# I Do and I Understand. Not Yet True for Security APIs. So Sad

Luigi Lo Iacono and Peter Leo Gorski
Cologne University of Applied Sciences (Germany)
{luigi.lo_iacono, peter.gorski}@th-koeln.de

***Abstract*—Usable security puts the users into the center of cyber security developments. Software developers are a very specific user group in this respect, since their points of contact with security are application programming interfaces (APIs). In contrast to APIs providing functionalities of other domains than security, security APIs are not approachable by habitual means. Learning by doing exploration exercises is not well supported. Reasons for this range from missing documentation, tutorials and examples to lacking tools and impenetrable APIs, that makes this complex matter accessible.**

**In this paper we study what abstraction level of security APIs is more suitable to meet common developers' needs and expectations. For this purpose, we firstly define the term security API. Following this definition, we introduce a classification of security APIs according to their abstraction level. We then adopted this classification in two studies. In one we gathered the current coverage of the distinct classes by the standard set of security functionality provided by popular software development kits. The other study has been an online questionnaire in which we asked 55 software developers about their experiences and opinion in respect of integrating security mechanisms into their coding projects. Our findings emphasize that the right abstraction level of a security API is one important aspect to consider in usable security API design that has not been addressed much so far.**

## I. Introduction

Usability is important, especially for security technologies, mechanisms and products, since usability problems in security often result in serious threats for the users [1], [2], [3]. Usable Security is a domain of active research and development that tries to understand the correlation between usability and security, which is not a contradiction or conflict of objectives as commonly assumed [4]. The users' capabilities, perceptions and expectations are put into the center of the considerations in the development process. This avoids making the user the weakest link of a security system.

Since the first notable publications in the usable security domain starting from 1996 [5], [6], [7], the main research focus has been on end users. Only recently the focus has been drawn to software developers, system integrators and system administrators as special user groups [8], [9]. Understanding the needs of software developers is especially important, though, since failures on their site triggers an avalanche, affecting all the end users utilizing the vulnerable digital product.

The interfaces, software developers most commonly interact with, are application programming interfaces (APIs). Developers can access certain functionalities hidden behind APIs, providing them with a strong tool from which they can build their own software by composing it out of API calls. Security APIs have been and still are a major hassle for developers, though. This has been shown most critically for TLS/SSL connection establishment in Android [10] and iOS [11], TLS/SSL certificate validation in non-browser software [12] and also for OAuth single-sign-on implementations [13]. This issue becomes increasingly relevant, since with the growing distribution and multi-disciplinarity of systems security is getting pervasive for software developers. When developing distributed services systems, e.g., software developers have to deal with the implementation of service security measures [14] for which a rich stack of security functionalities is available through APIs.

Unfortunately, the common approach carried out by developers for exploring and then adopting an API seems not to be compatible with current security APIs. For instance, the straight forward method of searching for ready to use code examples via community platforms like Stack Overflow does often not lead to secure software products [15].

Learning and understanding security APIs by doing is far from being real. There is a clear demand for a better understanding of developers' needs in terms of security APIs in order to be able to provide usable programming interfaces for security mechanisms that are equipped with basic usability properties such as learnability and fault tolerance.

Most of the research available so far is mainly targeting APIs for cryptographic schemes or protocols [8]. Security APIs do encompass much more than cryptographic APIs, though [13], [16]. A classification is required to structure this field and as a crucial prerequisite to guide targeted research and development activities. Such a classification can support the systematic analysis of security APIs especially in the light of their usability. It would allow identifying, e.g., the targeted user group. Software developers with a specialization in cryptography or security might be able to use any security API, but may also prefer a certain class of APIs. On the other hand, common developers might require a certain minimum level of abstraction in order to fit their language and mental models.


EuroUSEC '17, 29 April 2017, Paris, France

http://dx.doi.org/10.14722/eurousec.2017.23015

In this paper we introduce a classification of the various security APIs. We consider the level of abstraction as the main classification criteria. Our proposed classification is then part of two studies. In the first one we performed a quantitative analysis of the provided security APIs in the most popular programming environments to date. This study should provide qualitative insights on the security APIs available to developers out of the box. In the second study we gather an opinionated view of developers by means of a questionnaire-based online study. The goal of this study is to determine the preferences of software developers in terms of the adequate abstraction level of security APIs and its implications to consider.

The rest of the paper is structured as follows. In Section II we provide some required definitions in order to set the scenes for this paper. Section III introduces the proposed classification of security APIs. In order to gather first insights on the usefulness of the introduced classification, we conducted two studies (see Section IV). We discuss our findings in Section V and conclude the paper in Section VI.

## II. Foundations

A classification needs to be built upon a solid foundation. Thus, a careful definition of the term *Security API* is required to be able to identify and comprehensibly outline the inherent structure of the security API domain. As security APIs are a particular subgroup of APIs, common constraints for a classification are determined by a general API definition. For the classification given in Section III a matured explanation of this term is used, which had been proposed by Joshua Bloch [17]:

> "*An application programming interface (API) specifies a component in terms of its operations, their inputs and outputs. Its main purpose is to define a set of functionalities that are independent of their implementation, allowing the implementation to vary without compromising the users of the component. An API augments a programming language (or a set of languages with an interoperable calling convention). Alternatively, an API may be described in an interface definition language.*" [17].

The most important concept of an API, which is also the core reason for its ubiquitous utilization, is the abstract reuse of functionalities offered by already available implementations. Conceptually, applications and their functionalities can be integrated in different ways. An overview of common styles is given by Figure 1 which has been adapted and extended from [18]. The *file-based data exchange* style is straightforward but this software integration approach does not use interfaces satisfying the above definition of an API. A component specifying the functionalities by its operations, inputs and outputs is simply missing in this integration style. In fact, there is only a general data interchange mechanism via file input and output. The same is true for the *shared database* integration strategy. There is no specific functionality available besides generic data retrieval and storage operations. APIs according to the above definition are deployed in the integration style denoted as *monolithic APIs*. This term states the use of specific APIs that are limited to a determined execution environment running on a single host. This category most commonly includes stand-alone applications that are tightly composed by compile time API calls. These can be found as part of libraries, toolkits, frameworks or development kits and are synonymously referred to as API [19]. The *remote procedure call* and *message bus* strategies also make extensive use of APIs. In addition to monolithic APIs both enclose remote APIs that enable the integration of functionalities provided by external services running on distributed and potentially heterogeneous hosts. This is e.g. required for implementing Web Services. Modern internet-based applications which are commonly based on the architectural style REST [20] adopt the RPC integration strategy. These remote APIs are commonly specified using interface definition languages.

Following the above given definition for APIs, the subgroup of security APIs has to be determined by the offered set of functionalities. The different scopes of available definitions for security APIs are too limited and thus are not appropriate for a comprehensive classification. The definition by Michael Bond [21] is, e.g., strictly bound to cryptographic functionalities:

> "*A security API is an application programmer interface that uses cryptography to enforce a security policy on the interactions between two entities.*" [21].

Focardi et. al [22] set a security API in a specific relationship of trust between entities:

> "*A security API is an application program interface that allows untrusted code to access sensitive resources in a secure way.*" [22].

And Graham Steel [23] specifies a security API by its quality of resistance against malicious calls:

> "*An application program interface (API) is called a security API if it is designed so that no matter what sequence of function calls are made by the possibly malicious application code, certain security properties continue to hold.*" [23].

These three formulations describe different sets of functionalities and are thus defining distinct security API subgroups. A definition of security APIs should follow a more general definition, though, trying to consider the current field of application comprehensively. Such a consolidated and expanded definition has been provided by Gorski et al. [24]:

> "*A security API is an application programming interface that provides developers with security functionalities that enforces one or more security policies on the interactions between at least two entities.*" [24].

This definition is used as foundation of this paper and as a point of reference for the proposed classification.

## III. Classification of Security APIs

Reflecting the available definitions for the term security API in the previous section made obvious, that early definitions have a strong focus on cryptography while more contemporary once broadened the scope to include compounded security features. This emphasizes already that there are distinct classes

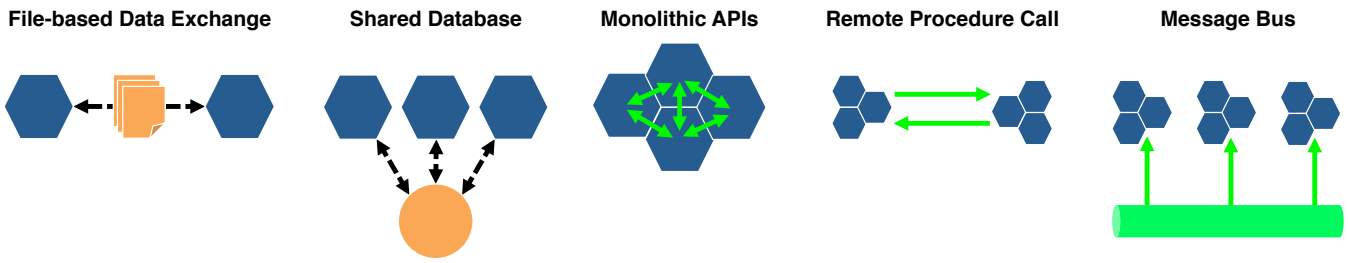


Fig. 1. Overview of application integration styles (adapted and extended from [18]). The three rightmost styles encompass the usage of APIs.

of APIs for implementing security mechanisms. A more profound understanding of this structure is required, since it might provide important new insights. One assumption here is, e.g., that a classification of security APIs might drive a more fine-grained clustering of the related user group of software developers as well. In this respect, the distinct security API classes could be assigned to more specifically defined groups of developers with their particular properties, abilities, deficiencies and mental models. Consequently, this would lead to more tailored and henceforth usable artifacts for the distinct groups including documentation, tooling and APIs. To obtain this better understanding, the security API field has to be structured in a viable manner.

### A. Methodology

In order to develop an appropriate classification for the purpose of this paper, we analyzed standards related to security mechanisms as well as various catalogs of security patterns [25], [26], [27], [28] and security controls [29]. Based on this input, we distilled a set of security mechanisms by compiling a list of unique components mentioned at least once in one of the analyzed artifacts.

### B. Classification

By the means of the aforementioned methodology we developed a classification of security APIs that is shown in Figure 2. It is important to understand, that the indicated classification does not provide a comprehensive picture of the whole domain. It shows the main structure that sorts the field in respect to the abstraction level that each individual component addresses.

The proposed classification splits up the domain of security APIs into two main subgroups (see Figure 2). The APIs with a rather low level of abstraction are denoted as *security primitives APIs*. The APIs with a higher level of abstraction are denoted as *security controls APIs*.

The APIs providing primitive security functions include most prominently cryptographic APIs, but also basic steganographic and watermarking schemes for information hiding. This end of the spectrum is made up by foundational means that can be used to realize basic security services such as confidentiality, integrity, authenticity and non-repudiation. These APIs provide a high flexibility since they allow selecting, initialize and combine the associated primitives to specific security controls as needed. This flexibility, however, demands a high degree of knowledge and expertise from developers. Otherwise they will fail developing robust and effective security protection means.

The other end of the spectrum consists of more concrete security controls including means for secure communication or storage. The APIs containing to this category are less flexible than the once belonging to the security primitives. On the other hand they are more goal-oriented and ready to use. When implemented correctly, they encapsulate a lot of security expertise and knowhow to lift this burden from the shoulders of the developers using a security controls API.

In these two first order classes of APIs several relationships can be observed. Security controls are most commonly constructed by a proper composition of security primitives. A security controls API offering the functionality of irretrievable deleting a file is build, e.g., upon a pseudo random generation API so that the file can be overwritten multiple times with junk data. This relation is unilateral and does not exist the other way round. Moreover, it does not apply to all security controls. There are some security controls that do not depend on security primitives at all for performing their internal tasks. The filtering of input is an example for those controls. No cryptography is needed for whitelisting an input to a network packet filter or web application. Another existing relationship can be observed between security controls themselves. In such a reflexive relation one control builds its functionality on top of another one. Consider a single-sign-on API as an example. It requires an authentication and possibly an identity management API in order to implement its own functionality.

The presented structure with its inherent relationships advocates for segmenting the group of software developers into distinct sets and focus them separately. Knowledgeable developers with the required experience in deploying security primitives should be the primary user group accessing the according APIs. Common developers should find their required security controls in an appropriate packaging respecting their mental model and providing an effective protection with secure defaults. In cases expert users have to fulfill routine tasks for which suitable security controls APIs are available, they most probably will make use of these shortcuts. The more critical path seems to be the lack of a required security control or its low flexibility forcing a common developer to build it based on security primitives APIs. If there exists a sufficiently diverse coverage of security APIs to satisfy the security needs of common software development projects and whether developers really desire this assumed classification to

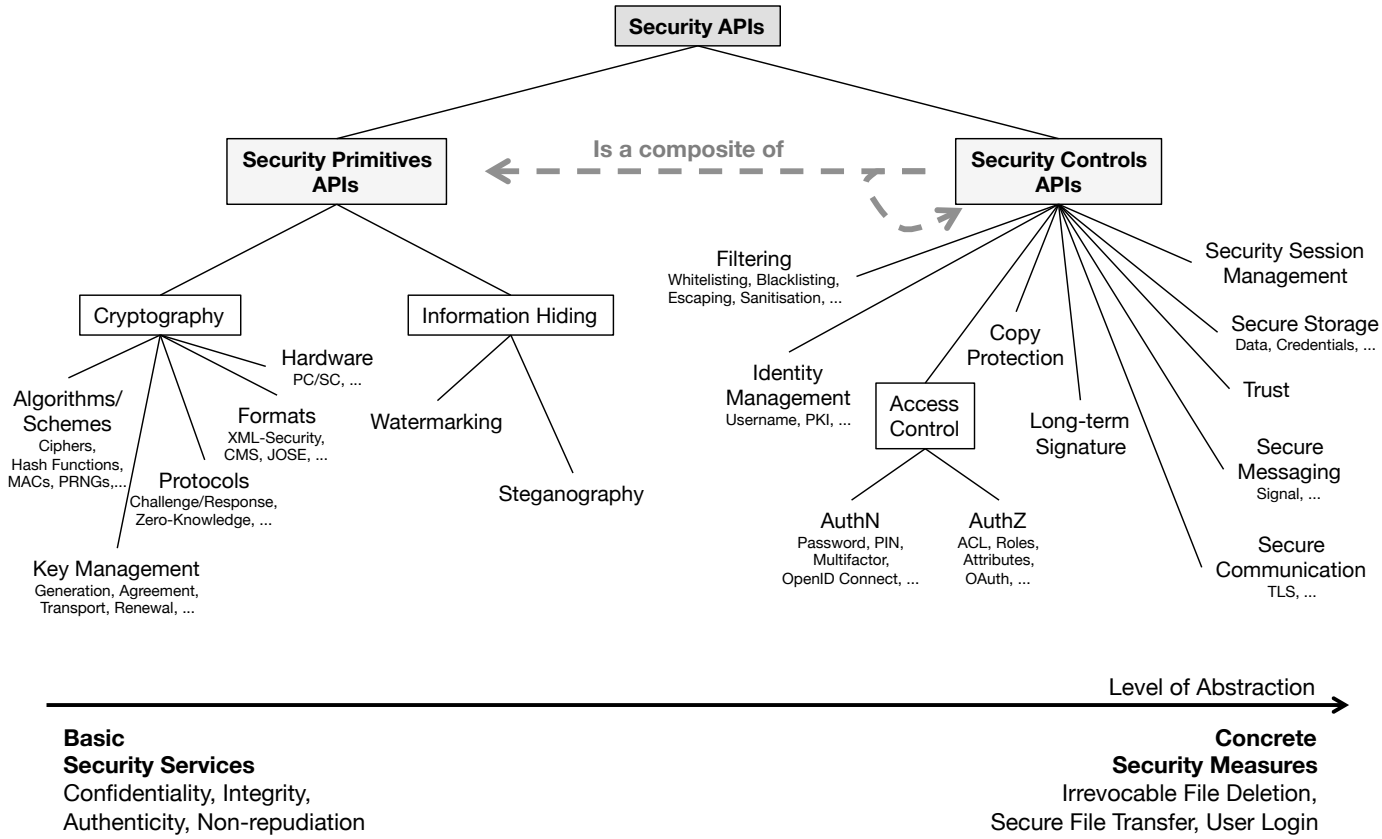


Fig. 2. This classification of security APIs shows the main structure of the domain. *Security Primitives APIs* provide common mechanisms for implementing basic security services such as confidentiality and integrity. *Security Controls APIs* instead provide more concrete security measures including user login and secure file transfer.

be fully available have initially been assessed by the following two studies.

## IV. Performed Studies

To evaluate whether the introduced classification of security APIs provides a beneficial tool for further research work we designed and performed two initial studies. One had the goal to cross-check the classification and to obtain an overview on what types of security APIs are available to developers in popular programming environments (see Section IV-A). With the other study we wanted to assess the preferences of software developers in respect to the granularity of security APIs (see Section IV-B).

### A. Popular Programming Environments Analysis

The landscape of programming languages and software development kits available to developers is huge. To focus our analysis to the most relevant environments to date, we selected the top ten programming languages listed in the *IEEE Top Programming Languages* (IEEE TPL) ranking [30].

Besides a pure ranking of programming languages, the IEEE TPL also indicates whether they are used to implement mobile, web, embedded or enterprise applications. Amongst the top ten, C, C++, Java, C#, and JavaScript are most prominently used for developing mobile applications. Developers of web applications do have a focus on languages including Java, Python, C#, PHP, JavaScript, Ruby, and Go. Since a large part of enterprise applications are nowadays developed using modern web technologies, the set of languages used for web programming has a large overlap with the ones used in enterprise application development, namely C, Java, Python, C++, R, C#, Ruby, and Go. Embedded developments are dominated by C and C++.

Since from a conceptual and technology viewpoint enterprise and web software development do have a lot in common nowadays, both types are merged for the market research study. An equivalent pairing can be observed for the embedded and mobile classes. Thus, the programming languages listed in both classes are considered only once. In the embedded class C/C++ and in the mobile class Java, C# and JavaScript are considered.

In the analysis we first identified what security mechanisms are "build-in" and can potentially be used by developers without the need to search for appropriate components. We then focus on externally available functions provided by third parties.

### B. Questionnaire-based Online Study

The questionnaire-based online study was conducted in March 2017 with the main goals to (1) validate the usefulness of the introduced classification (see Section 2, (2) gather in-

sights about what level of abstraction software developers wish to have when using security APIs and (3) to identify possible discrepancies between developers' needs and circumstances in practice. The full questionnaire is contained in the appendix of this paper. The related artifacts will be published on our website after the workshop [31].

*1) Survey Design:* The survey consists of 19 questions plus three additional ones depending on the answer. We focused on qualitative results. Thus, the survey contains eight questions in which the study participators were asked to answer with free text. For the question design, we orientated at related work. Robillard conducted a survey to identify learnability issues of APIs in general [32] and Nadi et al. conducted two surveys with a focus on usability issues of Java cryptography APIs [33].

Demographical data was collected in the first five questions including (Q1) the country of residence, (Q2) occupation, (Q3) development experience, (Q4) what types of software are developed and (Q5) what programming languages are used. The familiarity with the security domain was evaluated on one hand by (Q6) asking for an educational background in security and, if this is the case, for (Q6a) specific aspects of this education. We also wanted to know if there is a transitive connection between education, struggling with security APIs and the demanded level of abstraction. On the other hand we accessed (Q7) the frequency of security mechanisms usage.

In the questionnaire's core we first asked (Q8) who is from a developers point of view responsible for integrating security mechanisms in software systems. The intention behind this question was to understand if there is a general awareness of the responsibility lying with developers and designers of APIs, tools and documentation. To verify and give further substance to the classification proposed in Section III we asked (Q9) for currently applied security mechanisms and (Q10) a usage ranking of prominent features offered by security primitives APIs as well as security controls APIs. In order to identify specific aspects of security APIs and their usability we wanted to know (Q11) whether there are typical work steps to implement security mechanisms and (Q12) if there is any difference between security-related programming tasks and non-security related tasks. Additionally to implicitly find hints to the needed level of abstraction of security APIs we asked (Q13) whether a developer had problems implementing security mechanisms and, if this is the case, (Q13a and Q13b) for specific information and reasons. With two questions we asked directly (Q14) which level of abstraction a security API should offer to meet development needs and (Q16) what kind of security API is more appropriate. In addition we were interested in (Q15) whether the participants would recommend any API, tool or information resource to a colleague or friend who struggles with implementing a security mechanism.

The survey concluded with (Q17) the possibility to give further comments, thoughts or suggestions and to provide an (Q18) email address for (Q19) receiving results and/or invitations to further studies. The answers to these last three questions have been stored into a separate database with no link to the given answers and they have been not been considered in the analysis.

*2) Survey Respondents:* Executives or heads of department of various software development companies personally known to the authors were asked to forward a survey invitation to in-house software developers. Additionally programmers personally known to the authors have been invited to fill the questionnaire. We have received 59 full responses. The invalid answers of four participants could not be evaluated so we had to sort them out. Thus, the following analysis is based on $N = 55$ answers in total.

46 of the participants (83%) came from Germany. The other nine answers reached us sparsely from eight different countries including Austria, France, Mexico, Netherlands, Norway, Republic of Korea, Switzerland and USA.

By our recruiting we were able to gain answers from 38 participants (69%) who describe themselves as industrial and eight (15%) as freelance developers. We got also one samples from an industrial researcher, one from an undergraduate student, one from a graduate student and five from other occupation groups. One participant preferred not to answer. Furthermore, the sample is mostly represented by experienced developers. Almost half of the participants (45%) have more than ten years of experience in software development followed by 25% between five and ten years, 18% between two and five years and the remaining 11% less than two years (see Figure 3).

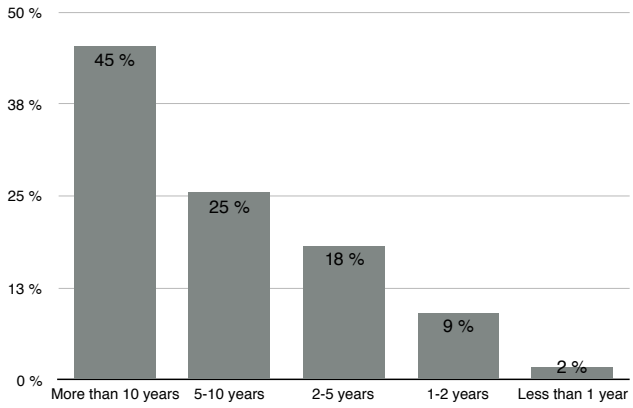


Fig. 3. Programming experience of the study participants

Developers of all five main software domains are represented in the sample. However the participants' focus lies with 85% on web applications as well as enterprise applications with 58% (see Figure 4). As previously discussed in Section IV-A the set of programming languages a developer is mainly using depends on the types of software she is developing. Consistently to the distribution of software types a wide range of languages is named by the participants with a strong emphasize on Java and JavaScript (see Figure 5).

Summarizing, the survey participators have much experience in programming various types of software with a large set of programming languages. This gives evidence that a relevant target group was reached. Although, the sample is mostly limited to software developers from Germany the gained data is appropriate to achieve the above defined objectives of this survey. The results of both conducted studies are discussed in the following section.

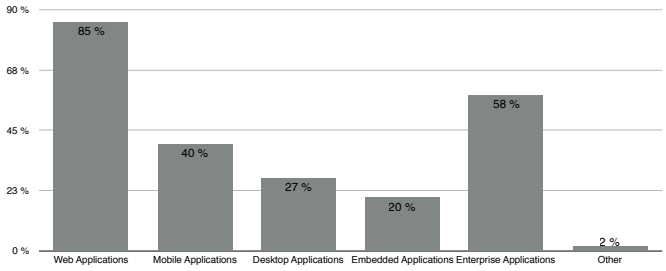


Fig. 4. Types of software which the participants develop

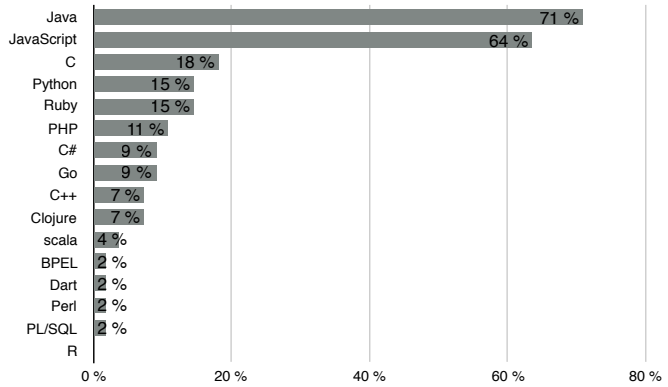


Fig. 5. Primarily used programming languages by the participants

## V. Results and Discussion

The structure of this section follows the defined objectives in Section IV-B. Beginning with the assessment of the introduced classification from section III, the results give evidence that it can be applied to structure the field of security APIs. The given free text answers to the question (Q9) which security mechanisms are generally implemented can almost completely and unambiguously assigned to either security primitives APIs or security controls APIs by using the proposed classification (see Figure 2) for a content-related coding. Therefore mechanisms like e.g. the protection against SQL injection or cross site scripting vulnerabilities were assigned to “Escaping” or for instance cross-origin resource sharing to “Whitelisting”. Named end user software products have been ignored in this process. Also the answers are mainly reflecting the determined rating options in Q10 and thus validate their validity. Derived from these ratings, developers have to use security mechanisms most often for (1) filtering functionalities like input validation (#1: 42%, #2: 22%), (2) authorization and authentication for access control (#1: 25%, #2: 33%) and (3) mechanisms for secure connections and communication (#1: 22%, #2: 16%). These three groups of security control functionalities take in total 89% of the first rank (#1) and still 71% of the second rank (#2). This result is further strengthened by the comparison of total ratings (see Figure 6), which takes selections for all fifteen ranks into consideration. Most of the participants also generally use cryptography like encryption and decryption (67%) and have to securely store credentials like user names and passwords (73%).

However, over 80% of the respondents are using these rated security mechanisms just occasionally. 27 participants

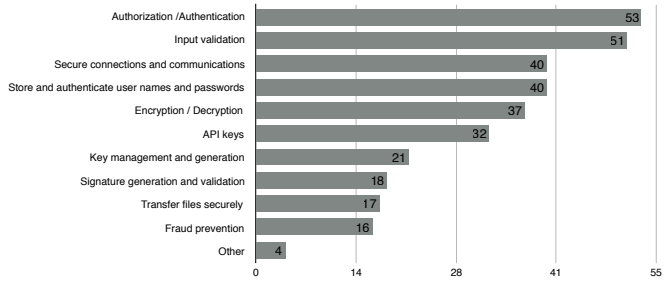


Fig. 6. Ranking of security-related functions most commonly used by study participants in their software projects

(49%) use security mechanisms rarely in less than 33% of their development tasks. 19 persons (35%) use security mechanisms occasionally in more than 33% but less than 66% of their tasks. Only nine participators (16%) have to deal frequently with security in more than 66% of their tasks. But, with no exception all participants have to integrate security mechanisms in their code. This indicates that most participants have other primary programming tasks but also have to implement security functionalities from time to time.

The results from Q9 and Q10 have a clear tendency to security control functionalities. Also if a developer just rarely or occasionally uses this kind of APIs she will likely make more errors with low level designs for which a detailed background knowledge is required. This target group has to be considered by designing the abstraction level of security APIs. A tendency concerning the needed level of abstraction was examined more directly by Q14 and Q16.

The result of Q14 is shown in Figure 7. The applied differentiation between the levels of abstraction is based on three distinct software developer personas proposed by Clarke [34], [35]: Opportunistic (High), Pragmatic (Medium), Systematic (Low).

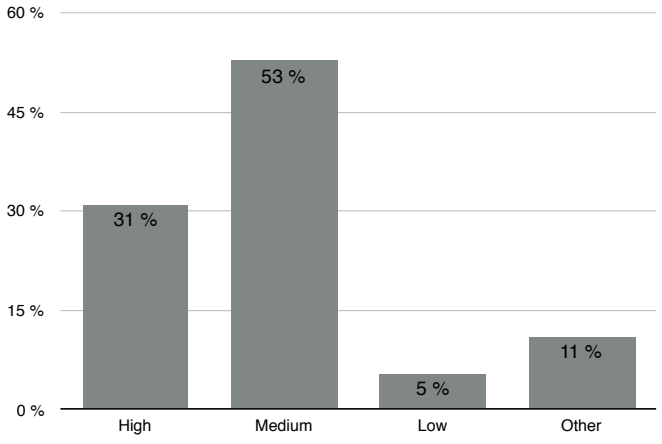


Fig. 7. Demanded security API abstraction level

Only 5% of software developers who took part in our survey want to determine implementation details by working with a low level API. 31% agree with not being interested in implementation details and just want the security to work by using a high level API. 53% prefer a medium level API, which

TABLE I. AVAILABLE SECURITY APIS IN POPULAR PROGRAMMING ENVIRONMENTS. THE BUILD-IN AVAILABILITY IS DENOTED AS ●. THIRD PARTY SECURITY FUNCTIONALITIES ARE DENOTED AS ◐. IF NO ACCORDING SECURITY APIS ARE AVAILABLE, ○.

| | Security Primitives APIs | | | Security Controls APIs | | | | |
|---|---|---|---|---|---|---|---|---|
| | *Cryptography* | *Steganography* | *Watermarking* | *Authentication* | *Authorization* | *Secure Communication* | *Filtering* | ... |
| Embedded | | | | | | | | |
| C/C++ | ● | ◐ | ○ | ◐ | ◐ | ◐ | ○ | |
| Mobile | | | | | | | | |
| Java/Android | ● | ◐ | ○ | ● | ● | ● | ● | |
| C#/Windows Phone | ○ | ◐ | ○ | ○ | ◐ | ○ | ○ | |
| JavaScript/Hybrid | ◐ | ◐ | ○ | ◐ | ◐ | ◐ | ● | |
| Objective-C/iOS | ● | ◐ | ○ | ◐ | ◐ | ● | ● | |
| Swift/iOS | ● | ◐ | ○ | ◐ | ◐ | ● | ● | |
| Enterprise | | | | | | | | |
| C/C++/STL | ◐ | ◐ | ○ | ● | ● | ◐ | ◐ | |
| Java/JEE | ● | ◐ | ○ | ● | ● | ● | ● | |
| Python | ● | ◐ | ○ | ● | ◐ | ● | ● | |
| R | ◐ | ◐ | ○ | ◐ | ◐ | ○ | ● | |
| C# | ◐ | ◐ | ○ | ○ | ◐ | ○ | ○ | |
| Ruby | ● | ○ | ○ | ◐ | ◐ | ◐ | ● | |
| Go | ● | ◐ | ○ | ◐ | ◐ | ◐ | ◐ | |
| Web | | | | | | | | |
| Java/Spring | ● | ◐ | ○ | ● | ● | ● | ● | |
| Python/Django | ● | ◐ | ○ | ● | ◐ | ● | ● | |
| C#/ASP.NET | ● | ◐ | ○ | ● | ◐ | ● | ● | |
| PHP/Symfony | ○ | ◐ | ○ | ● | ◐ | ● | ● | |
| JavaScript/Meteor | ◐ | ◐ | ○ | ● | ◐ | ● | ● | |
| Ruby/Ruby on Rails | ● | ○ | ○ | ● | ◐ | ● | ● | |
| Go/Revel | ● | ◐ | ○ | ● | ◐ | ● | ● | |

should offer low as well as high levels of implementation details and opportunities, depending on the actual programming task. 11% do not agree with the offered statements and gave other wordings.

In Q16 we directly asked which kind of security API the developers do find more appropriate for their needs. We gave an explanation for the used terms as follows: security APIs can be grouped into cryptographic APIs (e.g. encryption algorithms, hash functions and digital signatures) and security controls APIs (e.g. security protocols and mechanisms for authentication or authorization). 53% answered both, 36% security controls APIs, 7% cryptographic APIs and 4% none (see Figure 8). There is an obvious tendency towards security controls APIs. Also expert users demand for more abstract controls. A reason might be that they appreciate the shortcuts when developing frequent standard protections. Still, in the answers to question Q15 the given recommendation on particular security APIs do not show such a tendency. Only nine recommendations included security controls APIs, ten pointed to cryptographic APIs and 11 participants would recommend general information resources. The other participants would need more context to answer the stated question or were not able give any recommendation. This might be due to a lack of sufficient security controls APIs as we observed in our market analysis.

The outcomes from the analysis of available security

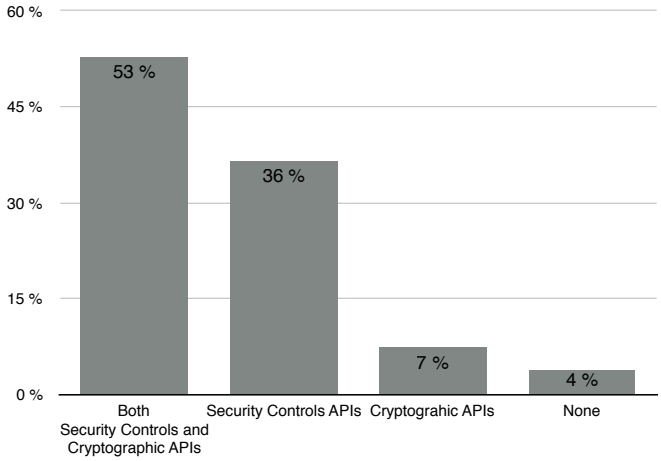


Fig. 8. The participants' needs for security APIs

APIs in popular programming environments used to date (see Section IV-A and Table I) emphasize that the security APIs directly available to developers as build-in functions are dominated by security primitives APIs. The identified security mechanisms that are "build-in" and which can potentially be used by developers without requiring to search for appropriate components are denoted by a filled circle ('●'). We then focused on externally available functions provided by third

parties (denoted as '◐'). In cases, in which neither a build-in nor an external component could have been found for the targeted security API, the corresponding table cell contains a non-filled circle symbol ('○'). As can be seen from Table I developers—whether security-experienced or not—most commonly have to deal with basic cryptographic functionalities. Only in application domains that do have a longer history in terms of being distributed systems by design include some security controls in addition. There is a clear demand to close this gap and to expand the set of security controls APIs. This discrepancy between the identified developers' needs of high and medium level security APIs and mostly offered low-level APIs in practice lead to various problems.

34 participants (61%) encountered problems while implementing security mechanisms. Seven respondents generally state they had problems just "doing it right". Eleven others explained in more detail that they have struggled to understand or use an API:

> "*Some security libraries are not user friendly. Hence, people tend to just get it running due to time restrictions. By this they may break the actual security mechanism, making the whole usage of the library senseless.*"

Others reported issues with their answers to Q13 in respect to the understanding of underlying security concepts (four times), the complexity of concepts (four times) and finding best practices as well as safe implementations (three times). The participants see different root causes for these problems. (1) Themselves, as human factor, have little experience (five times) or make insufficient efforts (three times). (2) Other developers, as nine respondents determine the design of an API, library or documentation as reason for their problems:

> "*Libraries that are not well documented (missing documentation or very detailed documentation that is for the praxis irrelevant, we need HowTos that for the most common scenarios allow a fast implementation).*"

(3) The organizational environment, as it is responsible for a tight timescale (named four times) and not counting security to business priorities (named three times). (4) The last issue mentioned by five developers as root cause is again the complexity of security concepts.

In this context we also tried to find a connection between the educational security background and having problems with the adoption of security APIs. The relation between participants who have some kind of educational security background (58%) and those who have not (42%) is almost balanced in the sample. Those 32 persons who answered with yes were additionally asked for concrete aspects. As expected the answers do not contain complete handbooks of modules but indicate a broadly based educational landscape. Most named topics range from cryptography and encryption (20 times), network security and security protocols (8 times), vulnerabilities and attacks (6 times), security services (6), public key infrastructures (4 times), practical countermeasures (4 times) and best practices and guidelines (3 times).

Although most software developers have to integrate security mechanisms in their software, a large number do not have an educational background in security. This should be especially considered by the design of high or medium level APIs to lower the initial hurdle for programmers with minor experience in security. This should be especially the case for frequently used functionalities like input validation and sanitation, authorization and authentication as well as for secure connections and communications. Nevertheless we were not able to proof a statistically significant correlation between previous knowledge from education and having problems when implementing security mechanisms (Chi-squared test p-value=0.493).

As with (1) the developer himself, (2) other developers and (3) the organizational environment different roots for problems with security APIs have been reported, the question arises whether who is responsible for integrating security mechanisms in software systems in the first place (Q8). The participants' opinions are widely dispersed, ranging from just "the developer" (11 times) or just "the software architect" (5 times) to "everybody" (8 times) or "the entire team" (3 times). Others name specialists in the domain of security (3 times) or in respect of project knowledge ("*The developer or software architect who knows the software in depth.*"). 23 answers describe multiple groups of persons. Summarizing, this diverse groups include—in addition to software developer, software architect and specialist—software designer, tester, operator, customer, business analyst, quality analyst, requirements engineer, group leader, software integrator as well as language designer (named one time) and framework provider (named two times).

As the development of complex software systems may follow various software engineering processes involving many stakeholders with different competences it can not be generally stated who takes finally the responsibility for bringing security measures into software. The results show that software developers mostly see themselves, at least partially, responsible for the integration. But they do also rely on other entities such as programming interfaces, e.g., which offer security functionalities, as a telling example in the web framework AngularJS shows:

> "*Each version of AngularJS 1 up to, but not including 1.6, contained an expression sandbox, which reduced the surface area of the vulnerability but never removed it.* ***In AngularJS 1.6 we removed this sandbox as developers kept relying upon it as a security feature even though it was always possible to access arbitrary JavaScript code if one could control the AngularJS templates or expressions of applications.***" [36]

This is just one prominent example which is supporting the findings of our survey. The subsequent question (Q11) asked in which work steps developers should be supported to solve a security related task. The obtained answers to Q11 give valuable insights to this. Some of the answers contained very technically detailed descriptions, which are not suitable for the evaluation and which have thus been excluded. The rest of the answers could be clustered in several groups. 13 different answers describe the need to specify requirements in the first step. As software developers have an information need at the beginning of the programming task, six participants would start

with a general research to understand a security mechanism and get information about their current development status. Additionally seven respondents would work out a risk or threat model. In all steps 20 participants (36%) mentioned the searching for best practices and available tools like libraries, frameworks or security APIs. After creating a concept (named 7 times) and consulting other persons (named 10 times) the measure would be implemented (named 14 times). In all steps 32 participants (58%) would write and/or perform tests like functional unit testing or fuzzing. This is also relevant for already existing systems which have to be analyzed (named 4 times) or tested for e.g. bug fixing or patching after release. As a second line of validation 10 developers would request a code audit or review after the implementation.

Different proposals consider the usability of security APIs as special in contrast to general API usability, following specific additional principles [24], [8]. Because there is no empirical evidence for this yet, Q12 was meant to give more substance to previous research results. However, the respondents have controversial opinions concerning the question if there is a difference between security-related programming tasks and non-security related tasks. The difference is argued mainly by the security context. Summarizing, security is described as a non-functional requirement which brings additional complexity, requires different thinking, special domain knowledge, more effort, attention as well as care and which results in more serious consequences. These might be reasons for a frequent negative perception of security-related tasks:

> "*Security-related tasks are not welcomed by sponsors, tend to have poor management attention and often feel like a burden. They tend to require more attention for review and quality assurance because it is often more complex to prove that they fulfill their purpose.*"

A different perspective, which is shared by 14 participants is relativizing the aforementioned opinion. From their point of view all tasks are security related or require the same conceptual approach. One respondent found the following wording:

> "*I think most tasks that are involved in creating a software system do involve security to a certain extend. Having a strict separation between security and non-security tasks is a reason that a lot of systems do have security holes after their release.*"

Whether there is a difference between security-related programming tasks and non-security related tasks or not, both described paradigms demand for a due diligence when dealing with security related implementation tasks, because of the special implications.

## VI. Conclusion and Outlook

One approach to design security APIs that do not incorporate obvious usability problems is to adhere to general usability heuristics [37]. The *match between system and the real world* principles says that a system should speak the users' language, with words, phrases and concepts familiar to the user [38]. When applying this to the security API classification introduced in this paper it follows that this can be offered more likely by security controls APIs. Another usability principle says that interfaces should not contain information, which is irrelevant or rarely needed. Again, this principle can be adhere to more easily by security controls APIs. Those APIs provide the right level of abstraction that allows diminishing insignificant parts behind suitable function calls. This would not only provide inexperienced users with an effective security control offering secure defaults, but also the expert user with an accelerator allowing to tailor frequent actions as recommended by the *flexibility and efficiency of use* heuristic.

All kinds of security APIs deserve a comprehensive research and development in order to improve their usability for software developers as has been initiated in [24], [8], [15]. Still, according to the results obtained by our studies, there must be a diverse look at the usable security API field. As our introduced classification advocates for, there should be at least a distinction of security APIs in security primitives and security controls. This diverse view supports also distinguishing various distinct groups of software developers. For the ones being inexperienced and unknowing in respect to security, the focus should be to develop appropriate and sufficient security controls APIs. This class offers first of all the demanded level of abstraction that is also closest to the users' language and mental models. Most security controls are made up by a composition of various security primitives and may hide the involved complexity behind a more graspable API. By this, security controls APIs do not interfere as much with the primary goal of developers—i.e., finalizing the software—as security primitives APIs do, since the latter requires an indepth understanding of the involved concepts, algorithms and relationships amongst the security primitives. Such a expertise and knowledge that requires a whole lot of experience will remain the competence of a specialized and henceforth rather small—-in comparison with the general case—group of developers.

With a clear lack of adequate security controls, common developers have to continue coping with security primitives APIs for the time being. Future work needs to continue with general research on security API usability with all its included aspects in order not to make developers the weakest link, ultimately putting their users into risk. A specific focus on security controls APIs shows to open the path for promising approaches towards usable security APIs. The obtained results from this initial contribution demand for follow-up studies. The survey needs to be broadened in respect to internationalization and the analysis needs to be extended as well as more intensively systematized.

## References


[1] I. Kirlappos and M. A. Sasse, *What Usable Security Really Means: Trusting and Engaging Users*. Cham: Springer International Publishing, 2014, pp. 69–78. [Online]. Available: http://doi.org/10.1007/978-3-319-07620-1_7

[2] A. J. DeWitt and J. Kuljis, "Aligning Usability and Security: A Usability Study of Polaris," in *Proceedings of the Second Symposium on Usable Privacy and Security*, ser. SOUPS '06. New York, NY, USA: ACM, 2006, pp. 1–7. [Online]. Available: http://doi.acm.org/10.1145/1143120.1143122

[3] M. Sasse, S. Brostoff, and D. Weirich, "Transforming the "Weakest Link" - a Human Computer Interaction Approach to Usable and Effective Security," *BT Technology Journal*, vol. 19, no. 3, pp. 122–131, 2001. [Online]. Available: https://doi.org/10.1023/A:1011902718709

[4] M. A. Sasse and M. Smith, "The security-usability tradeoff myth," *IEEE Security Privacy*, vol. 14, no. 5, pp. 11–13, Sept 2016. [Online]. Available: https://doi.org/10.1109/MSP.2016.102

[5] M. E. Zurko and R. T. Simon, "User-centered security," in *NSPW '96: Proceedings of the 1996 workshop on New security paradigms*. ACM, Sep. 1996. [Online]. Available: https://doi.org/10.1145/304851.304859

[6] A. Adams and M. A. Sasse, "Users Are Not the Enemy," *Commun. ACM*, vol. 42, no. 12, pp. 40–46, Dec. 1999. [Online]. Available: http://doi.acm.org/10.1145/322796.322806

[7] A. Whitten and J. D. Tygar, "Why Johnny Can't Encrypt: A Usability Evaluation of PGP 5.0," in *Proceedings of the 8th Conference on USENIX Security Symposium - Volume 8*, ser. SSYM'99. Berkeley, CA, USA: USENIX Association, 1999, pp. 14–14. [Online]. Available: http://dl.acm.org/citation.cfm?id=1251421.1251435

[8] 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, Sept 2016. [Online]. Available: https://doi.org/10.1109/MSP.2016.111

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

[10] S. Fahl, M. Harbach, T. Muders, L. Baumgärtner, B. Freisleben, and M. Smith, "Why eve and mallory love android: An analysis of android ssl (in)security," in *Proceedings of the 2012 ACM Conference on Computer and Communications Security*, ser. CCS '12. New York, NY, USA: ACM, 2012, pp. 50–61. [Online]. Available: http://doi.acm.org/10.1145/2382196.2382205

[11] S. Fahl, M. Harbach, H. Perl, M. Koetter, and M. Smith, "Rethinking ssl development in an appified world," in *Proceedings of the 2013 ACM SIGSAC Conference on Computer & Communications Security*, ser. CCS '13. ACM, 2013, pp. 49–60. [Online]. Available: http://doi.acm.org/10.1145/2508859.2516655

[12] M. Georgiev, S. Iyengar, S. Jana, R. Anubhai, D. Boneh, and V. Shmatikov, "The most dangerous code in the world: Validating ssl certificates in non-browser software," in *Proceedings of the 2012 ACM Conference on Computer and Communications Security*, ser. CCS '12. New York, NY, USA: ACM, 2012, pp. 38–49. [Online]. Available: http://doi.acm.org/10.1145/2382196.2382204

[13] S.-T. Sun and K. Beznosov, "The devil is in the (implementation) details: An empirical analysis of oauth sso systems," in *Proceedings of the 2012 ACM Conference on Computer and Communications Security*, ser. CCS '12. New York, NY, USA: ACM, 2012, pp. 378–390. [Online]. Available: http://doi.acm.org/10.1145/2382196.2382238

[14] P. L. Gorski, L. L. Iacono, H. V. Nguyen, and D. B. Torkian, "Service security revisited," in *2014 IEEE International Conference on Services Computing*, June 2014, pp. 464–471. [Online]. Available: https://doi.org/10.1109/SCC.2014.68

[15] Y. Acar, M. Backes, S. Fahl, D. Kim, M. L. Mazurek, and C. Stransky, "You get where you're looking for: The impact of information sources on code security," in *2016 IEEE Symposium on Security and Privacy (SP)*, May 2016, pp. 289–305. [Online]. Available: https://doi.org/10.1109/SP.2016.25

[16] S. Türpe, "Idea: Usable platforms for secure programming – mining unix for insight and guidelines," in *Engineering Secure Software and Systems: 8th International Symposium, ESSoS 2016, London, UK, April 6-8, 2016. Proceedings*, ser. LNCS, J. Caballero, E. Bodden, and E. Athanasopoulos, Eds., vol. 9639. Cham: Springer International Publishing, Apr. 2016, pp. 207–215. [Online]. Available: http://testlab.sit.fraunhofer.de/downloads/Publications/tuerpe2016idea.pdf

[17] J. Bloch, "A brief, opinionated history of the API," Talk, 2014, workshop on Evaluation and Usability of Programming Languages and Tools (PLATEAU). [Online]. Available: http://2014.splashcon.org/event/plateau2014-invited-speaker-josh-bloch

[18] C. Pautasso, O. Zimmermann, and F. Leymann, "Restful web services vs. "big"' web services: Making the right architectural decision," in *Proceedings of the 17th International Conference on World Wide Web*, ser. WWW '08. New York, NY, USA: ACM, 2008, pp. 805–814. [Online]. Available: http://doi.acm.org/10.1145/1367497.1367606

[19] J. Stylos and B. Myers, "Mapping the space of api design decisions," in *IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC 2007)*, Sept 2007, pp. 50–60. [Online]. Available: https://doi.org/10.1109/VLHCC.2007.44

[20] R. T. Fielding, "REST: architectural styles and the design of network-based software architectures," Doctoral dissertation, University of California, Irvine, 2000. [Online]. Available: http://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm

[21] M. K. Bond, "Understanding security APIs," Doctoral dissertation, University of Cambridge, UK, 2004. [Online]. Available: https://www.cl.cam.ac.uk/~mkb23/research/Thesis.pdf

[22] R. Focardi, F. L. Luccio, and G. Steel, *An Introduction to Security API Analysis*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, pp. 35–65. [Online]. Available: http://doi.org/10.1007/978-3-642-23082-0_2

[23] G. Steel, *Formal Analysis of Security APIs*. Boston, MA: Springer US, 2011, pp. 492–494. [Online]. Available: http://doi.org/10.1007/978-1-4419-5906-5_873

[24] P. L. Gorski and L. Lo Iacono, "Towards the usability evaluation of security apis," in *Tenth International Symposium on Human Aspects of Information Security & Assurance, HAISA 2016, Frankfurt, Germany, July 19-21, 2016, Proceedings.*, 2016, pp. 252–265. [Online]. Available: http://www.cscan.org/openaccess/?paperid=287

[25] O. S. Architecture. Security architecture patterns. [Online]. Available: http://www.opensecurityarchitecture.org/cms/library/patternlandscape

[26] E. Fernandez-Buglioni, *Security Patterns in Practice: Designing Secure Architectures Using Software Patterns*. John Wiley & Sons, Apr. 2013.

[27] B. Blakley and C. Heath, "Security Design Patterns," The Open Group, Tech. Rep., 2004. [Online]. Available: http://pubs.opengroup.org/onlinepubs/9299969899/toc.pdf

[28] J. Yoder and J. Barcalow, "Architectural patterns for enabling application security," in *4th Conference on Patterns Language of Programming (PLoP'97).*, 1997.

[29] O. S. Architecture. Control catalog. [Online]. Available: http://www.opensecurityarchitecture.org/cms/library/0802control-catalogue

[30] S. Cass. The 2016 top programming languages - c is no. 1, but big data is still the big winner. [Online]. Available: http://spectrum.ieee.org/computing/software/the-2016-top-programming-languages

[31] Data & Application Security Group. Research artifacts. [Online]. Available: https://das.th-koeln.de/artifacts

[32] M. P. Robillard, "What makes apis hard to learn? answers from developers," *IEEE Software*, vol. 26, no. 6, pp. 27–34, Nov 2009. [Online]. Available: https://doi.org/10.1109/MS.2009.193

[33] S. Nadi, S. Krüger, M. Mezini, and E. Bodden, "Jumping through hoops: Why do java developers struggle with cryptography apis?" in *Proceedings of the 38th International Conference on Software Engineering*, ser. ICSE '16. New York, NY, USA: ACM, 2016, pp. 935–946. [Online]. Available: http://doi.acm.org/10.1145/2884781.2884790

[34] J. Stylos and S. Clarke, "Usability implications of requiring parameters in objects' constructors," in *29th International Conference on Software Engineering (ICSE'07)*, May 2007, pp. 529–539. [Online]. Available: http://doi.acm.org/10.1109/ICSE.2007.92

[35] S. Clarke, "How usable are your APIs?" in *Making software: what really works, and why we believe it*, 1st ed., ser. Theory in practice, A. Oram and G. Wilson, Eds. O'Reilly, pp. 545 – 565.

[36] AngularJS. AngularJS: Developer guide: Security. [Online]. Available: https://docs.angularjs.org/guide/security

[37] J. Nielsen and R. Molich, "Heuristic Evaluation of User Interfaces," in *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, ser. CHI '90. New York, NY, USA: ACM, 1990, pp. 249–256. [Online]. Available: http://doi.acm.org/10.1145/97243.97281

[38] J. Nielsen, *Usability Engineering*. Morgan Kaufmann, 1994.

## APPENDIX: QUESTIONNAIRE

Welcome to our survey! We are researchers from the Cologne University of Applied Sciences (Germany) and our goal is to improve the usability of Security APIs. The purpose of this survey is to understand what developers do require to integrate security functionalities into their software. The

survey consists of 20 questions which will be answered in approximately 15 minutes. A note on privacy: This survey is anonymous. The record of your survey responses does not contain any identifying information about you, unless a specific survey question explicitly asked for it. If you have any question, remark or comment about this survey, please contact Peter Gorski (peter.gorski@th-koeln.de).

Q1: What country do you live in? (dropdown list)

Q2: What is your current occupation? (multiple choice): (1) Freelance developer (2) Industrial developer (3) Industrial researcher (4) Academic researcher (5) Graduate student (6) Undergraduate student (7) Prefer not to answer (8) Other (free text)

Q3: How many years of development experience do you have? (radio list): (1) More than 10 years (2) 5-10 years (3) 2-5 years (4) 1-2 years (5) Less than 1 year (6) Prefer not to answer

Q4: What type(s) of software do you develop? (multiple choice): (1) Web Applications (2) Mobile Applications (3) Desktop Applications (4) Embedded Applications (5) Enterprise Applications (6) Other (free text)

Q5: What programming language(s) do you use primarily? (multiple choice): (1) C (2) Java (3) Python (4) C++ (5) R (6) C# (7) PHP (8) JavaScript (9) Ruby (10) Go (11) Other (free text)

Q6: Have IT security topics been part of your educational background? (yes/no)

If the answer is yes:

a) What aspects of IT security have been part of your education? (free text)

Q7: How often do you need to integrate security mechanisms in your code? (radio list): (1) Frequently - I use security mechanisms in more than 66% of my development tasks (2) Occasionally - I use security mechanisms in more than 33% but less than 66% of my development tasks (3) Rarely - I use security mechanisms in less than 33% of my development tasks (4) Never - I never use security mechanisms in any of my development tasks

Q8: Who is, in your opinion, responsible for integrating security mechanisms in software systems? (free text)

Q9: Which security mechanisms did you implement in your code? (free text)

Q10: What are the most common security-related functions you use in your code? Don't rank functions you never used. Use "Other (1)" to "Other (5)" for missing functions as needed. (ranking): (1) API keys (2) Authorization / Authentication (3) Encryption / Decryption (4) Fraud prevention (5) Input validation (6) Key management and generation (7) Secure connections and communications (8) Signature generation and validation (9) Store and authenticate user names and passwords (10) Transfer files securely (11 - 15) Other (free text)

Q11: Assume your current task is to integrate a security mechanism in your code (e.g. secure communication, user login or input validation). Please list the first work steps you would do. (multiple free text): (1) 1. step (free text) (2) 2. step (free text) (3) 3. step (free text) (4) 4. step (free text) (5) 5. step (free text)

Q12: What is, in your opinion, the difference between security-related programming tasks and non-security related tasks? (free text)

Q13: Did you ever had problems implementing security mechanisms or integrating them into your code? (yes/no)

If the answer is yes:

a) What were those problems? (free text)

b) What do you consider as the root cause for these problems? (free text)

Q14: Which level of abstraction should a security API offer to meet your development needs? (radio list): (1) High - It should be a high level API. I am not interested in implementation details. I just want the security to work. (2) Medium - An API should offer me low as well as high levels of implementation details and opportunities, depending on my actual programming task. (3) Low - It should be a low level API. I want to determine implementation details myself. (4) Other (free text)

Q15: Is there any API, tool or information resource you would recommend a colleague or friend who struggles with implementing a security mechanism? (free text)

Q16: What do you think is more appropriate for your needs? Security APIs can be grouped into cryptographic APIs (e.g. encryption algorithms, hash functions and digital signatures) and security controls APIs (e.g. security protocols and mechanisms for authorization or authentication). (radio list): (1) Cryptographic APIs (2) Security controls APIs (3) Both (4) None

Q17: Do you have further comments, thoughts or suggestions? (free text)

Thank you for taking part in the survey! If you like to receive the study results and/or to participate in future studies then please provide your email address. This data will be stored seperately from the survey and will not be published.

Q18: Check any that apply (multiple choice): (1) I would like to receive the study results (2) You may consider me for future studies

Q19: Please enter your email address (free text)