providers, but the responsibility for providing security against them is shifted to the providers—outsourcing risk management is a common practice in standards such as ISO 21434 [64]. Still, additional threats and thus additional security features may be considered. Participants working with customers who do not take security seriously are less likely to apply threat modeling.

EPSF$_4$: A baseline of standard security features is often implemented but rarely extended and customized
15 agree; 0 disagree; 11 not mentioned

Companies implement a baseline set of security features required by almost every system, which are only re-engineered as requirements change or vulnerabilities are detected.

Interviewees commonly report that *"there's a set of minimum security features the developers have to implement"* (I21, Manager, Cloud) which is usually derived from company guidelines or regulations. This baseline typically includes authentication, authorization, validation, logging, and occasionally cryptography. Once implemented, they are later re-engineered to meet changing requirements, which is often challenging as *"it's always a struggle to search for a library that fulfills the new given requirements"* (I9, Developer, Insurance).

EPSF$_5$: Security features are implemented by ordinary developers as part of their everyday duties
23 agree; 0 disagree; 3 not mentioned

Developers implement security features in the parts of the system they are responsible for and consult specialized developers for advice rather than having them implement these features. Security-specific task assignments are the exception.

All but three interviewees report that ordinary developers without a security background are responsible for implementing security features as a normal part of their tasks. Developers are expected to identify and implement relevant security features according to the system's security design. However, some software architects and project managers report they assign security features not to every developer and *"always have an eye on what they are doing"* (I22, Architect, Insurance).

Most companies train developers to become security experts in certain areas (cf. SSF$_4$), allowing them to handle specific security features without needing a dedicated security engineer. Security engineers with deep security expertise, who aid developers in implementing security features, are only mentioned in four interviews. Developers proactively seek advice on implementing security features. Additionally, they are usually tasked with reviewing code written by developers.

EPSF$_6$: Companies rely on secure practices of frameworks and service providers
20 agree; 0 disagree; 6 not mentioned

Frameworks such as AWS, Azure, or Spring, with best practices and defaults, are widely used and perceived by many to be sufficient for implementing secure systems.

Three fourths of the participants explicitly report building their systems based on popular frameworks such as Spring or those provided by cloud providers, e.g., AWS or Azure. These frameworks often provide secure defaults and best practices, assisting developers in securing software systems. Following these guidelines helps companies, particularly those with a lower security focus, implement security-by-design principles with low effort. However, even though participants trust them, one interviewee states that *"you never know what's going on there"* but thinks that they adhere to best practices – *"at least that's what I hope"* (I2, Manager, Quantum).

EPSF$_7$: Security features are often tested, but the majority of testing techniques is not security-specific
25 agree; 1 disagree; 0 not mentioned

Security features are tested like any other feature, with specific vulnerabilities testing practices rarely applied. Since test coverage is a widely used metric, they are often well covered. Penetration tests are adopted to a notable degree.

Thirteen participants indicate that security features are tested similarly to functional feature through unit, regression and integration tests. Libraries are usually trusted, but larger companies occasionally test the security of libraries before using them.

Less than half of the interviews mention security-specific testing techniques. Pentesting is the most common technique, mentioned in nine interviews, usually reported to be cumbersome and ineffective. Five respondents explain that code reviews or audits by security experts verify the correct implementation and usage of security features.

When explicitly testing security features, three participants report that developers write test cases to verify that unintended behavior does not occur. For example, access control features are tested to ensure that unauthorized access is properly denied.

In this case, developers require a *"mindset of 'I want to break things'"* (I10, Architect, Medical, Automotive, Robotics). However, tests for abuse scenarios are rarely mentioned.

EPSF$_8$: Security feature maintenance is often limited to small fixes and updating dependencies
17 agree; 2 disagree; 7 not mentioned

Developers perceive maintaining security features as easy, involving quick fixes or updating libraries. Neglecting maintenance leads to accumulating problems and increased effort.

Maintenance mainly involves updating security libraries, often supported by automated dependency checks. Five interviewees mention that *"it is usually enough to just update the libraries"* (I8, Developer, Logistics). Two interviewees rely on external service providers for implementing and updating security features, eliminating the need for maintenance. Therefore, developers perceive maintaining security features as easy, since tasks such as identifying the code location require low effort.

Due to a lack of security metrics, two interviewees noted that maintenance needs often become apparent only after a vulnerability is discovered or exploited. Four interviewees report that maintenance effort is high when neglected for long periods, as vulnerabilities accumulate and familiarity fades.

EPSF$_9$: Security features are not communicated systematically and with varying degrees of granularity
21 agree; 2 disagree; 3 not mentioned

The development phase and stakeholder experience determine how security features are communicated. Planning requires a high-level view, implementation low-level technical details.

Eleven interviewees noted that security features are typically considered at a high level during planning, while twelve stated they must consider details during implementation. Five interviewees explain developers mainly discuss security concepts, rarely discussing low-level details. Two developers claim that the lack of implementation details is manageable because *"most of the time you can really work on a high-level abstraction, thanks to the frameworks"* (I11, Developer, Systems Management).

Selecting appropriate communication granularity strongly depends on the stakeholders involved in a discussion as their knowledge about security features differs. Five interviewees state that communication to managers or customers is mostly kept on a high level as they lack technical understanding.

Security features are considered in all phases of software development, and communicated at different levels of granularity. Even though companies do not employ specific security design processes to model or test security features, most companies employ threat modeling to identify threats. Initially, security features are realized based on what is commonly done and continuously adapted during development until meeting the requirements. Selecting service providers supports companies in minimizing security considerations with secure defaults and built-in security mechanisms. Finally, maintenance tasks for security features are perceived to require little effort.
— Engineering Process of Security Features (RQ2) —

VII. Characteristics of Security Features (RQ$_3$)

We now focus on the code-level characteristics of security features, including size (lines of code in relation to other code), tangling (the extent to which a feature intersects/interacts with other features) and scattering (the extent to which a feature is distributed over the codebase) of security features.

CSF$_1$: Security features are implemented using frameworks and libraries
25 agree; 0 disagree; 1 not mentioned

Developers rely on external and internal security frameworks or libraries to avoid implementing vulnerable security features. Still, developers write custom code for security features when no suitable library exists or to meet needs.

Participants reported adhering to best practices by using security frameworks and libraries to implement security features. Primary motivations are to avoid errors, save costs, and leverage widely used and tested third-party implementations. For instance, cryptographic algorithms are considered difficult to understand and thus hard to implement. Still, *"you can find a library for everything"* (I26, Developer, Manufacturing).

However, developers from companies focusing on security solutions occasionally need to write their own code, typically when cutting-edge security features lack libraries: *"Some functions, like for example, if you have some elliptic curves that are not supported, then we implement them on our own, because they do not come with a library yet"* (I4, Manager, Automotive).

Despite existing logging libraries, logging often involves custom implementation, since developers need to consider what events to log. The implementation effort is then not related to creating logging functionalities, but to identifying the correct places in the system to add log statements, which leads to more usage than developers perceive for other security features. Additionally, many cases of input validation are perceived as specific to the concrete application, requiring a custom implementation tailored to the application-specific needs.

CSF$_2$: Compared to functional features, the code for security features is minimal
14 agree; 4 disagree; 8 not mentioned

Since developers rely on libraries, code for security features is limited to their integration using glue code (code that allows features to interoperate) and data transformations.

Our interviews indicate that the size of features in terms of lines of code and thus the effort required to implement them varies, but is kept to a minimum. Eleven interviewees mention building wrappers around security features to integrate them into the system. Seven of these claim that this practice facilitates the usage of security libraries by abstracting low-level details. Still, three interviewees explain that writing wrappers for unfamiliar security features requires additional effort.

In summary, when developing security features, developers primarily write code to pass data to an API of a security library in the correct format or to convert returned data. For example, two participants mentioned writing wrappers to receive authentication tokens. In two other interviews,

developers explained that creating API wrappers also simplifies data encryption: *"If I have to implement the API wrapper on my own, then I also don't implement the encryption part"* (I8, Developer, Logistics). To this end, understanding how to integrate security libraries incurs most effort when writing security features.

CSF_3: Functionality of security features is perceived to be well encapsulated, but usages scattering over the codbase
20 agree; 3 disagree; 3 not mentioned

While the core functionality of security features is implemented in a modular way and can be well-encapsulated, their use is scattered throughout the codebase.

Interviewees report that security features can be well encapsulated and reused throughout the software system. One interviewee emphasizes: *"you should definitely encapsulate your security code, especially if it's cryptographic code, [...] you don't want to write the same call all over your code, because then if you change something at one place, [...] and then forget about all the others, then your code will have a vulnerability again."* (I10, Architect, Medical, Automotive, Robotics). Four participants state that this approach works well for authentication and authorization, while two confirm this practice for cryptography.

CSF_4: Security features are perceived as being tangled only to a low degree
16 agree; 4 disagree; 6 not mentioned

Security feature code is perceived as being independent and well separated from other features. Developers feel working on security features does not impact other features.

Interviewees generally stated that functional features only interact with security features at specific, well-separated locations. Five participants report that even though functional features interact with security features, changing the security features does not have a direct impact on behavior. Still, three participants state that they *"try to keep it to a bare minimum, but sometimes you cannot avoid it completely"* (I4, Manager, Automotive). The only scenario mentioned by interviewees where security feature code is directly woven into the implementation of the functional feature is logging. One participant perceived a stronger tangling in mainly in the use of cryptographic features such as certificates in the authentication process.

Developers primarily use libraries for security features, mostly writing wrapper code for integration. The functionality of most security features is perceived to be locally encapsulated and well separated from other features' code. Still, the use of these security features is scattered throughout the code base. Counterintuitive to this scattering, security features are perceived as not being tangled with each other.

Code-level characteristics of Security Features (RQ3)

VIII. Challenges of Engineering Security Features (RQ_4)

Finally, we focus on challenges the participants face in engineering security features and how they overcome them.

CH_1: Non-experts lack foundational knowledge of security
18 agree; 0 disagree; 8 not mentioned

Insufficient security knowledge among customers and developers hinders security feature engineering, necessitating that companies build a foundation in security knowledge.

To effectively realize security features, *"one requires a basis of knowledge of the security domain"* (I1, Architect, Logistics), which stakeholders usually lack. Seven interviewees criticize customers' lack of knowledge and concern for security features, with six noting that they only consider them after issues occur.

Four interviewees highlight challenges developers face when implementing security features for the first time, often unsure if they have done so correctly. One developer reports that library documentation rarely shows examples of misuse or common mistakes, which is *"information I need so that I can write code in a secure way"* (I10, Architect, Medical, Automotive, Robotics). Consequently, developers are often unsure how security features could be bypassed, because they do not know *"what the attackers do"* (I19, Architect, Banking).

To mitigate this challenge, four participants reported that companies organize workshops and team discussions to improve the general understanding of security features and where people acting in different roles share their knowledge. However, we observed that many participants were unfamiliar with concepts such as the CIA Triad, preventing them from describing protection goals, therefore highlighting a knowledge gap.

CH_2: Developers often underestimate the effort of engineering security features
11 agree; 0 disagree; 15 not mentioned

Lack of knowledge makes it harder for developers to accurately estimate the effort for engineering security features, often leading to complications during implementation.

Developers lack an understanding of security features and their underlying concepts. Before implementing security features, they must first research about them to obtain a basic understanding. Therefore, two interviewees report it is not possible for them to reliably estimate the effort of realizing security features.

Even after researching security features, developers lack practical experience and conceptual knowledge that is required to effectively implement advanced security features. Therefore, they often *"initially underestimate the complexity because you only learn about a certain feature when you actually started working on it"* (I21, Manager, Cloud). In total, six participants report to frequently experience such underestimation, and it often takes longer than expected to realize sophisticated features. One participant mentions an impact of the development phase, claiming that DevOps security can be estimated well, while estimating implementation effort is challenging.

CH_3: Developers are unable to implement all security features due to cost constraints
15 agree; 1 disagree; 10 not mentioned

Developers struggle to implement all security features at an "optimal" level due to cost and effort. They struggle to prioritize security features over others to achieve full security.

Seven interviews reveal that developers are not able to implement all security features within a project. They mention it is impractical to aim for full coverage as *"you will never be 100% secure"* (I24, Developer, Manufacturing). Instead, developers *"try to have a security level that is not so extreme,"* and *"not too expensive to actually implement"* (I4, Manager, Automotive). One explains, some security features are difficult to prioritize over other features, since they vary in effort.

In addition to implementation effort, participants identify learning details as an overhead cost. This challenge was raised by two participants who explain that developers need to learn about security features before they are able to properly implement them, since they are *"not experts in security, but come from another domain"* (I2, Manager, Quantum).

Most participants see a barrier to prioritizing security features over functional features, *"because you have to make the whole thing* [the project] *a reality before you find focus on things like that"* (I2, Manager, Quantum). Three participants state that time constraints prevent them from prioritizing security over functionality. Customers are an additional impediment, primarily concerned with functionality, which rarely includes security.

CH_4: Developers struggle with estimating the side effects of changes to security features

9 agree; 0 disagree; 17 not mentioned

As security features are used in many parts of the system, estimating how local changes of security features impact the overall system is hard, since changes to security features are assumed to not impact the functional behavior of the system.

One-third of participants report that estimating the side effects of changes is a major challenge when changing security features. When implementing a new feature, *"it's really hard to see the entire picture [...] and what effects that might have"* (I6, Developer, Real Estate). One of the participants reports that they needed to change access permissions for a new feature, but could not predict how this would affect other existing functionality. Particularly when functionality which other parties rely on is involved, communication barriers impede the analysis of change impact of security features. In the worst case, this can lead to outages, e.g., if changes in the authentication services were not properly addressed by consuming parties.

Stakeholders, especially developers, lack knowledge, to estimate the effort required to develop security features. Additionally, developers feel overwhelmed learning details about security. They struggle to prioritize security features over others and fully implement them because of cost and time constraints. Finally, it is challenging to estimate the impact of changes of security features, since they propagate through the system.

Challenges of Engineering Security Features (RQ4)

IX. Discussion and Common Assumptions

We now discuss the findings of our study.

A. Assumptions in relation to practice

The goal of our study is validating assumptions made in security research. Table III relates each theme to the assumptions.

Assumption 1: Ordinary developers lack knowledge to securely engineer security features (Partially Accurate)
Studies such as Ryan et al. [16] challenge the underlying assumption that developers' perceptions of whether security is needed in their work environment are accurate. Instead, they conclude that developers who are not well-informed about software security may be poor judges of when it is required. Corresponding observations are also contained in our study, but reveal that developers often lack conceptual knowledge, yet still have an idea on how to realize security feature practically.

In practice, security features are indeed developed by ordinary developers (cf. $EPSF_5$), who require experience to effectively select security features (cf. SSF_3). However, our interviews revealed a significant lack in security-related knowledge (cf. CH_1) and a tendency to underestimate the effort related to securely engineering security features (cf. CH_2). The lack of knowledge could be one reason for a lack of security-specific test cases (cf. $EPSF_7$). Particularly, the implementation of detailed security functionalities or their configuration is more complicated, since a system appears to be working correctly, but may lack important implementation details or usage at specific places of the system (cf. CH_4). Therefore, specializing in individual security features helps in increasing security and reducing the overhead of learning about all of them (cf. SSF_4). The effectiveness of individual experts and the resulting knowledge transfer was also observed in other studies [22]. Relying on service providers that provide secure defaults additionally assists developers in handling the knowledge gap (cf. $EPSF_6$).

Assumption 2: (Model-based) security-by-design techniques are not used in practice (Not Accurate)
While researchers from the security-by-design community assume that (model-based) threat modeling techniques are used in practice, other researchers challenge this assumption and paint them as practically irrelevant. We lack an understanding of whether (model-based) security-by-design techniques are truly perceived as irrelevant in practice.

In the interviews, we find that almost everyone perceives security as important (cf. SSF_1) and incorporates security into their design, although the specifics vary considerably (cf. $EPSF_2$). Particularly threat modeling is widely adopted in practice (cf. $EPSF_3$). Companies often consider relying on external service providers or frameworks providing the required functionality through secure defaults (cf. $EPSF_6$), acknowledging the inheritance of potential threats that may need to be monitored (cf. $EPSF_8$). Even when interviewees stated that security is not a priority, we learned that there are "security sprints" or security features that have been developed from the start (cf. $EPSF_2$). Security-by-design is, therefore, widely adopted in practice, although sometimes not perceived as such, and usually much less systematically than techniques such as UMLsec [7] or STRIDE [8]. However, concrete security requirements are usually lacking at project start, leading to challenges, especially in unregulated domains (cf. $EPSF_1$). Finally, we found that security features are not systematically communicated (cf. $EPSF_9$) or captured in models (cf. $EPSF_3$) throughout the development process. Our interviews highlight

TABLE III: Mapping between themes and assumptions (✓ supports the assumption, X opposes the assumption)

	SSF_1 (importance)	SSF_2 (features)	SSF_3 (experience-based)	SSF_4 (specialization)	SSF_5 (prioritization)	$EPSF_1$ (requirements)	$EPSF_2$ (by-design)	$EPSF_3$ (threat-model)	$EPSF_4$ (baseline)	$EPSF_5$ (developers)	$EPSF_6$ (externals)	$EPSF_7$ (testing)	$EPSF_8$ (maintenance)	$EPSF_9$ (communication)	CSF_1 (libraries)	CSF_2 (code)	CSF_3 (modularity)	CSF_4 (separation)	CH_1 (knowledge)	CH_2 (effort)	CH_3 (cost)	CH_4 (side-effects)	participant perception[1] supporting	opposing	neither
Assumption 1: Ordinary developers lack knowledge to securely engineer security features (PARTIALLY ACCURATE)			X	✓						X	✓	✓			X		X		✓	✓		✓	4	2	**20**
Assumption 2: (Model-based) security-by-design techniques are not used in practice (NOT ACCURATE)	X					✓	X	X			X		X	✓									1	**18**	7
Assumption 3: Security libraries are complicated to use, resulting in many vulnerabilities (PARTIALLY ACCURATE)													X		X	(X)	(X)	(X)		✓			1	**19**	6
Assumption 4: Maximum security is infeasible in practice and security features must be assigned priorities (ACCURATE)	✓	✓			✓			✓	✓											✓	✓		**25**	0	1

[1] If a 2/3 majority of a participant's agreements or disagreements with the themes related to an assumption support or oppose this assumption, the participant is assumed to support or oppose the assumption.

a need for existing techniques to be more practical for effective usage in the development process.

Assumption 3: Security libraries are complicated to use, resulting in many vulnerabilities (PARTIALLY ACCURATE) Insecure use of cryptographic APIs has been identified as a common reason for insecure code. Tools such as CogniCrypt [18] address this by providing automated checks and code generation for usages of cryptographic APIs. It is a common conception that the documentation of libraries is a major factor in secure use, albeit often considered inadequate [65].

In general, participants explain that they use libraries wherever possible (cf. CSF_1). In contrast to common assumptions [65], [66], [67], the interviewees of our study generally were satisfied with the quality of provided APIs and their documentation, claiming they can be easily integrated with minimal effort (cf. CSF_2), well separated from other parts of the system (cf. CSF_3 and CSF_4), and easily updated when required (cf. $EPSF_8$). However, studies have shown that even if developers use only libraries, the written code can be insecure due to their incorrect usage [21]. We learned from the interviews that incorrect use is often difficult to identify (cf. CH_4), leading to a discrepancy between perceived ease and practical challenge. Ultimately, developers perceive implementing security features as easy, while it is difficult in reality (cf. CH_2). Developers need a general understanding of how a library works internally, but do not want or can go deeply into the algorithmic details. Consequently, developers may oversimplify implementations, potentially introducing vulnerabilities. Although, we asked participants in which security features they identify the most bugs, the answers did not reveal a pattern or participants were unable to confidently provide an answer.

Assumption 4: Maximum security is infeasible in practice and security features must be assigned priorities (ACCURATE)
Since complete security is not feasible, assigning priorities to security features is required [63]. The participants confirm that security is an important aspect, but since functionality is more critical (cf. SSF_1), only limited resources are available and they cannot realize every security feature (cf. CH_3). Other studies find that 20 % of the median effort for new projects is due to security [24]. Prioritization of security features is an essential part of the development process in practice (cf. SSF_5), often involving the consideration of different threats (cf. $EPSF_3$). Since the task of assigning priorities to security features must be at least partially performed by non-expert developers, often doing many of these things for the first time, they often underestimate the effort involved in realizing them (cf. CH_2). Since the selection of security features is based on experience (cf. SSF_3), developers often struggle in selecting security features beyond the common baseline (cf. SSF_2 and $EPSF_4$). This makes effort estimation difficult, which makes assigning priorities difficult. Ultimately, this might result in important security features not being implemented because of other priorities.

B. Findings and Observations

We provide insights into the engineering of security features:
Finding 1. Security is a cross-cutting concern that affects many domains, but requires specific security knowledge. In line with the intuition of researchers, this knowledge is only available to and recognized by a few developers. However, counterintuitively, developers still perceive themselves to be able to implement security features correctly because security libraries seem to be easy to use. Still, when comparing with previous work [21], it is uncertain whether developers truly implement them correctly, since a software system may operate functionally correct even when security features are missing.
Finding 2. Systematically engineering security features requires specific processes, e.g., threat modeling, that are orthogonal to software development. While companies integrate such activities into their development processes, they mainly integrate them without changing the process. Counterintuitively, this leads to activities such as threat modeling being used in practice in a lightweight manner tailored towards existing processes.
Finding 3. While intuition suggests that implementing security features involves huge effort, security feature implementation is perceived to require little code to integrate libraries. They are perceived to be well encapsulated, rarely tangled, and easy to maintain. Counterintuitively, estimating side effects is perceived especially difficult, since it is unclear what parts are affected by a change.Overall, proper planning of needed security

features is particularly challenging. Therefore, security-by-design methodologies should be automated to enable developers with limited skills to perform security assessment quickly [10].

C. Implications

Our study highlights implications on the engineering of security features for both researchers and practitioners.

Security knowledge. We notice a gap in foundational security knowledge among developers and stakeholders. Therefore, they require better tools and resources that help them integrate security features early in the development process. *Researchers* should focus on creating documentation including both correct usage examples and common misuse cases. *Practitioners* need to build deeper security knowledge to be able to notice insecure practices by e.g., adopting the concept of lighthouses.

Prioritization. The difficulty of balancing security and functionality with limited project resources is a significant challenge. *Researchers* need to explore methods for effectively prioritizing security features, both in terms of functionality and against each other, without compromising essential functionality. This includes developing strategies that assist in making informed decisions about security investments. *Practitioners* implicitly prioritize security features continuously, but should explicitly document decisions to facilitate reasoning and security hardening.

Security metrics. The lack of security metrics hampers the ability to assess clear and actionable insights about the security level of a system. Effective security metrics could not only assist developers in assigning priorities for security features, but also encourage stakeholders in understanding risks of neglecting security features. We identify a pressing need for security metrics which *researchers* should further explore. *Practitioners* could adopt formal security-by-design techniques which allow a systematic reasoning about a system's security [7], [68], [69].

Modeling techniques. Threat modeling is a crucial practice among developers, yet often performed in an informal and light-weight way. The *research community* should therefore investigate more efficient and accessible modeling techniques, that aid developers in integrating them into their workflows.

X. Threats to Validity

Internal Validity. Author bias could impact our results. Since we used semi-structured interviews, some questions might have been missed. We employed two authors to conduct each interview, both ensuring all questions were covered. To reduce participant bias, we mainly used open-ended questions, resorting to closed-ended ones only for clarification. We gave specific examples only if participants struggled to answer, thus minimizing bias and fostering understanding. The first author coded the interviews, but two authors thoroughly discussed the code book to validate and counteract any potential biases.

External Validity. The generalizability of our results might be threatened due to the composition of participants [70]. Company standards influence participants' experiences. We chose companies from diverse domains with varying security importance, finding security relevance in nearly all. Participants' backgrounds may affect generalizability, since willingness to participate depends on self-perceived experience [70]. Therefore, we recruited individuals with varied experiences and roles to ensure diversity. Response bias could threaten generalizability, when participants are hesitant to admit insecure practices [70]. We thus contrasted and validated interviewee perceptions with other empirical studies, especially where conflicts with empirical findings arose. Furthermore, the inability of participants to answer all questions could limit the accuracy of answers due to response bias [70]. We still received detailed responses, covering all research questions and reaching saturation. Lastly, our study provides insights on the engineering of security features from developer perspective. Since we did not have access to the participants' projects, we were unable to triangulate them with additional artifacts. Future research should investigate discrepancies between this perception and actual implementation of security features by analyzing software systems.

XI. Conclusion

While the research community provides security technologies and usage guidelines with many underlying assumptions based on intuition, little is known about how developers face challenges in practice when developing security features. To address this gap, we contribute an exploratory interview study with 26 experts from industry. We elicited their experience to validate and contextualize four common assumptions.

Our study reveals that companies generally follow security-by-design principles, yet specific security requirements are sparse. Threat modeling is widely used to identify potential threats and devise corresponding security measures. Security features are routinely included in most software systems by using established frameworks and libraries, adhering to best practices and default settings. Developers perceive security features as easy to implement, effectively separating security features from other functionalities and minimizing the amount of code.

Nonetheless, developers struggle with engineering security features due to a lack of foundational knowledge about them, often underestimating the effort needed. To address these challenges, developers often undergo training in specific areas of security. However, time constraints force them to prioritize functional features over security enhancements, hampering the comprehensive implementation of necessary security measures.

As our study has shown, many assumptions in security research may not be empirically validated and partially or even entirely inaccurate. Researchers should aim to validate more assumptions empirically by, e.g., conducting surveys and interviews with developers, or case studies of software projects, to ensure that their theories match practice.

Acknowledgements

We thank all 26 interviewees for their participation. This work was partially funded by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) under Germany's Excellence Strategy - EXC 2092 CASA - 390781972.

REFERENCES

[1] G. McGraw, "Software Security," *IEEE Security & Privacy*, vol. 2, no. 2, pp. 80–83, 2004.

[2] J. Cawthra, M. Ekstrom, L. Lusty, J. Sexton, and J. Sweetnam, "Data Integrity: Detecting and Responding to Ransomware and Other Destructive Events," National Institute of Standards and Technology, Tech. Rep. NIST SP 1800-26, 2020.

[3] D. B. Parker, "Toward a New Framework for Information Security," in *Computer Security Handbook*, 1st ed., S. Bosworth, M. E. Kabay, and E. Whyne, Eds. Wiley, 2015, vol. 1, ch. 3, pp. 3.1–3.23.

[4] G. Sindre and A. L. Opdahl, "Eliciting security requirements with misuse cases," *Requirements Engineering*, vol. 10, no. 1, pp. 34–44, 2005.

[5] C. B. Haley, R. C. Laney, J. D. Moffett, and B. Nuseibeh, "Security Requirements Engineering: A Framework for Representation and Analysis," *Transactions on Software Engineering (TSE)*, vol. 34, no. 1, pp. 133–153, 2008.

[6] D. Mellado, C. Blanco, L. E. Sánchez, and E. Fernández-Medina, "A systematic review of security requirements engineering," *Computer Standards & Interfaces*, vol. 32, no. 4, pp. 153–165, 2010.

[7] J. Jürjens, *Secure Systems Development with UML*. Springer, 2005.

[8] A. Shostack, *Threat Modeling: Designing for Security*. John Wiley & Sons, 2014.

[9] M. Salnitri, F. Dalpiaz, and P. Giorgini, "Designing secure business processes with SecBPMN," *Software & Systems Modeling (SoSyM)*, vol. 16, no. 3, pp. 737–757, 2017.

[10] V. Casola, A. De Benedictis, M. Rak, and U. Villano, "A novel security-by-design methodology: Modeling and assessing security by SLAs with a quantitative approach," *Journal of Systems and Software (JSS)*, vol. 163, 2020.

[11] M. Green and M. Smith, "Developers are not the enemy!: The need for usable security APIs," *IEEE Security & Privacy*, vol. 14, no. 5, pp. 40–46, 2016.

[12] B. Potter and G. McGraw, "Software security testing," *IEEE Security & Privacy*, vol. 2, no. 5, pp. 81–85, 2004.

[13] S. Peldszus, K. Tuma, D. Strüber, J. Jürjens, and R. Scandariato, "Secure Data-Flow Compliance Checks between Models and Code based on Automated Mappings," in *International Conference on Model-driven Engineering Languages and Systems (MODELS)*, 2019, pp. 23–33.

[14] K. Tuma, S. Peldszus, D. Strüber, R. Scandariato, and J. Jürjens, "Checking Security Compliance between Models and Code," *International Journal on Software and Systems Modeling (SoSyM)*, 2022.

[15] D. Di Dario, V. Pontillo, S. Lambiase, F. Ferrucci, F. Palomba *et al.*, "Security testing in the wild: An interview study," in *Euromicro Conference on Software Engineering and Advanced Applications (SEAA)*, 2023, pp. 191–198.

[16] I. Ryan, U. Roedig, and K. Stol, "Unhelpful assumptions in software security research," in *Conference on Computer & Communications Security (CCS)*, 2023, pp. 3460–3474.

[17] G. Wurster and P. Oorschot, "The Developer is the Enemy," in *New Security Paradigms Workshop*, 01 2008, pp. 89–97.

[18] S. Krüger, S. Nadi, M. Reif, K. Ali, M. Mezini, E. Bodden, F. Göpfert, F. Günther, C. Weinert, D. Demmler, and R. Kamath, "CogniCrypt: Supporting developers in using cryptography," in *International Conference of Automated Software Engineering (ASE)*, 2017, pp. 931–936.

[19] A.-K. Wickert, L. Baumgärtner, F. Breitfelder, and M. Mezini, "Python crypto misuses in the wild," in *International Symposium on Empirical Software Engineering and Measurement (ESEM)*, 2021. [Online]. Available: https://doi.org/10.1145/3475716.3484195

[20] M. Egele, D. Brumley, Y. Fratantonio, and C. Kruegel, "An empirical study of cryptographic misuse in android applications," in *Conference on Computer & Communications Security (CCS)*, 2013, pp. 73–84.

[21] Y. Acar, M. Backes, S. Fahl, S. Garfinkel, D. Kim, M. L. Mazurek, and C. Stransky, "Comparing the usability of cryptographic APIs," in *Symposium on Security and Privacy (SP)*, 2017, pp. 154–171.

[22] T. D. Oyetoyan, D. S. Cruzes, and M. G. Jaatun, "An empirical study on the relationship between software security skills, usage and training needs in agile settings," in *International Conference on Availability, Reliability and Security (ARES)*, 2016, pp. 548–555.

[23] S. Nadi, S. Krüger, M. Mezini, and E. Bodden, "Jumping through hoops: why do Java developers struggle with cryptography APIs?" in *International Conference on Software Engineering (ICSE)*, 2016, pp. 935–946.

[24] E. Venson, R. Alfayez, M. M. F. Gomes, R. M. da C. Figueiredo, and B. W. Boehm, "The impact of software security practices on development effort: An initial survey," in *International Symposium on Empirical Software Engineering and Measurement (ESEM)*, 2019, pp. 1–12.

[25] "Replication package," https://doi.org/10.5281/zenodo.14237228, 2025.

[26] T. Berger, D. Lettner, J. Rubin, P. Grünbacher, A. Silva, M. Becker, M. Chechik, and K. Czarnecki, "What is a feature? a qualitative study of features in industrial software product lines," in *International Systems and Software Product Line Conference (SPLC)*, 2015.

[27] J. Bosch, *Design & Use of Software Architectures—Adopting and Evolving a Product Line Approach*. Addison-Wesley, 01 2000.

[28] D. Batory, J. N. Sarvela, and A. Rauschmayer, "Scaling Step-Wise Refinement," *Transactions on Software Engineering (TSE)*, vol. 30, no. 6, pp. 355–371, 2004.

[29] K. Chen, W. Zhang, H. Zhao, and H. Mei, "An approach to constructing feature models based on requirements clustering," in *International Conference on Requirements Engineering (RE)*, USA, 2005, p. 31–40.

[30] K. Kang, S. Cohen, J. Hess, W. Novak, and A. Peterson, "Feature-oriented domain analysis (FODA) feasibility study," Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA, Tech. Rep. CMU/SEI-90-TR-021, 1990.

[31] M. Riebisch, "Towards a more precise definition of feature models," *Modelling Variability for Object-Oriented Product Lines*, 2003.

[32] K. Hermann, S. Schneider, C. Tony, A. Yardim, S. Peldszus, T. Berger, R. Scandariato, M. A. Sasse, and A. Naiakshina, "A taxonomy of functional security features and how they can be located," 2025. [Online]. Available: https://arxiv.org/abs/2501.04454

[33] K. Tsipenyuk, B. Chess, and G. McGraw, "Seven pernicious kingdoms: a taxonomy of software security errors," *IEEE Security & Privacy*, vol. 3, no. 6, pp. 81–84, 2005.

[34] L. Bass, P. Clements, and R. Kazman, *Software Architecture In Practice*. Addison-Wesley Longman, 01 2003.

[35] J. H. Allen, S. Barnum, R. J. Ellison, G. McGraw, and N. R. Mead, *Software Security Engineering*. Pearson Education, 2008.

[36] M. Schumacher, E. Fernandez-Buglioni, D. Hybertson, F. Buschmann, and P. Sommerlad, *Security Patterns: Integrating security and systems engineering*. John Wiley & Sons, 2013.

[37] S. Peldszus, *Security Compliance in Model-driven Development of Software Systems in Presence of Long-Term Evolution and Variants*. Springer, 2022.

[38] J. Ryoo, B. Malone, P. A. Laplante, and P. Anand, "The use of security tactics in open source software projects," *IEEE Transactions on Reliability*, vol. 65, no. 3, pp. 1195–1204, 2016.

[39] L. Passos, J. Padilla, T. Berger, S. Apel, K. Czarnecki, and M. T. Valente, "Feature scattering in the large: A longitudinal study of linux kernel device drivers," in *14th International Conference on Modularity (MODULARITY)*, 2015.

[40] S. Apel, D. Batory, C. Kästner, and G. Saake, *Feature Interactions*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 213–241.

[41] R. Werlinger, K. Hawkey, and K. Beznosov, "An integrated view of human, organizational, and technological challenges of it security management," *Information Management & Computer Security*, vol. 17, no. 1, pp. 4–19, 2009.

[42] S. Klivan, S. Höltervennhoff, R. Panskus, K. Marky, and S. Fahl, "Everyone for themselves? a qualitative study about individual security setups of open source software contributors," in *2024 IEEE Symposium on Security and Privacy (SP)*, 2024, pp. 1065–1082.

[43] F. Shull, J. Singer, and D. I. Sjøberg, *Guide to Advanced Empirical Software Engineering*. Springer-Verlag, 2007.

[44] N. K. Denzin and Y. S. Lincoln, *The Sage handbook of qualitative research*. Sage, 2011.

[45] S. C. Weller, B. Vickers, H. R. Bernard, A. M. Blackburn, S. Borgatti, C. C. Gravlee, and J. C. Johnson, "Open-Ended Interview Questions and Saturation," *PLOS ONE*, vol. 13, no. 6, 2018.

[46] M. M. Hennink, B. N. Kaiser, and V. C. Marconi, "Code Saturation Versus Meaning Saturation: How Many Interviews Are Enough?" *Qualitative Health Research*, vol. 27, no. 4, pp. 591–608, 2017.

[47] "Whisper," https://github.com/openai/whisper.

[48] R. E. Boyatzis, *Transforming Qualitative Information*. sage, 1998.

[49] "MAXQDA Website," https://www.maxqda.com/, 2024.

[50] A. S. Ahmadian, S. Peldszus, Q. Ramadan, and J. Jürjens, "Model-based privacy and security analysis with CARiSMA," in *Joint Meeting on Foundations of Software Engineering (ESEC/FSE)*, E. Bodden, W. Schäfer, A. van Deursen, and A. Zisman, Eds. ACM, 2017, pp. 989–993.

[51] F. Reiche, T. Weber, S. Becker, S. Weber, R. Heinrich, and E. Burger, "Consistency Management for Security Annotations for Continuous Verification," in *International Conference on Model Driven Engineering Languages and Systems (MODELS)*, M. Wimmer, A. Egyed, B. Combemale, and M. Chechik, Eds. ACM, 2024, pp. 1096–1105.
[52] M. Krausz, S. Peldszus, F. Regazzoni, T. Berger, and T. Güneysu, "120 Domain-Specific Languages for Security," *CoRR*, vol. abs/2408.06219, 2024.
[53] K. Tuma, G. Calikli, and R. Scandariato, "Threat Analysis of Software Systems: A Systematic Literature Review," *Journal of Systems and Software (JSS)*, vol. 144, pp. 275–294, 2018.
[54] K. Tuma, R. Scandariato, and M. Balliu, "Flaws in flows: Unveiling design flaws via information flow analysis," in *International Conference on Software Architecture (ICSA)*, 2019, pp. 191–200.
[55] A. Apvrille and M. Pourzandi, "Secure software development by example," *IEEE Security and Privacy*, vol. 3, no. 4, pp. 10–17, 2005.
[56] B. Best, J. Jürjens, and B. Nuseibeh, "Model-based security engineering of distributed information systems using UMLsec," in *International Conference on Software Engineering (ICSE)*, 2007, pp. 581–590.
[57] J. Jürjens, J. Schreck, and P. Bartmann, "Model-based security analysis for mobile communications," in *International Conference on Software Engineering (ICSE)*, 2008, pp. 683–692.
[58] D. A. Basin, M. Clavel, and M. Egea, "A Decade of Model-Driven Security," in *Symposium on Access Control Models and Technologies (SACMAT)*, 2011, pp. 1–10.
[59] S. Seifermann, R. Heinrich, and R. H. Reussner, "Data-Driven Software Architecture for Analyzing Confidentiality," in *International Conference on Software Architecture (ICSA)*, 2019, pp. 1–10.
[60] S. Peldszus, J. Bürger, T. Kehrer, and J. Jürjens, "Ontology-driven evolution of software security," *Data & Knowledge Engineering (DKE)*, vol. 134, 2021.
[61] S. Peldszus, J. Bürger, and J. Jürjens, "UMLsecRT: Reactive security monitoring of java applications with round-trip engineering," *Transactions on Software Engineering (TSE)*, vol. 50, no. 1, pp. 16–47, 2024.
[62] T. Gorschek, E. Tempero, and L. Angelis, "On the use of software design models in software development practice: An empirical investigation," *Journal of Systems and Software (JSS)*, vol. 95, pp. 176–193, 2014.
[63] M. L. Mazurek, "We are the experts, and we are the problem: The security advice fiasco," in *Conference on Computer & Communications Security (CCS)*, 2022.
[64] I. S. 32, "Road vehicles – cybersecurity engineering," International Organization for Standardization (ISO), International Standard ISO/SAE 21434, 2021.
[65] H. Zhong and Z. Su, "Detecting API documentation errors," in *International Conference on Object Oriented Programming Systems Languages & Applications (OOPSLA)*, 2013, pp. 803–816.
[66] M. Piccioni, C. A. Furia, and B. Meyer, "An empirical study of API usability," in *International Symposium on Empirical Software Engineering and Measurement (ESEM)*, 2013, pp. 5–14.
[67] B. A. Myers and J. Stylos, "Improving API usability," *Communications of the ACM*, vol. 59, no. 6, p. 62–69, 2016.
[68] A. Shostack, "Experiences threat modeling at microsoft," in *MODSEC*, 2008.
[69] K. Tuma, R. Scandariato, and M. Balliu, "Flaws in flows: Unveiling design flaws via information flow analysis," in *2019 IEEE International Conference on Software Architecture (ICSA)*, 2019, pp. 191–200.
[70] D. M. Elston, "Participation bias, self-selection bias, and response bias," *Journal of the American Academy of Dermatology*, 2021.