Proceedings of the Tenth International Symposium on
Human Aspects of Information Security & Assurance (HAISA 2016)

Towards the Usability Evaluation of Security APIs

P.L. Gorski and L.L. Iacono

Cologne University of Applied Sciences, Germany
e-mail: {peter.gorski, luigi.lo_iacono}@th-koeln.de

Abstract

Application Programming Interfaces (APIs) are a vital link between software components as well as between software and developers. Security APIs deliver crucial functionalities for programmers who see themselves in the increasing need for integrating security services into their software products. The ignorant or incorrect use of Security APIs leads to critical security flaws, as has been revealed by recent security studies. One major reason for this is rooted in usability issues. API Usability research has been deriving recommendations for designing usable APIs in general. Facing the growing relevance of Security APIs, the question arises, whether the observed usability aspects in the general space are already sufficient enough for building usable Security APIs. The currently available findings in the API Usability domain are selective fragments only, though. This still emerging field has not produced a comprehensive model yet. As a consequence, a first contribution of this paper is such a model that provides a consolidated view on the current research coverage of API Usability. On this baseline, the paper continues by conducting an analysis of relevant security studies, which give insights on usability problems developers had, when using Security APIs. This analysis leads to a proposal of eleven specific usability characteristics relevant for Security APIs. These have to be followed up by usability studies in order to evaluate how Security APIs need to be designed in a usable way and which potential trade-offs have to be balanced.

Keywords

Security APIs, Usable Security, Software Security, API Usability, Evaluation

1. Introduction

One consequence that comes along with the digital transformation and advances in all spheres of business and life is that sensitive data is increasingly produced, stored, transmitted and processed in digital form by numerous kinds of electronic devices and their applications. Moreover, most current software in this context is part of one or more distributed systems and, thereby, needs to interact with various remote services. Such interconnected systems are the driving engine for many application fields including the industry, transportation, energy, consumer and healthcare domains. A strong demand for security is, henceforth, required in order to protect users against malicious actions.

Application Programming Interfaces (APIs) are ubiquitously used to develop the digital transformation in terms of the underlying software. The API concept enables the simple reuse of functionalities by abstraction. Security services are one such type of functionality. Due to the high complexity of security concepts, security software components are designed and implemented by developers specialized in security.

Non-specialized developers perform the adoption of Security APIs, in contrast. Thus, the security of contemporary software is heavily depending on the effective use of Security APIs by common programmers. In fact, the results of recent security studies give evidence that one main reason for security flaws in deployed software products lies in the unintended incorrect use of Security APIs, which in turn is caused by bad API design decisions making them hard to use properly. Defectively integrated security features in software products are not only originated from novice or hobby programmers, but also from professional software companies (Fahl *et al.* 2012). Thus, this is a far-reaching problem, which cannot only be explained by ignorance only.

These usability issues of Security APIs affect distinct areas in frontend and backend software, middleware or platforms and therefore cannot be improved by just fixing one central hub. So far, no research has been conducted to picture a specific concept for the usability evaluation of Security APIs. Proposed recommendations in the context of security studies address symptoms of either respective security mechanisms like OAuth 2.0 (Hardt, 2012) or execution platforms like Android (Google, 2016). Thus, one contribution of this paper is an initial proposal of common and general usability aspects that need to be considered when designing APIs for security mechanisms.

The rest of the paper is organized as follows. Section 2 defines the term Security API as required foundational prerequisite. Section 3 presents related work before introducing the underlying methodology used for this work. A coherent model for the current state of API Usability is introduced in Section 5. It lays the fundament for analyzing the degree of maturity and applicability concerning Security API Usability. Derived supplement evaluation topics of API Usability by so far unconsidered common and specific characteristics of Security APIs are introduced in Section 6. The contributions of this paper are summarized and discussed in Section 7 before concluding with an outlook on future work.

2. Security API

The term Security API has first been coined by scientific disciplines focusing on security protocol analysis. To satisfy the definition by Bond (2004), "*a Security API is an application programmer interface that uses cryptography to enforce a security policy on the interactions between two entities*". This would exclude APIs, which don't apply cryptography to offer security functionalities such as input validation libraries for reducing the risk of injection attacks including e.g. Cross-Site-Scripting and SQL Injection (OWASP 2013). Steel (2011) defines Security APIs to be a link between a trusted and an untrusted area. He also considers its behavior against arbitrary combinations of function calls. The first aspect doesn't match e.g. the trust relationship built by the TLS (Transport Layer Security) (Dierks and Rescorla, 2008) protocol.

As can be seen, these definitions do not cover all contemporary use cases of Security APIs in distributed software applications. For the purpose of this paper the term

Security API is henceforth defined according to the definition of Bond (2004) as follows: *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.*

3. Related Work

To the knowledge of the authors, there have not been any studies on neither the usability evaluation of Security APIs nor on the applicability of general API Usability aspects to Security APIs. Merely minor points of contact with Security APIs can be found in a few early studies, which examined API usability in general. Ellis *et al.* (2007) evaluated the usability of the Factory pattern in API design. In one of the assigned tasks, the participants have been instructed to instantiate an SSLSocket using the Java Standard Edition (SE) API version 1.5. Important security relevant downstream tasks such as certificate validation have been out of focus, though. Five of twelve participants failed to complete the task in the given time. Thus, Ellis *et al.* (2007) concluded, that the Factory pattern hinders usability of an API. This result provides evidence that general API Usability research also applies to Security API usability. Still, the Factory pattern is the design of choice to construct SSLSockets in the latest Java SE version 8. A web authentication task has been part of a user study conducted by Stylos and Myers (2008). They used a self-modified version of the Apache Axis2 API (Apache, 2016) in order to focus on specific user behavior with optional classes. However, the security context has not been particularly mentioned in the study results.

In a security study conducted by Fahl *et al.* (2013) first efforts have been undertaken in the direction of API usability evaluation. They interviewed fourteen developers who had integrated Secure Socket Layer (SSL) (Freier *et al.* 2011) defectively in their applications. Additionally they pre-tested the usability of an own framework approach for SSL development, but detailed usability measures have not been described. In the recent past Green and Smith (2015) advocated for more communication between Security API designers and software developers and the application of developer-centered design approaches. They also called attention to the need for qualitative and quantitative empirical studies in this research area.

4. Methodology

To create a solid base for research on the usability of Security APIs a model for general API Usability is elaborated by an extensive literature research. The result also allows a consolidated view on the current research coverage, which also glances at Security APIs and thus emphasizes that the general findings can also be adapted to security specific contexts. In order to analyze and judge, whether these approaches are already sufficient to treat Security APIs, specific usability aspects of Security APIs have to be identified. Concrete indications for poor usability in Security APIs can almost only be found in the results of security studies so far. Their purpose is, however, not to perform usability research for identifying general insights about the design of usable Security APIs. Consequently, such work focuses on

countermeasures and recommendations for improving security in terms of improved security mechanisms. To retrieve common and specific characteristics of Security API Usability, a bottom-up approach has been used, reviewing ten security analysis publications of the last four years, which do not study malicious attacks but logic errors. Here, it is possible to establish a relation to usability shortcomings in API design. Focusing on widely deployed security mechanisms, which are thus relevant for many developers, studies of the SSL/TLS protocol, the OAuth 2.0 Framework and OpenID Connect (Sakimura *et al.* 2014) have been selected.

5. Modeling the Current State of API Usability

As a first contribution of this paper this section introduces an elaborated API Usability model, which adopts the comprehensive usability model approach by Winter *et al.* (2007) and adapts it for the API context. Moreover, a consolidated view on the current research coverage as well as on untreated topics is integrated in addition (see Table 1). The considered current work reflects empirical studies only, because of their scientific validity and excludes guidelines based on expert knowledge or opinion. The model's two-dimensional vertical structure has been determined respecting the ISO 9241-11 (ISO 9241-11, 1998) usability framework. Hence, a developer's interaction with an API is influenced by the product (1.) and the context of use (2.). Following the approach by Winter *et al.* (2007), the product is differentiated between the physical interface (1.1) and the logical architecture (1.2). The documentation (1.3), which is a hardly separable part of an API, is added in addition. The context of use covers the user (2.1), the task (2.2), the equipment (2.3) and the environment (2.4). The fine-granular structure is populated with relevant publications. The space of API design decisions (1.2.3) has, e.g., been introduced by Stylos and Myers (2007). Some additional aspects, for which no prior research results could be found, are integrated as well. These can be identified by empty table cells.

The model's one-dimensional horizontal structure consists of action targets while a developer interacts with an API (A-K). These low-level details turned out to be appropriate for classifying previous research. Available API Usability recommendations are represented by positive (+), negative (-), positive and negative (±) or neutral (●) impact indicators. These are strongly related to the usability context of an empirical study [X]. Due to space constraints more detailed attributes such as the ones proposed in (Winter *et al.* 2007) have been suppressed.

The elaborated model visualizes the contemporary space of API Usability, which is not meant to have an immutable structure, if this is possible at all. Rather this is the current state of the research field, which can be supplemented and extended by missed or further findings. It enables an easy access for novices and it allows the uncovering of open research questions in particular. The model enables to derive that the available work, because of its basic nature and overall pertinence for all APIs, builds also a crucial fundament for the usability of Security APIs. But it also can be seen that still a lot of research has to be done to picture a holistic API Usability approach. In particular the current space of API Usability doesn't take specific

Interactions / Action Targets Product / Aspects	A) **Form** intention	B) **Find** an API	C) **Explore** the API	D) **Find** class, method, etc.	E) **Select** class, method etc.	F) **Creating** Code	G) **Find** Example	H) **Understand** API	I) **Debug** Code	J) **Learn** an API	K) **Maintain an** API
1. Product											
1.1 Physical Interface											
1.2 Logical architecture											
1.2.1 Form of appearance											
1.2.1.1 Libraries											
1.2.1.2 Toolkit											
1.2.1.3 Framework											
1.2.1.4 Web-APIs											
1.2.2 Programming Languages											
1.2.2.1 Idioms											
1.2.3 API-design decisions											
1.2.3.1 Structural Design											
1.2.3.1.1 Design Patterns											
1.2.3.1.1.1 Factory Patterns						–[1]					
1.2.3.1.2 Package design											
1.2.3.1.2.1 Number of classes				–[2]							
1.2.3.1.2.2 Sub packages				+[3]							
1.2.3.1.3 Configuration-based design											
1.2.3.1.3.1 Annotations						±[3]					
1.2.3.1.3.2 File-based						±[3]					
1.2.3.1.3.3 Fluent Interfaces						±[3]					
1.2.3.1.3.4 Combinations											
1.2.3.2 Class design											
1.2.3.2.1 Class names				+[4]				+[5]			
1.2.3.2.2 Design Patterns											
1.2.3.2.2.1 Create-Set-Call			+[6]								
1.2.3.2.3 Method placement				–[4], ●[4]							
1.2.3.2.4 Number of methods				+[2]							
1.2.3.2.5 Method names				+[2, 7]				+[5]			
1.2.3.2.6 Method overloads				–[7]							
1.2.3.2.7 Parameter Design				+[4, 5, 8]		–[2]					
1.2.3.2.8 Exceptions								–[8]			
1.2.3.2.9 Object creation											
1.2.3.2.9.1 Default constructors						+[6]					
1.2.3.2.9.2 Optional constructors						●[6]					
1.2.3.2.9.3 Required parameters						●[5], –[6]			●[6]		
1.2.3.2.9.4 Static methods						–[2]					
1.2.3.2.10 Access rules						+[9]					
1.2.4 Implementation of the functionality											
1.2.4.1 Performance											
1.2.4.2 Reliability											
1.2.5 Runtime Behavior											
1.3 Documentation											
1.3.1 Form											
1.3.1.1 Written documentation										+[9, 10]	
1.3.1.2 Examples						±[11]				+[12], ±[10]	
1.3.1.3 Runnable tests											
1.3.1.4 Comments in source code							+[12]		+[5]		
1.3.1.5 Web resources						± [8]					
1.3.2 Content											
1.3.2.1 Design concept					+[10]						
2. Context of use											
2.1 User											
2.1.1 User types											
2.1.2 Skills and knowledge											
2.1.3 Personal attributes											
2.1.3.1 Programming Style			±[13]							±[13]	
2.1.4 Expectations											
2.1.4.1 Mental models								+[14]			
2.1.4.2 Conventions										+[12]	
2.2 Task											
2.2.1 Security-critical requirements											
2.3 Equipment											
2.3.1 Development Environment											
2.3.1.1 Operating systems											
2.3.1.2 IDEs								+[7]			
2.3.1.3 Web resources										±[11]	
2.3.2 Development Tools											
2.3.2.1 Debugger											
2.3.2.2 Auto completion				+[7, 15]		+[16, 17]					
2.3.2.3 Text editor											
2.3.2.4 Web-search							+[11], ±[18]				
2.3.2.5 Recommendations			+[19]		+[20]	+[21]					
2.4 Environment											
2.4.1 Organizational environment											
2.4.1.1 Development processes											
2.4.2 Technical Environment											
2.4.2.1 Development Guidelines											+[22]
2.4.3 Physical Environment											
2.4.4 Social Environment											

Legend: + positive impact | – negative impact | ± pos. as well as neg. impact | ● neutral | [X] In the usability context of the empirical study X:
[1]: (Ellis *et al.* 2007); [2]: (Scheller and Kühn, 2012); [3]: (Scheller and Kühn, 2013b); [4]: (Stylos and Myers, 2008); [5]: (Piccioni *et al.* 2013);
[6]: (Stylos and Clarke, 2007); [7]: (Scheller and Kühn, 2013a); [8]: (Duala-Ekoko and Robillard, 2012); [9]: (Zibran *et al.* 2011); [10]: (Robillard, 2009);
[11]: (Brandt *et al.* 2009); [12]: (McLellan *et al.* 1998); [13]: (Clarke, 2011); [14]: (Stylos *et al.* 2006); [15]: (Bruch *et al.* 2009); [16]: (Mooty *et al.* 2010);
[17]: (Omar *et al.* 2012); [18]: (Stylos and Myers, 2006); [19]: (Duala-Ekoko and Robillard, 2011); [20]: (Asaduzzaman *et al.* 2015); [21]: (Zhong *et al.* 2009);
[22]: (Espinha *et al.* 2014)

Table 1: The space of API Usability

usability characteristics of Security APIs into account.

6. Towards the Usability Evaluation of Security APIs

When analyzing the outcomes of recent security studies in the light of API Usability, it becomes clear that current API Usability research already provides some baseline approaches and tools for the usability evaluation of Security APIs. This is by far not sufficient enough for this special context of use, though. With the methodology described in Section 4, it has been possible to derive eleven Security API specific usability characteristics, which are introduced in the subsequent sections. Concrete usability aspects, with a lower level of abstraction, like those listed in the space of API Usability (see Table 1), have to be elaborated by further evaluations of these identified characteristics. Thus, the goal of consecutive research should be to extent the introduced API Usability model introduced in Section 5 for the particular space of Security API Usability.

6.1. End-user Protection

Intentional or unintentional defective software implementations can cause compromised user information security, often without the users even noticing. Thus, especially Security APIs must be designed while keeping the end-user's security in mind, also because this is its actual intention in the first place. The End-user Protection characteristic describes an API's ability to reduce or eliminate this dependency from the programmers. The "User Protection" characteristic has been proposed by Fahl *et al.* (2013) and they have defined it as a limitation of a developer's capabilities to prevent an invisible risk for end-user data. This definition has been based on the observation that developers of mobile applications take full responsibility for integrating of security functionalities as well as for communicating any security relevant information to end-users (Fahl *et al.* 2012), (Fahl *et al.* 2013).

In (Georgiev *et al.* 2012) corresponding issues have been identified for various SSL/TLS libraries, software development kits and middleware. Wang *et al.* (2012) refer to a due diligence for application developers who implement relying party components in single sign on systems. According to Wang *et al.* (2012) application developers are finally responsible for orchestrating user applications, relying parties and identity providers in a secure manner. But this is also true for programmers who implement libraries, software development kits or frameworks. An incorrect handling of tokens caused by unusable Security APIs in any of those software products could lead to the unauthorized access of user accounts even without possessing any credentials. Thus, a due diligence exists for all persons involved in a software development process to ensure the required End-user Protection.

6.2. Case Distinction Management

Error prevention and the handling of exceptions and errors are crucial aspects of APIs in general, but are indispensable for Security APIs. The term Case Distinction Management is introduced to name all considerable events, which might happen. In

the context of Security APIs, special attention needs to be drawn to exceptional events like e.g. a negative certificate validation, since this does not hinder security measures, but it is an essential implication to preserve them. As such cases happen frequently, they should not be treated as rare exceptions or software errors. In fact, these cases have to be well managed by an API design that empowers developers in handling case distinctions correctly.

The verification process of certificates, e.g., is a crucial part of the SSL/TLS protocol for establishing a trust relationship between client and server. This includes e.g. chain-of-trust verification, hostname verification and the review of the certificates' status. Georgiev *et al.* (2012) found that security critical events are indicated inconsistently by runtime errors, return values or internal flags, which have to be validated by additional function calls. This already resulted in the overriding of security functionalities in deployed software.

6.3. Adherence to Security Principles

More than forty years ago, Saltzer and Schroeder (1975) described fundamental principles of information security, which are still approved and prevailing. Since then, further principles have been evolved mostly with a specific focus, such as the "OWASP Coding Practices" (OWASP 2010) and documented risks like the "CWE/SANS Top 25 Most Dangerous Software Errors" (CWE 2011). By taking these security principles into account in the context of usable Security APIs, this introduced characteristic communicates explicitly, that adhering to the principles in API design will increase the effective use of the interfaces.

Several different examples where API design decisions are violating these and other security principles can be found in security studies. One of them is the Android SSL/TLS library (Google, 2016). In some parts it contradicts the "economy of mechanism" principle. Android applications are normally exchanging data with just a few hosts. Still, the Android system commonly trusts over 100 Certificate Authorities (CA) by default. Mechanisms like certificate or public key pinning, which allow selecting only needed CAs, have to be self-implemented by developers. As a consequence, they are forced to take a higher security risk by default. It has been shown that certificate or public key pinning is not in widespread use for Android (Fahl *et al.* 2012) or Web (Kranch and Bonneau, 2015) applications.

6.4. Testability

The security studies that this analysis is built upon are prime examples for how difficult it is for common software developers to test security mechanisms in their applications. Much effort and expertise is needed to develop test beds for static code analysis and conduct manual code audits. Still, software developers need to see clearly if security mechanism have been adopted, integrated and deployed correctly and this needs to be examined not only by self-written unit test code. Due to a lack of time and expertise or sometimes also the blind faith, some developers do not test integrated libraries or used frameworks at all (Fahl *et al.* 2013). Not less badly, even

modified code for testing purposes finds its way in deployed software products, causing security flaws (Georgiev *et al.* 2012). Supporting and reliable test routines, written by security experts, e.g., for certificate validation and adversarial testing in TLS implementations (Brubaker *et al.* 2014), should be available and easy to apply for programmers in typical use-cases.

6.5. Constrainability

It is in the nature of programming to customize code in order to meet the requirements. But customization in the context of security appears to cause substantial risks. There are functionalities like data validation where constraints represent essential means to establish security (Kern, 2015), e.g. against Cross-Site-Scripting. Georgiev *et al.* (2012) state "*in general, disabling proper certificate validation appears to be the developers' preferred solution to any problem with SSL libraries."* These findings seem to legitimate constraining the usage of a Security API and indicate a tradeoff between flexibility and error susceptibility in this context. If customization tends to be the rule rather than the exception, though, the design decisions of a Security API are most probably not appropriate for its target audience and thus has to be reconsidered. Evaluations have to show in which situations usage constraints support or hinder the usability of Security API.

The configurability of security mechanisms might be a usable instrument to force constraints. (Fahl *et al.* 2013) have proposed an approach for SSL/TLS development on Android, based on configuration instead of writing source code. Yet there have not been conducted any comparing usability studies to see if this is suitable in general for Security APIs. Examples showing the opposite can be found in emerging HTTP Strict Transport Security (HSTS) (Hodges *et al.* 2012) implementations, though. One crucial part of HSTS application is the HTTP header configuration. First deployed utilizations have not been in conformance with the standard, used malformed headers and misused header values mostly resulting in undermined security of end-users (Kranch and Bonneau, 2015). This makes obvious that the configuration of security functionalities, which also is an API aspect, has uncovered usability issues. This confirms the continuing trend of overriding security functionalities encouraged by unusable APIs for new security features in addition.

6.6. Information Obligation

The end-user as well as the application developer using APIs have to be well informed of security relevant specifics. The major challenge is to provide crucial information at the right place, in the right moment and in a usable manner (Garfinkel and Richter Lipford, 2014). If an application is designed without any protection means for confidentiality, e.g. ignoring SSL/TLS connections, an end-user will be incapable of responding to this situation, due to the lack of information. The same is true for Security APIs, which do not communicate security implications intrinsically by documentation or via development tools to the developer. A Security API must support application developers in communicating security relevant information to the end-user in a usable way.

6.7. Degree of Reliability

Application developers, who see themselves confronted with a security related programming task, need reliable information resources and APIs. This have been uncovered by interviews conducted with developers who implemented security mechanisms incorrectly (Fahl *et al.* 2013). When running into problems or unknown terrain, programmers make heavy use of Web resources. Still, the presented code fragments might come from an equal inexperienced source and should not be trusted without additional examination. Therefore reliable testing tools, as well as visible trust indicators preferably issued by a reliable institution are needed. Usability evaluations should examine what kind of resources application developers actually trust. This could be measured by a self-assessment asking for the level of confidence in own security relevant implementations. The results should indicate who should primarily deliver approved information or well tested code examples for various use cases to match the developers' expectations.

6.8. Security Prerequisites

Security APIs have mandatory prerequisites, which have to be fulfilled by developers to apply the provided security functionality effectively. It has become evident that Security Prerequisites are unknown, unclear or misused in many cases. Relying parties implementing OAuth 2.0 missed to utilize SSL/TLS for the protocol being confidential (Sun and Beznosov, 2012). R. Wang *et al.* (2012) notice shortcomings in correctly protecting and verifying tokens in single sign on systems. They suspect a missing comprehension of security implications to be the reason. Li and Mitchell (2014) were able to identify deficiencies against Cross-Site Request Forgery (CSRF) attacks in productive services caused by misused parameters. Static and guessable values have e.g. been used instead of unique character sequences. API designers have to respect their obligation to inform and support developers to counteract security risks caused by non-fulfillment of security prerequisites.

6.9. Execution Platform

Security APIs are needed in several different execution platforms. To be securely applicable they have to be tailored for different ecosystems. This includes existing platform specific possibilities and risks in particular. Software vulnerabilities can be traced back to API design, which does not consider execution platform specifics, which are exploitable by attackers and thus are able to compromise security functionalities (R. Wang *et al.* 2012). Using the OAuth 2.0 client-flow in web browsers, e.g., expose tokens to various browser specific attack vectors. Thus, Sun and Beznosov (2012) "*believe that OAuth 2.0 at the hand of most developers – without a deep understanding of web security – is likely to produce insecure implementations.*" Chen *et al.* (2014) call attention to sensitive differences between mobile and Web platforms showing difficulties in adapting OAuth for mobile applications, again leading to high numbers of vulnerable implementations. SSL/TLS was intentionally designed for the browser environment. Its prevalent employment for transport security in non-browser applications such as Android (Fahl *et al.* 2012,

H. Wang *et al.* 2015), iOS (Fahl *et al.* 2013) and other platforms (Georgiev *et al.* 2012) lead to widespread man-in-the-middle vulnerabilities potentially affecting millions of end-users. From this follows that Security API design process has to consider target execution platforms and needs of their developers. A central question here is, how security implications can be communicated effectively during development processes.

6.10. Delegation

The delegation of implementing security functionalities or informing end-users to unspecialized developers can be seen in already mentioned cases where this shift of responsibility had lead to incorrect implementations. Georgiev *et al.* (2012) found several SSL/TLS libraries delegating hostname verification or certificate validation to higher-level software. Brubaker *et al.* (2014) even encountered missing code in an "if" condition which just provided a comment of the API designer. This is especially critical if API users assume a complete security solution and instead get just a partial coverage. In such cases developers have to get well informed about open implementation tasks to fulfill security prerequisites. Even better would be to suggest concrete solutions or reliable best practices.

6.11. Implementation Error Susceptibility

The overall goal of Security API usability research should be to minimize the error susceptibility, which significantly rises by ignoring each aforementioned characteristic. Research need to strive a holistic approach to address end-user protection, case distinction management, adherence to information security principles, testability, constrainability, information obligation, degree of reliability, security prerequisites, execution platforms and delegation.

7. Conclusion and Outlook

Security APIs provide access to crucial building blocks that are indispensable in contemporary and future software systems. Thus, the incorrect or insufficient use of such APIs lead to extensive security flaws, which compromise end-user information security. As one reason for this problem, unusable API design decisions have been identified by several security studies. To effectively counteract these issues, the usability of Security APIs has to be improved by further research in the general field of API Usability and by initiating specific research activities in Security API Usability. For this purpose a comprehensive model to cover the current space of API Usability and to point out examined as well as open research questions has been introduced. This model has been further enriched by an extensive literature analysis of security studies. By this, it has been possible to identify eleven security specific usability characteristics, which has to be subject in future evaluations of Security APIs. Thereby, the present paper laid the ground for future research and development work in this field. The introduced eleven specific usability characteristics of Security APIs might still be an incomplete set of relevant topics. Future research will be

conducted to confirm the set in order to obtain a validated baseline for the usability evaluation of Security APIs.

8. Acknowledgment

This work has been funded by the German Federal Ministry for Economic Affairs and Energy (Grant no. 01MU14002).

9. References

Apache (2016). "Welcome to Apache Axis2/Java", http://axis.apache.org/axis2/java/core/, (Accessed 21 March 2016)

Asaduzzaman, M., Roy, C. K. , Monir, S. and Schneider, K. A. (2015). "Exploring API method parameter recommendations". *International Conference on Software Maintenance and Evolution (ICSME '15)*. Bremen, DE.

Bond, M. K. (2004). "Understanding Security APIs". *Dissertation*. University of Cambridge.

Brandt, J., Guo, P. J., Lewenstein, J., Dontcheva, M. and Klemmer, S. R. (2009). "Two studies of opportunistic programming: interleaving web foraging, learning, and writing code". *SIGCHI Conference on Human Factors in Computing Systems (CHI '09)*. Boston, MA, U.S.A.

Brubaker, C., Jana, S., Ray, B., Khurshid, S. and Shmatikov, V. (2014). "Using Frankencerts for Automated Adversarial Testing of Certificate Validation in SSL/TLS Implementations". *35th IEEE Symposium on Security and Privacy (S&P '14)*. San Jose, CA, U.S.A.

Bruch, M., Monperrus, M. and Mezini, M. (2009). "Learning from examples to improve code completion systems". *7th joint meeting of the European software engineering conference and the ACM SIGSOFT symposium on The foundations of software engineering. (ESEC/FSE '09)*. Amsterdam, NL.

Chen, E., Pei, Y., Chen, S., Tian, Y., Kotcher, R. and Tague, P. (2014). "OAuth Demystified for Mobile Application Developers". *21st ACM SIGSAC Conference on Computer and Communications Security (CCS '14)*. Scottsdale, AZ, U.S.A.

Clarke, S. (2011). "How Usable Are Your APIs?" *Making software: what really works, and why we believe it.* Oram, A. and Wilson, G. (Ed.). 1st ed., Theory in practice. Beijing: O'Reilly, S. 545–565. ISBN: 978-0-596-80832-7.

CWE (2011). "2011 CWE/SANS Top 25 Most Dangerous Software Errors Version: 1.0.3". Christey, S. (Ed.). *The MITRE Corporation*. http://cwe.mitre.org/top25/

Dierks, T. and Rescorla, E. (2008). "The Transport Layer Security (TLS) Protocol Version 1.2." *RFC 5246, Proposed Standard*. Internet Engineering Task Force.

Duala-Ekoko, E. and Robillard, M. P. (2011). "Using Structure-Based Recommendations to Facilitate Discoverability in APIs". *25th European Conference on Object-Oriented Programming (ECOOP '11)*. Lancaster, U.K.

Duala-Ekoko, E. and Robillard, M. P. (2012). "Asking and Answering Questions about Unfamiliar APIs: An Exploratory Study". *34th International Conference on Software Engineering (ICSE '12)*. Zurich, CH.

Ellis, B., Stylos, J. and Myers, B. (2007). "The Factory Pattern in API Design: A Usability Evaluation". *29th International Conference on Software Engineering (ICSE '07).* Minneapolis, MN, U.S.A.

Espinha, T., Zaidman, A. and Gross, H.-G. (2014). "Web API growing pains: Stories from client developers and their code". *IEEE Conference on Software Maintenance, Reengineering and Reverse Engineering, Software Evolution Week (CSMR-WCRE '14).* Antwerp, BE.

Fahl, S., Harbach, M., Muders, T., Smith, M., Baumgärtner L. and Freisleben, B. (2012). "Why Eve and Mallory Love Android: An Analysis of Android SSL (In)Security", *ACM Conference on Computer and Communications Security (CCS '12)*. Raleigh, NC, U.S.A.

Fahl, S., Harbach, M., Perl, H., Koetter, M. and Smith, M. (2013). "Rethinking SSL Development in an Appified World", *ACM SIGSAC Conference on Computer and Communications Security (CCS'13)*. Berlin, DE.

Freier, A., Karlton, P. and Kocher, P. (2011). "The Secure Sockets Layer (SSL) Protocol Version 3.0" *RFC 6101, Historic*, Internet Engineering Task Force.

Garfinkel, S. and Richter Lipford, H. (2014). "Usable Security: History, Themes, and Challenges", Bertino, E. and Sandhu, R. (Ed.) *Synthesis Lectures on Information Security, Privacy, and Trust*, Morgan & Claypool, San Rafael, ISBN: 978-1-62705-529-1.

Georgiev, M., Iyengar, S., Jana, S., Anubhai, R., BonehD. and Shmatikov, V. (2012). "The Most Dangerous Code in the World: Validating SSL Certificates in Non-Browser Software". *ACM Conference on Computer and Communications Security (CCS' 12)*. Raleigh, NC, U.S.A.

Google (2016). "Android Developers - Best Practices for Security & Privacy", https://developer.android.com/training/best-security.html, (Accessed 19 March 2016)

Green, M. and Smith, M. (2015). "Developers Are Users Too: Designing Crypto and Security APIs That Busy Engineers and Sysadmins Can Use Securely". *Talk at the USENIX Summit on Hot Topics in Security (HotSec '15)*. Washington, D.C., U.S.A.

Hardt, D. (2012). "The OAuth 2.0 Authorization Framework", *RFC 6749, Proposed Standard.* Internet Engineering Task Force.

Hodges, J., Jackson, C. and Barth, A. (2012). "HTTP Strict Transport Security (HSTS)", *RFC 6797, Proposed Standard*. Internet Engineering Task Force.

ISO 9241-11 (1998). "Ergonomic requirements for office work with visual display terminals (VDTs) – Part 11: Guidance on usability" *International Standard*, The International Organization for Standardization

Kern, C. (2015). "Preventing Security Bugs through Software Design". *Talk at the 24th USENIX Security Symposium (USENIX Security '15)*. Washington, D.C., U.S.A.

Kranch, M. and Bonneau, J. (2015). "Upgrading HTTPS in Mid-Air: An Empirical Study of Strict Transport Security and Key Pinning". *The Network and Distributed System Security Symposium (NDSS '15)*. San Diego, California, U.S.A.

Li, W., and Mitchell, C. J. (2014). "Security issues in OAuth 2.0 SSO implementations", *17th International Information Security Conference (ISC '14),* Hong Kong, CN.

McLellan, S. G., Roesler, A. W., Tempest, J. T. and Spinuzzi, C. I. (1998). "Building More Usable APIs". *IEEE Software* 15.3, S. 78–86.

Mooty, M., Faulring, A., Stylos, J. and Myers, B. A. (2010). "Calcite: Completing Code Completion for Constructors Using Crowds". *IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC '10)*. Leganes, ES.

Omar, C., Yoon, Y. S., LaToza, T. D. and Myers, B. A. (2012). "Active code completion". *34th International Conference on Software Engineering (ICSE '12)*. Zurich, CH.

OWASP (2010). "OWASP Secure Coding Practices - Quick Reference Guide Version 2.0". *OWASP - The Open Web Application Security Project*. https://www.owasp.org/index.php/OWASP_Secure_Coding_Practices_-_Quick_Reference_Guide

OWASP (2013). "OWASP Top 10 - 2013 - The Top 10 Most Critical Web Application Security Risks". Williams, J. and Wichers, D. (Ed.). *OWASP - The Open Web Application Security Project.* https://www.owasp.org/index.php/Category:OWASP_Top_Ten_Project

Piccioni, M., Furia, C. A. and Meyer, B. (2013). "An Empirical Study of API Usability". *ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM '13)*. Baltimore, Maryland, U.S.A.

Robillard, M. P. (2009). "What Makes APIs Hard to Learn? Answers from Developers". *IEEE Software* 26.6, S. 27–34.

Sakimura, N., Bradley, J., Jones, M., de Medeiros, B. and Mortimore, C. (2014). "OpenID Connect Core 1.0 incorporating errata set 1", *Final Specification*, The OpenID Foundation.

Saltzer, J. H., and Schroeder, M. D. (1975). "The protection of information in computer systems". *Proceedings of the IEEE* 63.9, S. 1278–1308.

Scheller, T. and Kühn, E. (2012). "Influencing Factors on the Usability of API Classes and Methods". *19th International Conference and Workshops on Engineering of Computer-Based Systems (ECBS '12)*. Novi Sad, RS.

Scheller, T. and Kühn, E. (2013a). "Influence of Code Completion Methods on the Usability of APIs". *12th IASTED International Conference on Software Engineering (SE '13)*. Innsbruck, AT.

Scheller, T. and Kühn, E. (2013b). "Usability Evaluation of Configuration-Based API Design Concepts". *1st International Conference on Human Factors in Computing & Informatics. South (CHI '13)*. Maribor, SI.

Steel, G. (2011). "Formal Analysis of Security APIs", Van Tilborg, H. C. A. and Jajodia, S. (Ed.). *Encyclopedia of Cryptography and Security*, Springer, Boston, MA, S. 492–494, ISBN: 978-1-4419-5907-2

Stylos, J. and Clarke S. (2007). "Usability Implications of Requiring Parameters in Objects' Constructors". *29th International Conference on Software Engineering (ICSE '07).* Minneapolis, MN, U.S.A.

Stylos, J., Clarke, S., Myers, B. A. (2006). "Comparing API Design Choices with Usability Studies: A Case Study and Future Directions". *18th Workshop of the Psychology of Programming Interest Group (PPIG '06)*. Brighton, UK.

Stylos, J. and Myers, B. A. (2006). "Mica: A Web-Search Tool for Finding API Components and Examples". *IEEE Symposium on Visual Languages and Human-Centric Computing. (VL/HCC '06)*. Brighton, U.K.

Stylos, J. and Myers, B. A. (2007). "Mapping the Space of API Design Decisions", *IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC '07)*. Coeur d'Alene, ID, U.S.A.

Stylos, J. and Myers, B. A. (2008). "The Implications of Method Placement on API Learnability". *The 16th ACM SIGSOFT International Symposium on Foundations of Software Engineering (SIGSOFT '08/FSE-16)*. Atlanta, GA, U.S.A.

Sun, S.-T. and Beznosov, K. (2012). "The Devil is in the (Implementation) Details: An Empirical Analysis of OAuth SSO Systems". *The ACM Conference on Computer and Communications Security (CSS '12),* Raleigh, NC, U.S.A.

Wang, H., Zhang, Y., Li, J., Liu, H., Yang, W., Li, B. and Gu, D. (2015). "Vulnerability Assessment of OAuth Implementations in Android Applications". *31st Annual Computer Security Applications Conference (ACSAC '15)*. Los Angeles, CA, U.S.A.

Wang, R., Chen, S., and Wang, X.F. (2012). "Signing Me onto Your Accounts through Facebook and Google: A Traffic-Guided Security Study of Commercially Deployed Single-Sign-On Web Services", *IEEE Symposium on Security and Privacy (S&P '12)*. San Francisco, CA, U.S.A.

Winter, S., Wagner, S. and Deissenboeck, F. (2007). "A Comprehensive Model of Usability". *Engineering Interactive Systems Joint Working Conferences (EHCI '07, DSV-IS '07, HCSE '07. EIS '07)*. Salamanca, ES.

Zhong, H., Xie, T., Zhang, L., Pei, J. and Mei, H. (2009). "MAPO: Mining and Recommending API Usage Patterns". *23rd European Conference on Object-Oriented Programming (ECOOP '09)*. Genoa, IT,

Zibran, M. F., Eishita, F. Z. and Roy, C. K. (2011). "Useful, But Usable? Factors Affecting the Usability of APIs". *18th Working Conference on Reverse Engineering (WCRE '11)*. Limerick, IE.