Int J Software Informatics, Vol.xx, No.xx, January 2015, pp. 1–27 E-mail: ijsi@iscas.ac.cn
International Journal of Software and Informatics, ISSN 1673-7288 http://www.ijsi.org
©2015 by Institute of Software, Chinese Academy of Sciences. All rights reserved. Tel: +86-10-62661040

An Ontology for Secure Web Applications

Marianne Busch and Martin Wirsing

{busch, wirsing}@pst.ifi.lmu.de

Ludwig-Maximilians-Universität München, Germany

Dedicated to Bernd Krieg-Brückner

Abstract It is commonly known that most applications suffer from security holes that are sooner or later exploited. One reason is that for developers the term “security” is difficult to grasp. Many security properties exist and there are many methods to enforce them or to avoid implementing common vulnerabilities in applications.

Ontologies can help to get an overview of web security and to structure this domain by relating relevant assets, methods, tools, security properties, vulnerabilities and threats (referred to as knowledge objects). In this paper, we present a novel ontology with a focus on secure web applications, called SecWAO. It is based on the Context model of SecEval, which is a domain model tailored to describe knowledge objects. By providing an overview, SecWAO supports teaching purposes and web developers when specifying security requirements or making design decisions.

Key words: security, web security, web engineering, ontology, taxonomy, overview, UML

M. Busch and M. Wirsing. An Ontology for Secure Web Applications. *Int J Software Informatics*, 2015, xx(xx): 1–27. **URL/xx.pdf**

1 Introduction

As web applications are becoming increasingly complex they also bare many risks for organizations and users. According to Kaspersky more than one billion attacks were launched from web resources in 2014 [Kas14]; for 2013 Symantec reports breaches of more than 550 million identities [Sym14], and Cenzic estimates that 96% of all web applications contain security holes [Cen14]. One reason is that for developers the term “security” is difficult to grasp. Many security properties exist and there are many methods to enforce them or to avoid implementing common vulnerabilities in web applications.

In such a situation, ontologies can help users and developers to get an overview and a common understanding of a domain. An ontology consists of a set of concepts and relations between these concepts, describing their various features and at-

† This work has been partially sponsored by the European Union under the FP7-project ASCENS, 257414.
‡ Manuscript received 2015-01-11; revised 20015-xx-xx; accepted 2015-xx-xx; published online 2015-xx-xx

tributes[1]. Ontologies have many uses in computer science, e.g. in artificial intelligence for representing knowledge [UG96], in autonomous systems for representing route graphs [KFL$^+$04], or in the Semantic Web for merging information and for searching across different domains [GB02]. In the MMISS project (coordinated by Bernd Krieg-Brückner) ontologies were the basis for creating a coherent set of teaching materials in the area of formal methods [KHL$^+$02]. The Common Body of Knowledge CBK [CBK15] on engineering secure software and services and the SecEval framework [BKW14b] for evaluating security-related artifacts are structured according to ontologies related with software engineering and security. Various languages are used for formalizing ontologies; e.g. CBK, SecEval and the MMISS ontology are represented in UML [Obj11]; another well-known ontology language is OWL [W3C].

In this paper, we present a novel ontology for secure web applications, called SecWAO, which is intended to support web developers when specifying security requirements or making design decisions. Similarly to the so-called Security Context model of SecEval, SecWAO distinguishes between methods, notations, tools, categories, assets, security properties, vulnerabilities, and threats. For each of these classes it provides relevant instances and relates them by different kinds of relationships such as “belongs to”, “depends on”, or “uses”. For example, availability, system integrity, noninterference, and data confidentiality, and data authenticity are security properties; cryptography, authentication, logging, auditing, error handling, session management, and input validation are methods; and web applications, databases, and transmissions are assets.

The remainder of this paper is structured as follows: section 2 presents related work and section 3 introduces the Security Context model of SecEval. In section 4 we present our SecWAO ontology before we conclude in section 5.

Personal Note: Bernd and Martin (the second author of this paper) have met almost forty years ago when both were research assistants of Friedrich L. Bauer and Klaus Samelson and collaborated within the Project “Computer-aided Intuition-guided Programming” (CIP) in the DFG Collaborative Research Center “Programming Technology” (SFB 49). The CIP project can be seen as an early precursor of the now popular model-driven development; its aim was to use transformations for the systematic development of programs from formal specifications. Together with several other researchers (such as Manfed Broy and Peter Pepper) Bernd and Martin worked on techniques for program transformation, algebraic specification and programming methods (cf. e.g. [BBD$^+$81, BK80, BMPW86]) and designed the wide spectrum language CIP-L which was one of the first languages comprising constructs for functional and procedural programming as well as for formal specification [BBB$^+$85]. Later, Bernd and Martin cooperated in the IFIP Working Group 1.3 [M$^+$], the ESPRIT Working Groups COMPASS I and II [KB97], the BMFT Cooperative Research Project KORSO on “Correct Software” [BJ95], and the BMBF Cooperative Project MMISS “MultiMedia Instruction in Safe and Secure Systems” [KB$^+$04]. During the latter project Bernd introduced Martin to ontologies and stimulated the system-

[1] Note that the origin of the notion of ontology stems from philosophy where it has a different meaning; in philosophy, ontology stands for the study of the nature of being (see e.g. [Hes14])

atic classification of the area of formal methods by an UML-based ontology (see e.g. [KHL$^+$02]). Our paper here is written in the spirit of these ideas: the UML-based ontology SecWAO is used to structure the domain of web security.

Cooperating with Bernd is a very pleasant experience; he is interested and knowledgeable in many scientific areas ranging from programming and formal methods to cognitive science and robotics, he is an excellent project coordinator, and last but not least, a good friend. We are looking forward to many further inspiring exchanges with Bernd.

2 Related Work

This section, which is partly based on [BKW14b], introduces related work in the area of ontologies for (web application) security.

It is commonly known that a good way to secure applications is to focus on security during the Software Development Life Cycle (SDLC). For example, companies use Microsoft's Security Development Life Cycle (SDL) [LH05] or they apply ISO 27001 [ISO13]. ISO 27001 defines an information security management system that requires the specification of security guidelines for policies, processes and systems within an organization.

An example for supporting secure development of web applications along the SDLC is the Open Web Application Security Project (OWASP). It comprises, beyond others, a set of guides for web security requirements, cheat sheets, a development guide, a code review and a testing guide, an application security verification standard (ASVS), a risk rating methodology, tools and a Top 10 of privacy risks as well as a Top 10 of web security vulnerabilities [OWA13].

A technique that can be used in the scope of a secure SDLC, is an ontology that describes and relates concepts (which are also called "knowledge objects"). In the domain of Software Engineering, knowledge objects are mainly assets, methods[2], tools[3], security properties[4], vulnerabilities and threats. Some knowledge bases use ontologies as a structure for the knowledge objects they contain, as e.g., [W$^+$15, K$^+$15, CBK15]. In this case, the term "ontology" refers to the structure of abstract concepts, not to the resulting ontology or knowledge base that comprises instances of concepts.

Ontologies aim at a shared understanding of a domain. In Software Engineering, this is especially helpful when large teams work together and experts are recruited for varying periods of time. Besides, knowledge bases aim at supporting software and security engineers to learn about the state-of-the-art to avoid reinventing the wheel [FFL13].

In addition, ontologies and their underlying structures, which are sometimes referred to as "(meta-)models", can support developers by facilitating the evaluation of methods, notations[5] or tools that could be used. SecEval is a conceptual framework for enabling software developers to evaluate methods, tools and notations for secure

[2] A method is a strategy for reaching a goal. For example countermeasures against common vulnerabilities are methods.
[3] A tool refers to software that supports methods.
[4] Synonyms for security properties are security goals or security features.
[5] A notation is a format to convey information.

software in a structured way [BKW14b, BKW14a]. It consists of three packages: a security context model describing attributes and relations of assets, security properties, vulnerabilities and threats as well as methods, notations and tools; a data collection model, which records how data is gathered when researchers or practitioners do research to answer a question; and a data analysis model specifying how reasoning on previously collected data is done.

Our SecWAO ontology for modeling the main notions of *web* application security is an instance of the Security Context model. SecEval's focus is on security aspects, even though its UML class model can describe non-security mechanisms as well. The next section introduces SecEval's Security Context model in more detail. A prototype of an implementation of SecEval as a knowledge base is presented in [Rei14].

Regarding web security, Salini and Kanmani [SK13] present an approach for creating an ontology of security requirements, including the concepts of assets, web applications, vulnerabilities, threats, security requirements and stakeholders. Although they define relations between these concepts, they do not provide attributes to describe them, as e.g., SecEval does. They mention some examples for these concepts, but do not provide a concrete ontology. However, we recommend their related work section for a chronological list of security-related ontologies.

In [DKF$^+$03, DKF05] Denker et al. provide examples of elements and their relations for secure web services. In contrast to SecWAO, their ontology is restricted to a tree of subclasses and properties of the concepts "Credential" and "SecurityMechanism". They use a high level of abstraction and due the lack of different concepts, they e.g., group "authentication, authorization, access control, data integrity, confidentiality, privacy, exposure control, anonymity, negotiation, policy, key distribution" inside of "SecurityNotation", which itself is a descendant of "SecurityMechanism". In SecEval and SecWAO, we differentiate between security properties like confidentiality, methods like authentication and assets like credentials.

In the following, ontologies and structures of ontologies from the domain of security are discussed that do not focus on web application security like SecWAO does.

The CBK (Common Body of Knowledge) [BEHS12] defines a model to collect and describe general methods, techniques, notations, tools and standards. We used the CBK as a starting point for our SecEval approach in [BKW14a]. In contrast to the CBK, SecEval explicitly focuses on security-related features by providing a model that distinguishes assets, vulnerabilities and threats and supports a fine-grained description of methods and tools according to their usage in the SDLC. The CBK is implemented as a semantic Wiki [CBK15] and serves as an open knowledge base.

Besides the work of Krieg-Brückner et al. [KHL$^+$02] and the CBK, another approach with teaching purposes regarding security is called Cyber Security Learning by security Ontology Browsing (SLOB). This project seems to be abandoned, as few information is available and the prototype[6] does not fully load. In addition, the same working group provides an editor for security ontologies, called Security Ontology eXpert tool (SOX)[7]. In SOX, an attribute that can be specified is an excerpt of books that describe a concept. The approach to enrich an existing ontology by exploring

[6] SLOB. `http://cis.csi.cuny.edu:8080/SLOB/` [C$^+$14]
[7] SOX. `http://cis.csi.cuny.edu/~project/SKATClient/` [C$^+$14]

textbook indexes is described in [WCG13]. The OWL ontology enriched in [WCG13] is presented by Herzog et al. in [HSD07]. It primarily focuses on the classification of assets, threats, vulnerabilities and countermeasures and also contains some web-related threats, as e.g., cross-site-scripting. However, it lacks common risks for web applications, as cross-site-request forgery, clickjacking or methods for session management (cf. section 4).

In [FE09], Fenz and Ekelhart introduce an ontology based on OWL that comprises (among others) assets, security attributes, threats, vulnerabilities and controls. Additionally, they use formal axioms to test if all necessary information to describe a concept is specified in a concrete ontology. Unlike SecWAO, their ontology is rather abstract, even so they also include physical security, as e.g., dumpster diving, safety doors or smoke detectors. Kim et al. [KLK07] use OWL to create an ontology that consists of several parts, like main security, credentials, algorithms, assurance or semantic web services' security. The main ontology includes security objectives and security concepts like protocols, mechanisms (e.g., for securing a network) or policies. Aside from cookies and some internet protocols, web application security is not mentioned. Unfortunately, the server hosting the full ontology files seems no longer to be online. In contrast to SecEval and the CBK, neither [HSD07], [FE09] nor [KLK07] allow for the representation of methods that are not directly related to security, but are used in the context of secure development.

Structures of ontologies can also be used as a basis for eliciting security requirements, as already seen in [SK13]. The i* [RWT14] metamodel is the basis of a vulnerability-centric requirements engineering framework introduced in [EYZ10]. This extended, vulnerability-centric i* metamodel aims at analyzing security attacks, countermeasures, and requirements based on vulnerabilities. Similar to our approach, the metamodel is represented using UML class models. Instances of this metamodel use an i*-specific representation, which is not based on UML. Main elements of the metamodel are: vulnerability, attack (which can exploit a vulnerability; executed by an actor), effect and security impact (e.g., on a resource). Although the terms ontology or taxonomy are not used in [EYZ10], Elahi et al. provide a detailed example of a browser and a web server, including threats by a hacker and a fake web site. For this example they analyze countermeasures for a concrete system in order to estimate the risk of vulnerability exploitation. As they do not provide an ontology, we think that SecWAO might enable security engineers to systematically examine relevant security mechanisms of web applications when modeling with i*. For evaluations of existing security ontologies, the interested reader is referred to [SSCW12, BLVG^{+}08].

3 The underlying "Security Context model"

In the following, SecEval's Security Context model is introduced. It is the UML class model for the instance model (object model) that will represent our ontology in section 4. This section is based on [BKW14b, BKW14a] and includes minor changes in the model, which has been further developed in the meantime.

The Security Context model provides a structure for the classification of (security-related) methods, notations and tools together with security properties, vulnerabilities and threats. Within this model a security feature is a class element, called

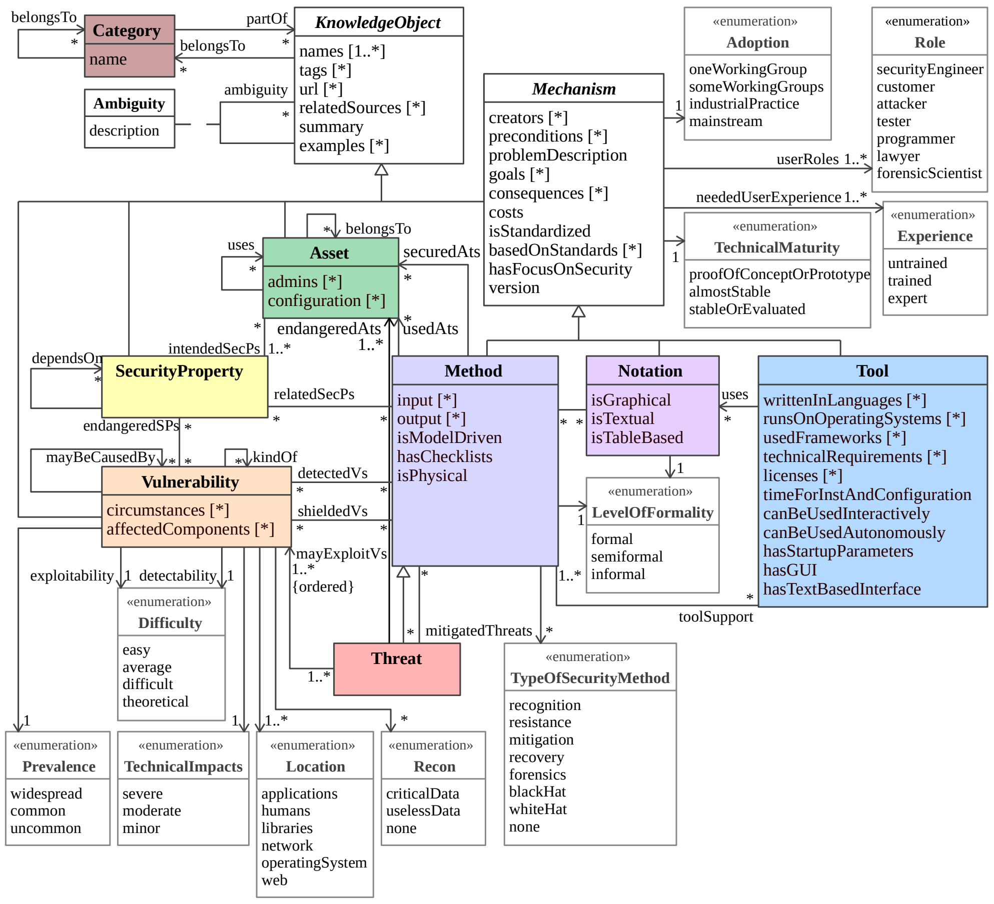


Fig. 1. SecEval: Security Context

`SecurityProperty`, and an abstract class `Mechanism` is used from which the classes `Method`, `Notation` and `Tool` inherit common attributes such as `goals` or `costs`.

In figure 1, enumerations are edged in grey and the background of classes which can directly be instantiated is colored and the names of them are not italic. The same colors are used in following figures. All attributes and roles are typed; however the types are not shown in the figures due to brevity. The main characteristics are specified as boolean types (can.., has.., is..).

A Mechanism is described by a problem statement, by the goals it strives for, by its costs and by the consequences it implies. Mechanisms can be based on standards or be standardized themselves. Before applying a mechanism, the preconditions that are necessary for using it have to be fulfilled. Furthermore, an estimation regarding technical maturity and adoption in practice might be given. It can also be expressed whether or not the mechanism has a special focus on security, because in practice many mechanisms might also be used for security purposes, but do not directly focus on them. Several levels of usability can be stated indicating the experience users need in order to employ a mechanism.

The classes Method, Tool and Notation inherit all these properties from the class `Mechanism` and have their own characteristics defined by a set of specific attributes. For example, a Method has some general attributes, such as input, output and if it is model-driven or provides checklists for developers. A method can be supported by notations or tools; this is represented in the model with corresponding associations between the classes.

For a Notation, characteristics such as whether the notation is graphical, textual or table-based are considered.

The description of a Tool is given, among others, by the information of languages it is written in, operating systems it supports, frameworks it uses and licenses under which it is released.

The abstract class KnowledgeObject is adopted from the CBK [BEHS12, figure 2.2] as a super class for all elements which are described by SecEval's Context model. A `KnowledgeObject` has associated names, tags and related sources, which could be any kinds of sources, as publications or URLs. The UML association class `Ambiguity` allows to connect and to describe ambiguous knowledge objects that are not (yet) clearly separated in practice.

For convenience, we allow to group knowledge objects within Categories, which themselves can belong to knowledge objects. This is especially useful if instance diagrams get bigger.

Security features, such as confidentiality[8] or integrity[9] are represented by the class Security Property. Security properties can also be based on other security properties.

Security properties are always related to Assets. For example the security property "confidentiality" can be a feature of a transmitted data package. For us, an asset is something of value that has to be protected, as web servers, web applications or information like passwords. Assets can belong to or be used by other assets and some of them, as web servers, might be described by a certain configuration. Threats endanger at least one asset and general methods can secure assets or use them. E.g., cryptographic hashing secures passwords by not storing them directly and authentication uses passwords to identify users.

A Vulnerability is "a weakness that makes it possible for a threat to occur" [Bis02, p.498]. Thus, it endangers security properties. Examples are injection[10], buffer overflows, etc. The objective of certain methods is to detect vulnerabilities or to shield them from being exploited by a threat. Every vulnerability belongs to at least one location; details about the component the vulnerability is located in can be given using the attribute `affectedComponent`. Furthermore, the categorization scheme from OWASP Top 10 [OWA13] is included (which is adapted from the OWASP Risk Rating Methodology [OWA14d]) using prevalence, impact level, detectability and exploitability. Regarding the latter two roles, the `Difficulty` "theoretical" means that it is practically impossible to detect or exploit a vulnerability (cf. figure 1). For concrete vulnerabilities, the inherited attribute `name` can refer to an identifier of the Common

[8] Confidentiality "is the concealment of information or resources". [Bis02, p.4]
[9] (Data) integrity "refers to the trustworthiness of data or resources, and it is usually phrased in terms of preventing improper or unauthorized change". [Bis02, p.5]
[10] "Injection flaws occur when untrusted user data are sent to the web application as part of a command or query". [Pau13, p.9]

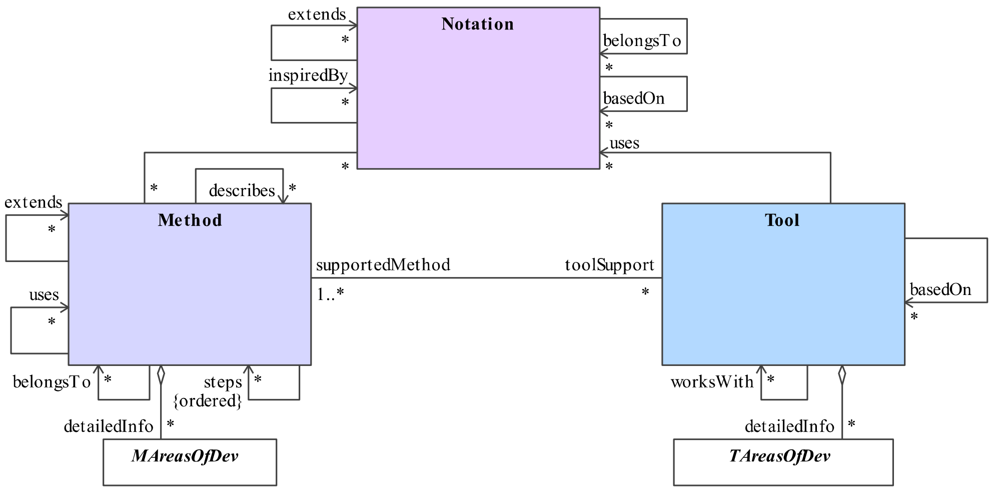


Fig. 2. Security Context: Further Associations

Vulnerabilities and Exposures List [The15]. SecEval uses the enumeration `Recon` to describe possible leakage of (possibly critical) data.

A Threat is “a potential violation of security” [Bis02, p.6]. It is treated as a kind of method which is vicious. At least one vulnerability has to be involved, otherwise a threat is not malicious (and the other way around), which is denoted by the multiplicity [1..*]. Additionally, threats can be mitigated by other methods.

For SecWAO, we primarily focus on the associations between classes. Figure 2 depicts that a notation can belong to another notation. For example Javadoc belongs to Java [Ora15] and extends it, whereas the Scala [O$^+$15] programming language is just inspired by Java (among others).

A tool can be based on other tools, which is the case when libraries are used or when plugins are written. The association `worksWith` denotes that tools work together; for example a tool might process the output of another tool that was executed before. The abstract classes `TAreasOfDev` and `MAreasOfDev` are depicted to indicate that further attributes, related to the development phases a tool or method is used in, can be specified with SecEval. The interested reader is referred to [BKW14b] for further details.

A method can describe other methods, as e.g., the OWASP Risk Rating Methodology describes the general method of rating risks. This methodology defines steps that can be represented as an ordered list. Besides, a method can extend other methods, which means it might change them. It is also possible that other methods are used without any changes. The role `belongsTo` usually involves using a method.

An advantage of our model is that it can easily be extended. If someone needs a more detailed approach, additional elements can be added, as discussed in further detail in [BKW14b].

4 SecWAO: Secure Web Applications' Ontology

In this section we present our approach by instantiating SecEval's Context model to establish an ontology for secure web applications. Although this paper focuses on web applications, SecEval can also be used for relating knowledge objects that are not in the scope of the internet. We introduce SecWAO by example, before we detail the relationships between web-related security properties, methods and vulnerabilities. Note that diagrams depict excerpts of the ontology by visualizing chosen perspectives.

4.1 Overview of SecWAO by example

In the following we illustrate relations of cross-site-scripting (XSS). XSS is a kind of injection that aims at adding malicious script code – usually JavaScript – to a website so that the browser executes the code. [She12, p.24] According to OWASP's Top 10, it is the third most risky web applications' vulnerability and the most widespread. [OWA13]

The upper part of figure 3 depicts the main classes of SecEval's Context model. Instances of these classes are shown in the lower part, around an instance from the class `Vulnerability` that is called `cross-site-scripting (XSS)`. In UML, an instance is denoted by an underlined object name, followed by a colon and the name of the class that it instantiates. In this section, the second part is hidden in the case that a legend provides distinctive features, as shades of color. We encourage reading the online version of this paper, as colors improve clarity.

In figure 3 SecWAO helps not only to express that XSS is a kind of injection but also that it is threatened by JavaScript code provided in a way to be executed. In practice, the name of threats and vulnerabilities are often used for both, e.g. the instance of `Threat` could also be named "cross-site-scripting (XSS)". There are subtypes of XSS: reflected XSS executes code in a browser that has (most of the time involuntarily) been sent by a user. Stored XSS is delivered to all browsers that visit a page that contains vulnerable content, which has been stored at the server since a successful attack. [Pau13, p.10] A vulnerability would be harmless if it does not jeopardize security properties. Weakening control flow integrity[11] can be especially harmful, because data security, as confidentiality of the user's data in the browser and the integrity of the data which is sent to the server are based on it. For example, JavaScript code injected into an online banking service might undermine the browser's Same-Origin-Policy[12] in order to secretly report the user's account balance to a third party. JavaScript code might also alter the amount of money after a user submitted a value for a bank transfer, which violates data integrity. In figure 3 we model general SecWAO assets as "user data" and "website in browser". However, assets can be finer grained in case developers want to extend SecWAO to model more concrete scenarios for their web application. General assets, which are not specific to web applications, can be found in [HSD07, figure 3].

[11] Control flow integrity is the property of software that restricts a user to execute functions in a predefined order, according to the program logic.
[12] The Same-Origin-Policy makes sure that "actors with differing origins are considered potentially hostile versus each other, and are isolated from each other" (`http://www.w3.org/TR/html5/browsers.html#origin`)

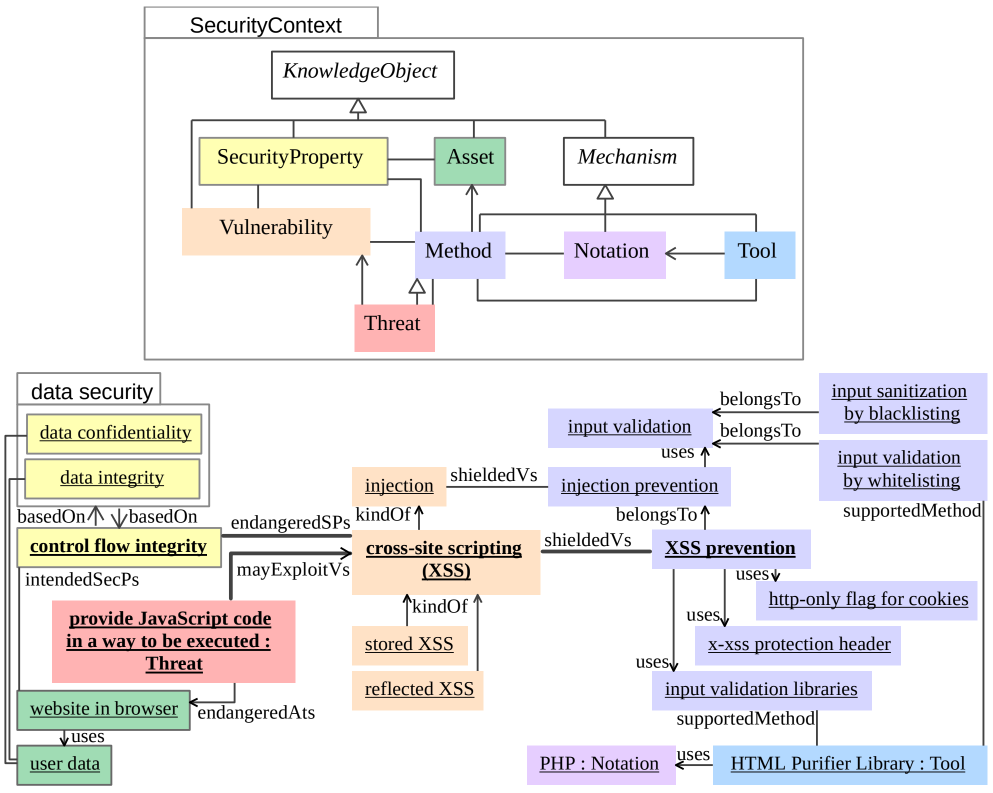


Fig. 3. Upper part: SecEval's Security Context model, serving as a legend for the lower part: SecWAO example of relations regarding cross-site-scripting (XSS).
(If printed without colors: instances of the class `Vulnerability` are located above and below "cross-site-scripting"; instances of the class `Method` are grouped around "XSS prevention".)

On the lower right of figure 3, methods to shield web applications from vulnerabilities like injection are depicted. In general, injection prevention uses input validation, preferably whitelisting, where allowed inputs are specified and different input is discarded. Other use cases require input sanitization by blacklisting, where developers struggle to filter all kinds of harmful inputs. [OWA15a]

For web applications, input validation libraries for XSS-blacklisting exist. Additionally, the http-only flag for cookies[13] can be set and the x-xss protection header can advise browsers to turn on built-in XSS protection [Tia14]. With instances of SecEval's Context model we can also specify that the HTML Purifier Library [Y^+15] is an input validation library written in PHP [PHP14].

4.2 Security Properties

Security requirements consist of at least two elements: the asset that should be *secured* and the kind of security that "*secured*" refers to. The latter can be expressed as an instance of the class `SecurityProperty` from SecEval's Context model.

[13] "The HttpOnly attribute limits the scope of the cookie to HTTP requests. In particular, the attribute instructs the user agent to omit the cookie when providing access to cookies via 'non-HTTP' APIs (such as a web browser API that exposes cookies to scripts)." [A. 11]

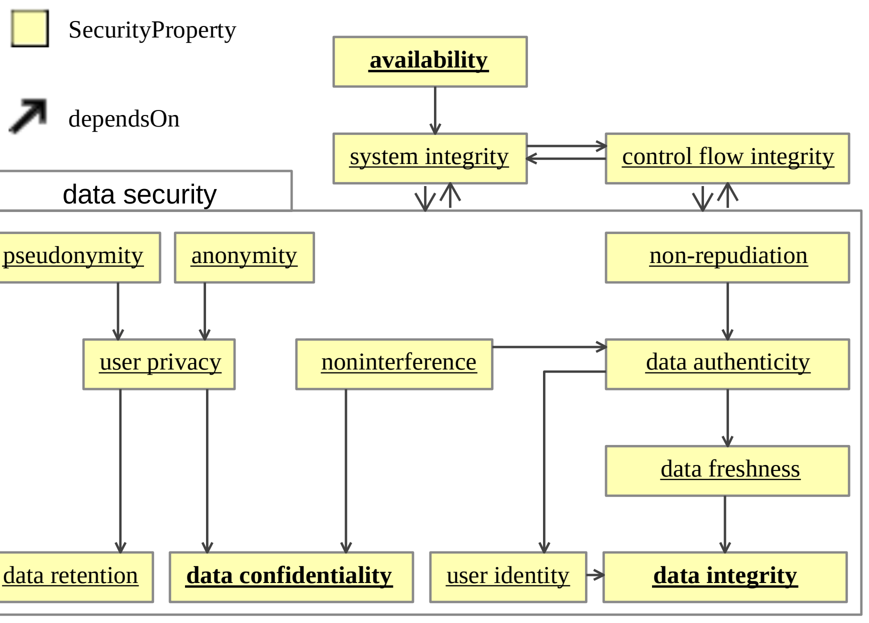


Fig. 4. SecWAO Security Properties: Overview

Figure 4 depicts common security properties and their interrelations, using instances of the association with the role `dependsOn` that belongs to the class `Security Property` (cf. figure 1). For us, "common" means that they are mentioned in Wikipedia[14] and in several scientific publications, which was verified with Google Scholar[15]. Please note that we provide informal definitions for security properties, as the first goal of SecWAO is to quickly get an overview of knowledge objects and their relations to each other. This is why we also explain concepts in plain terms and do not refrain from citing Wikipedia in case it provides comprehensible definitions. In this section, a "user" denotes a human user as well as automated users controlled by a computer system.

Availability[16] of a functionality depends on system integrity[17], because if e.g., an attacker has taken over a system, system integrity is not necessarily provided as the system might be shut down or destroyed to the attacker's liking. In case an attacker manages to violate control flow integrity, e.g., by using URLs that directly reference program functions, system integrity can be at stake. Vice versa, violating the control flow integrity of a program is possible if attackers control a system, as they can alter program code or the configuration of a web application firewall.

It is not surprising that data security (depicted as UML package in figure 4)

[14] Wikipedia. `https://www.wikipedia.org` (English or German)
[15] Google Scholar. `https://scholar.google.com`
[16] Availability "refers to the ability to use the information or resource desired". [Bis02, p.6]
[17] System integrity is a "condition of a system wherein its mandated operational and technical parameters are within the prescribed limits". (Wikipedia: System integrity)

depends on system integrity and control flow integrity and vice versa, especially as insecure data like disclosed configuration data might provide expedient information for attacks. Note that methods can shield from related vulnerabilities, as e.g., a JavaScript sandbox in a browser can prevent malicious code from endangering system integrity.

Data security refers to different security properties; the most common are DATA CONFIDENTIALITY and DATA INTEGRITY (depicted at the bottom of figure 4). When methods to ensure confidentiality and integrity are not implemented properly enough, all security properties that are based on them can be endangered. For example, disclosing confidential home addresses leads to a violation of user privacy[18] as well as anonymity or pseudonymity, in case a web application allows users to interact with a service without a (unique) name or by just using a nickname (cf. [PP06, p.614f]). Besides depending on confidentiality, privacy also depends on data retention[19]. Confidentiality, integrity and availability are often referred to as "CIA Triad", although many other security features are required and realized these days.

Noninterference[20] is also based on confidentiality, as it must be possible to restrict read or write access to classified data. Additionally, it depends on data authenticity[21], which requires data freshness[22], data integrity and user identity[23].

Non-repudiation[24] needs data authenticity and is transitively based on the security properties on which data authenticity depends. The term "traceability" is sometimes used for weaker forms of non-repudiation like logging. [BSS11, p.12]

In the main ontology of [KLK07], a list of security objectives is provided[25] that also has more elements than the CIA Triad. Our security properties share some of the concepts, although we differentiate between methods that help to reach security goals. For example, in SecWAO "replay prevention" is a method that helps to realize the security property of data freshness (cf.figure 7).

Security properties are closely related to assets they characterize. In figure 5, general data security properties such as data confidentiality and data authenticity are presented in the contexts of three different assets – corresponding to the boxes in the figure.

In the context of protocols for agreeing on a key for future communication (key exchange), data confidentiality may depend on forward anonymity and forward secrecy. The former means that recorded data traffic and compromised (long-term) keys

[18] Privacy "is the right to control who knows certain aspects about you, your communications, and your activities". [PP06, p.604]
[19] Data retention defines which information is stored and how long it will be kept.
[20] Noninterference "is a property that restricts the information flow through a system". [vTJ11, p.605] This usually means that information and users are grouped into categories with different levels of security and information from one level can only affect information of other levels according to policies.
[21] Data authenticity defines that received data was send from users that are who they claim to be.
[22] Data freshness is given if data is up-to-date (and not a replay of data that was sent in the past).
[23] (User) identity "is a set of information that distinguishes a specific entity from every other within a particular environment." [vTJ11, p.584]
[24] Non-repudiation "refers to an inability to disavow a previous agreement". [vTJ11, p.852] Simplified: a user cannot deny to have sent or received a message at a given time with a given content.
[25] Security goals mentioned in [KLK07] are confidentiality, availability, user authentication, message authentication, authorization, message integrity, key management, trust, host trust, replay prevention, covert channel prevention, separation, traffic hiding and anonymity.

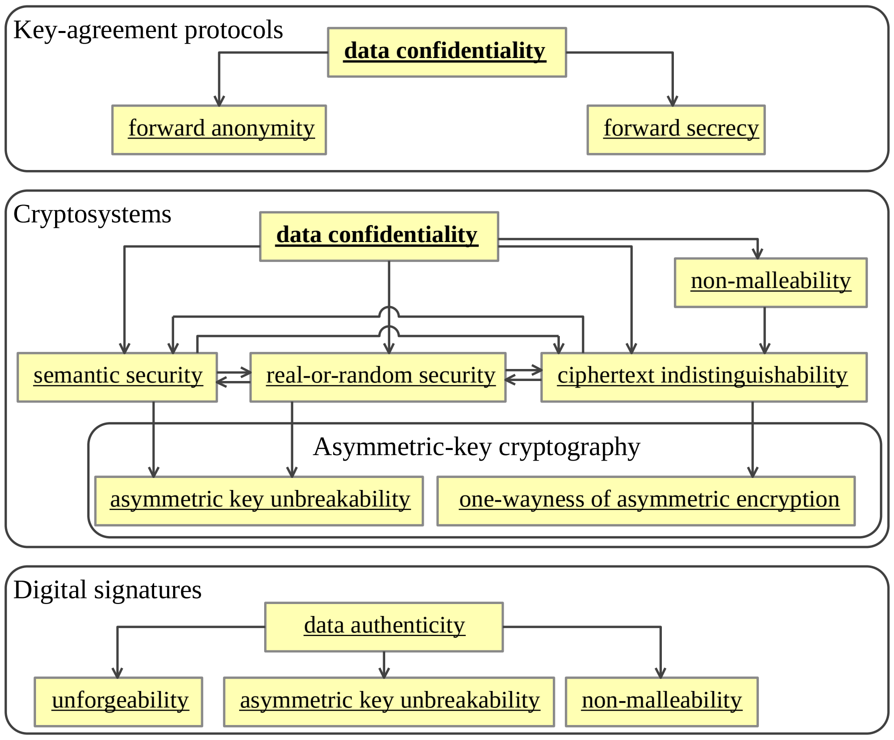


Fig. 5. SecWAO Security Properties: Details

do not disclose the identities of the communication partners. The latter means that "disclosure of long-term secret keying material does not compromise the secrecy of the exchanged keys from earlier runs". [DVOW92, p.7] In other words: session keys will not be revealed, even if all data traffic was recorded and long-term keys have been compromised after the sessions took place.

Confidentiality depends on properties of cryptosystems, as described in a Wikipedia article[26]. According to this article, semantic security[27], real-or-random security[28] and ciphertext indistinguishability[29] are equivalent and if they are broken, non-malleability[30] is also broken. Besides, typical security features that are assumed to hold for asymmetric-key cryptography are key unbreakability[31] and one-wayness[32].

Security properties for digital signatures are depicted at the bottom of figure 5:

[26] German Wikipedia: Sicherheitseigenschaften kryptografischer Verfahren
[27] Semantic security means that attackers can derive nothing more than the length of an encrypted message.[26]
[28] Real-or-random security means that attackers cannot distinguish two encrypted messages, even if one of it encrypts a plaintext they provided.[26]
[29] Ciphertext indistinguishability means that attackers cannot distinguish pairs of cyphertexts, even if they know their plaintexts.[26]
[30] Non-malleability means that "given the ciphertext it is impossible to generate a different ciphertext so that the respective plaintexts are related". [DDN03]
[31] Unbreakability means that attackers cannot calculate the private key from the public key.[26]
[32] One-wayness means that attackers cannot encrypt a given ciphertext.[26]

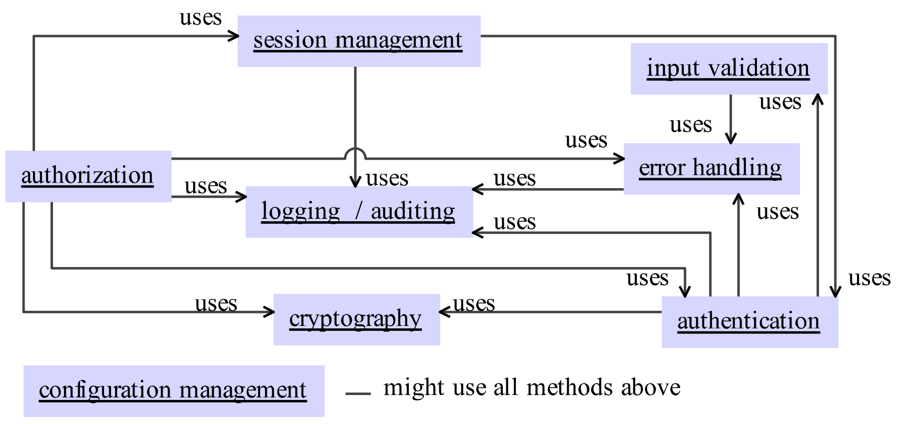


Fig. 6. SecWAO: Main Methods

Strong unforgeability[33] is based on existential unforgeability[34]. Asymmetric key unbreakability and non-malleability can also apply to digital signatures. In this context, non-malleability is referred to as not being able to create a second valid signature for a pair of a message and its valid signature.[26]

Other security features that are modeled with SecWAO, but not presented in this paper are e.g., collision resistance, which is a property of cryptographic hash functions or "provable security", which means that security properties of an asset are mathematically proven. Herzog et al. [HSD07] specify goals that contain trust and correctness, which we omitted due to the vagueness of these terms.

4.3 Methods

Methods help to establish security properties for an asset. The main methods used in SecWAO are depicted in figure 6. Input validation has to implement error handling for coping with illegal inputs, which is represented in the diagram by instantiating the `Method`'s association with the role `uses` (cf. figure 2). Errors as well as success messages (e.g., a successful authentication, i.e., a user login) or status messages (e.g., an expired user session) can be logged. Authorization is a synonym for access control [vTJ11, p.2] and relies on successful authentication to identify users that request access, e.g., to an internal web page. The management of user sessions is also based on authentication, although anonymous sessions are possible that only require to identify, e.g., a cookie instead of authenticating a user. Cryptography aims at protecting "a secret from adversaries, interceptors, intruders, interlopers, eavesdroppers, or simply attackers, opponents, and enemies" [vTJ11, p.283]. As the aim of authorization is similar, it often relies on cryptographic methods to enforce access control. For authentication cryptography is frequently employed to ensure confidentiality, integrity and freshness of requests in authentication protocols.

All tools that support the methods mentioned so far typically need to be configured. Otherwise, an authentication service might grant access to all users or an error

[33] Strong unforgeability "ensures the adversary cannot even produce a new signature for a previously signed message". [BSW06]
[34] Existential unforgeability means that "an adversary who is given a signature for a few messages of his choice should not be able to produce a signature for a new message". [BSW06]

message might disclose critical confidential information about a system. Configuration management is a method that relates to secure employment of methods.

In this paper, we focus on technical methods, although we think that non-technical secure development methods are equally important. Examples are security aware management (granting time and money for more secure implementations) and a security-aware software development life cycle, including e.g., security requirements elicitation, secure design methods, code reviews and penetration testing. Existing "applied security" literature typically includes a set of security principles[35] like simplicity, open design, minimum exposure, secure-by-default, fail securely etc. [BSS11] For example, secure-by-default and fail securely belongs to the methods "configuration management" and "error handling" in SecWAO.

The upper half of figure 7 details how CRYPTOGRAPHY relates to other methods. We use bold text in the object diagrams to identify methods that we think are central to recognize a certain set of methods, as e.g., methods related to cryptography. This set of methods support data integrity, authenticity, confidentiality and freshness, as can be seen at one glance in figure 7 (cf. instances of the class `SecurityProperty`). However we just exemplarily depict links to security properties and assets, as we focus on methods and their interrelations.

A cryptosystem[36] can be symmetric, asymmetric or hybrid, which is determined by whether a single key is used for decryption and encryption, or a public and a private key are used, or an asymmetric key is used for encrypting a generated symmetric key that encrypts a message. Especially for asymmetric keys, key management is important, as such keys are often reused many times in contrast to symmetric ones.

Other methods that belong to cryptography are the mechanisms of random number generators and cryptographic hash functions (i.e., one-way hash functions). Random number generators can be implemented in hardware or software. Software implementations are so-called "secure pseudo-random number generators": algorithms that produce random numbers in a sequence that is determined by a seed. The seed has to be given as an input for the algorithm [vTJ11, p.995]. Generators are e.g., used for digital signatures that enforce data authenticity.

INPUT VALIDATION – as known from the example of section 4.1 – is depicted on the bottom of figure 7. Besides XSS prevention, e.g., supported by the Content Security Policy[37], other types of injection prevention exist. For example, database query injection (referred to as "SQL injection") can be avoided by using libraries for prepared statements. Prepared statements are available in most programming languages and they distinguish between user input[38] and SQL statements so that the

[35] For a broad overview of practical security principles, the interested reader is referred to `https://www.owasp.org/index.php/Category:Principle`.

[36] A cryptosystem "is a system consisting of an encryption algorithm, a decryption algorithm, and a well-defined triple of text spaces: plaintexts, ciphertexts, and keytexts". [vTJ11, p.284]

[37] Content Security Policy is "a mechanism web applications can use to mitigate a broad class of content injection vulnerabilities, such as cross-site-scripting. The server delivers the policy to the user agent via an HTTP response header or an HTML meta element". [W3C14]

[38] "User input" refers to any input sent by users or their devices, including input the user entered in a text field, cookies or protocol headers sent by a browser, etc.

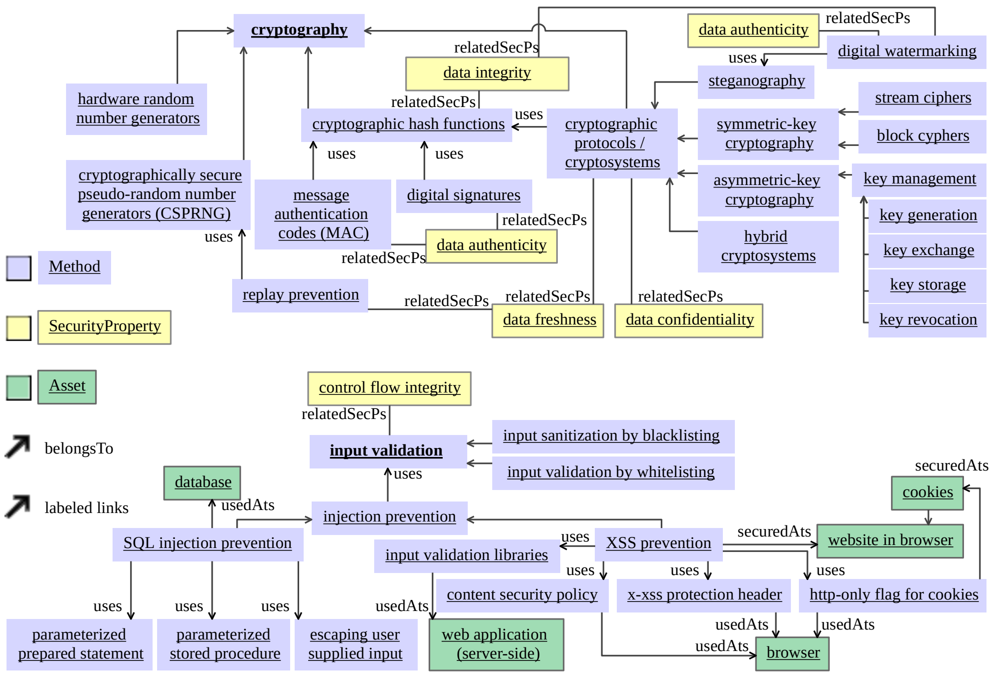


Fig. 7. SecWAO Methods: Cryptography and Input Validation

former cannot influence the latter. Other options, as stored procedures[39] or escaping all user supplied input according to the database syntax that is used, are considered less secure, as a single careless mistake can bear a severe security flaw. [OWA15c]

Figure 8 focuses on AUTHENTICATION, including typical methods that are used for web applications, as registering for an account using common types of registration, ways to securely recover credentials and typical logout mechanisms. Authentication itself is often used as a synonym for checking the validity of the user's identification, which we can express by adding a link with the role `ambiguity` that belongs to `KnowledgeObject` (cf. figure 1). A less well-known method is to check if the user enters a password for accessing the application in a so-called "panic mode". This mode allows users from unsafe regions around the world to give away a valid password if threatened. This password permits an attacker to sign in to a web application that does not give rise to suspicion, while hiding personal user data and restricting access to critical functionality. That is why the panic mode is also related to state-based access control (cf. figure 9). [OWA15d]

The upper part of figure 8 shows knowledge objects related to SESSION MANAGEMENT. On the upper left, common methods like starting or ending a session are depicted. In the case a critical error occurred during a session, it might be advisable to end the session. Note that sessions also exist for unauthenticated users; however the session identifier (ID) should be changed after authentication to avoid session fix-

[39] Stored procedures are located in the database and can be called from a client using parameters, which shifts the problem of securing SQL statements from the application to the stored procedure.

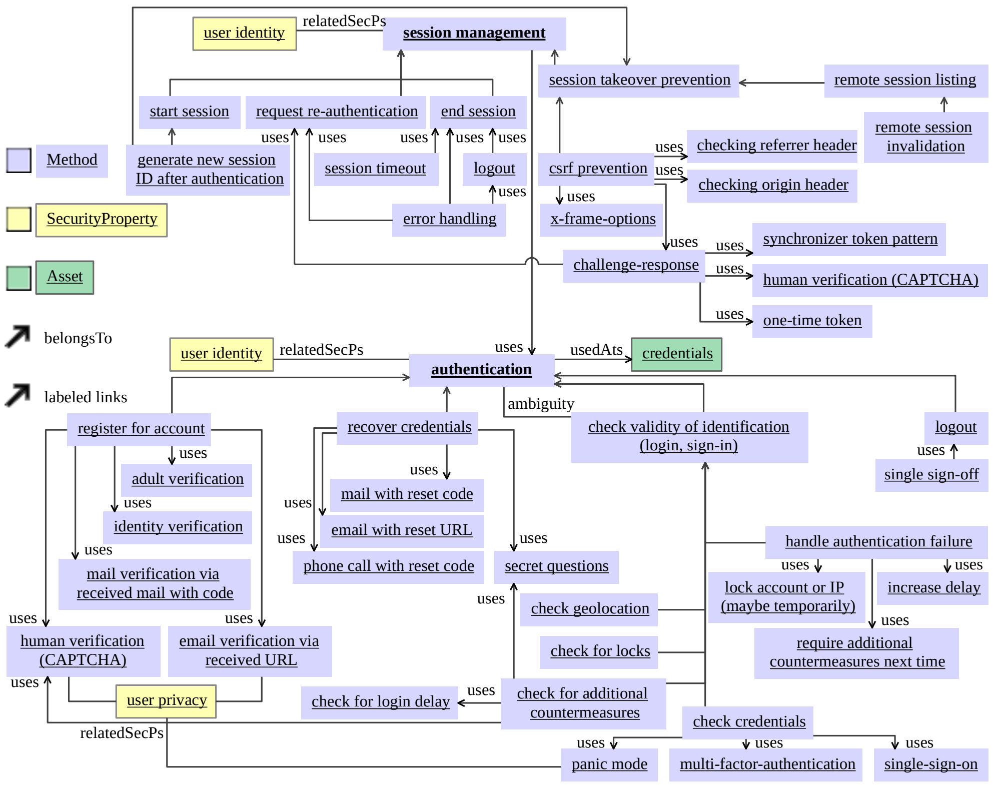


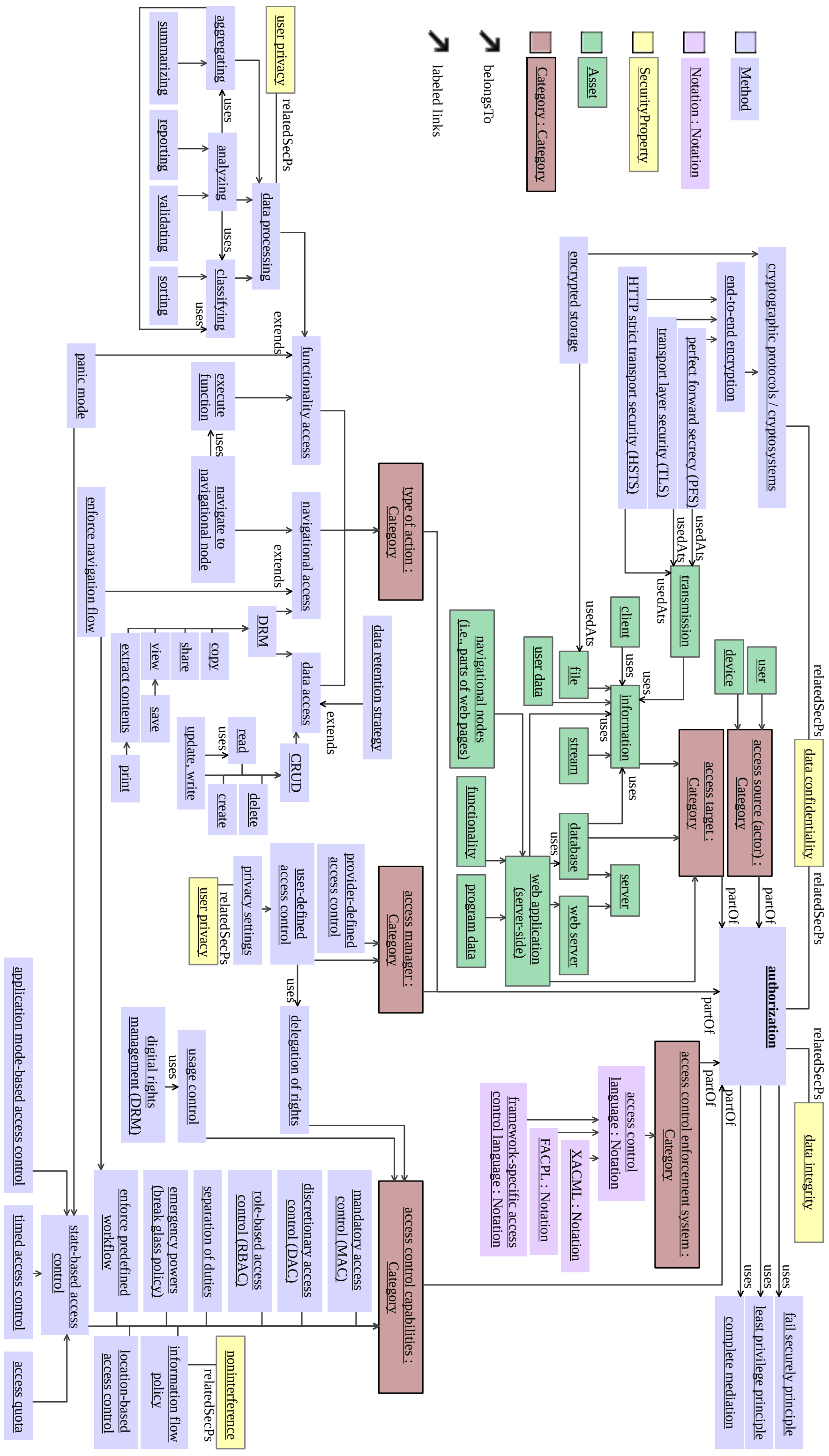
Fig. 8. SecWAO Methods: Authentication and Session Management

ation[40]. Other methods to prevent session takeover are depicted on the right: It can be helpful to enable users to list their active sessions so that they can invalidate them in case a device was stolen. Methods to prevent cross-site-request-forgery (CSRF)[41] are modeled according to their description in [OWA14b].

Authorization defines an access source, an access target and actions that are permitted to be executed. In addition, figure 9 depicts an access control enforcement system that decides whether or not to permit a request for access, according to access control policies, which can be noted in languages as XACML [OAS05]. These policies can be defined by a provider or by users, as expressed by the role `belongsTo` that is played by the category `access manager`. `Access control capabilities` describe common approaches used for authorization, as role-based access control (the user belongs to roles and access is specified for roles) or state-based access control (access control policies can, e.g., refer to the current time or the mode an application is in, like maintenance mode).

[40] Session fixation exploits cases where session IDs are not changed after the login, as an attacker can access a website to obtain an ID, trick a user to access and sign in to the same website using this ID (e.g. provided by a URL within an email) and can continue using the – now authenticated – session. [OWA15b]

[41] A CSRF attack "causes a users web browser to perform an unwanted action on a trusted site for which the user is currently authenticated" [OWA14b]. For example a user clicks on a link within a junk mail that uses an active session of an online shop to buy an unwanted product.


Method
Notation : Notation
SecurityProperty
Asset
Category : Category
belongsTo
labeled links
authorization
access source (actor) : Category
access target : Category
access manager : Category
access control enforcement system : Category
access control capabilities : Category
type of action : Category
user
device
information
file
stream
database
server
web server
client
transmission
user data
navigational nodes (i.e., parts of web pages)
functionality
program data
web application (server-side)
cryptographic protocols / cryptosystems
end-to-end encryption
perfect forward secrecy (PFS)
transport layer security (TLS)
HTTP strict transport security (HSTS)
encrypted storage
data confidentiality
data integrity
fail securely principle
least privilege principle
complete mediation
access control language : Notation
XACML : Notation
FACPL : Notation
framework-specific access control language : Notation
data retention strategy
provider-defined access control
user-defined access control
privacy settings
user privacy
delegation of rights
usage control
digital rights management (DRM)
mandatory access control (MAC)
discretionary access control (DAC)
role-based access control (RBAC)
separation of duties
emergency powers (break glass policy)
enforce predefined workflow
information flow policy
noninterference
location-based access control
state-based access control
application mode-based access control
timed access control
access quota
functionality access
navigational access
data access
execute function
navigate to navigational node
panic mode
enforce navigation flow
DRM
copy
share
view
save
extract contents
print
CRUD
read
update, write
create
delete
data processing
aggregating
analyzing
classifying
summarizing
reporting
validating
sorting
uses
usedAts
partOf
relatedSecPs
extends
Fig. 9. SecWAO Methods: Authorization

For web applications, common usages for cryptosystems are secure connections between a browser and a server to transmit confidential information. The protocol TLS [T. 08] with its option for Perfect Forward Secrecy (cf. security property forward secrecy in section 4.2) and the browser policy HSTS[42] are methods that are used in practice.

Besides, in figure 9 we use SecWAO to clarify the difference between `types of action`: functions of an application can be executed and data can be accessed. An example is DRM (Digital Rights Management), which aims at restricting copying, viewing or extracting information from files or videos. Data access can also be restricted by defining policies for CRUD (create, read, update, delete) on objects, as e.g., database records. Additionally, web applications focus on restricting navigational access. This allows to specify whether a user is allowed to navigate to a so-called "navigational node", i.e. a (part of) a web page. This avoids dead-ends, as it avoids navigating to a function a user is not allowed to access.

In figure 10 LOGGING, ERROR HANDLING and CONFIGURATION MANAGEMENT are presented. They closely relate to system integrity and are used by many other methods (cf. figure 6). In the context of web applications, system integrity is also important for the client. Consequently, it is common courtesy to provide users with possibilities for download verification, especially when downloading programs. Digital signatures or cryptographic hashes can verify a download in case it was not transmitted by a protocol, as e.g., TLS that supports integrity.

Non-repudiation, beyond re-authentication before executing critical actions, has not commonly been realized for typical web applications so far. Digital signatures and logging (ideally by a third party) could ease traceability and forensics for critical actions like purchases, or changes in the configuration of safety-critical appliances that provide a web interface.

Monitoring uses logging and log analysis for intrusion detection as well as for intrusion prevention. Log sanitization means that sensitive information is removed[43] from log messages. This can be important for information that has to be deleted after a certain period of time due to legal regulations and for debugging by developers who are not allowed to access concrete data sets of the production system. For error handling, a sanitization method should be applied to error messages in order to keep details of the system and the algorithm internal. Consequently, an internal error message should be replaced by a short message, which does not confuse normal users and does not allow attackers to gain any insights. Additionally, the instance **`fail securely principle`** reminds developers to carefully consider error states as regular part of their program, as web applications should be constantly available – meaning that restarts to recover a consistent state are undesirable.

Regarding configuration management, we provide some examples in figure 10, as server management and browser configuration. Browsers can be configured by users, but their behavior can also be influenced by web servers' responses, as already introduced above, e.g., by using a content security policy and other http headers [OWA14c].

[42] HSTS enables "web sites to declare themselves accessible only via secure connections" [J. 12], which prevents a man-in-the-middle from hijacking unencryped requests (http) users sent to servers that usually redirect from unencrypted pages to an encrypted pages (https) *after* this first request.
[43] Wikipedia: Sanitization (classified information)

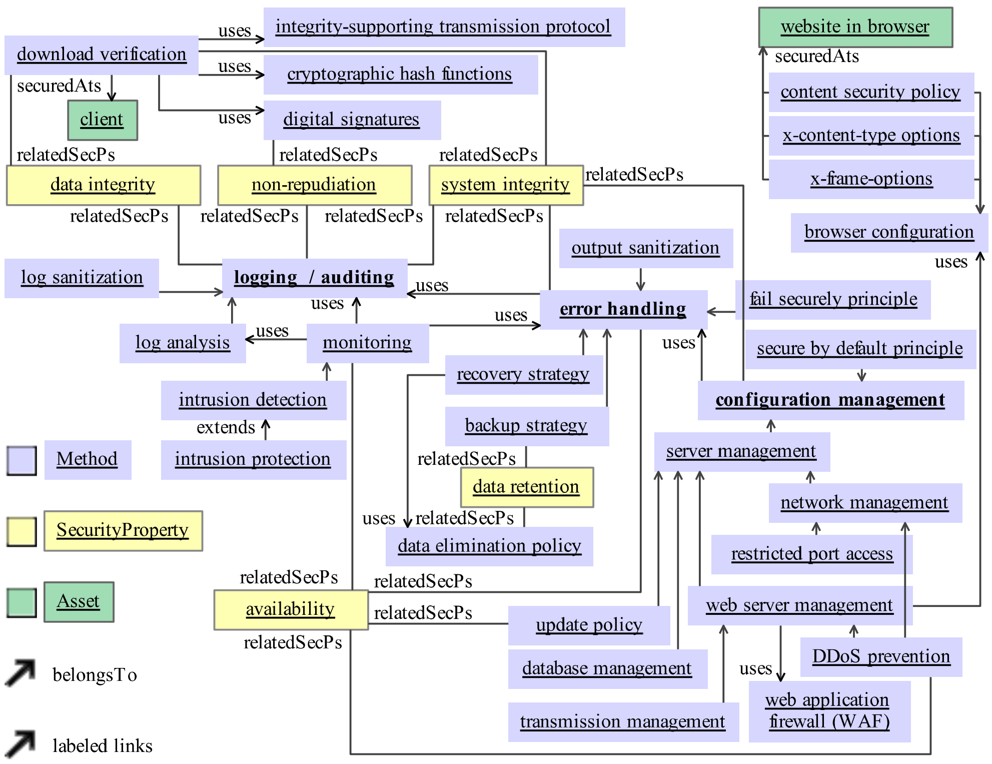


Fig. 10. SecWAO Methods: Logging, Error Handling and Configuration Management

In general, objects of SecWAO can be grouped according to different aspects. For example methods could also be grouped according to a certain security property like data integrity, to learn about methods that are related to it. Purists might as well query the tree that contains all UML elements.

4.4 Vulnerabilities and Threats

Figure 11 depicts the ten top vulnerabilities of web applications (according to OWASP [OWA13]) and relates them with major threats that may exploit these vulnerabilities. The diagram is roughly grouped according to the main methods we used in the previous subsection. For example the instance of the class `Vulnerability` that is named `unvalidated input` corresponds to the method's instance `input validation`, depicted in figure 7. These correspondences can be modeled using associations from figure 1 (i.e. with the roles `detectedVs` or `shieldedVs`), as presented in figure 3.

Besides the vulnerabilities from the OWASP Top 10, the diagram in figure 11 depicts several related vulnerabilities ranging from general vulnerabilities like error-prone memory management or insecure credentials[44] to web-specific vulnerabilities

[44] Insecure credentials are, e.g., passwords that are easily guessable. This can mean that they are either too short so that a brute-force attack is possible, or too common so that a rainbow-table attack is effective. Both exploit that brute-force credential guessing is possible, either on stolen password hashes (in case they are not hashed with an up-to-date cryptographic hash function and salted) or on web applications that apply no means to restrict credential guessing attempts. (cf. lower right of figure 11)

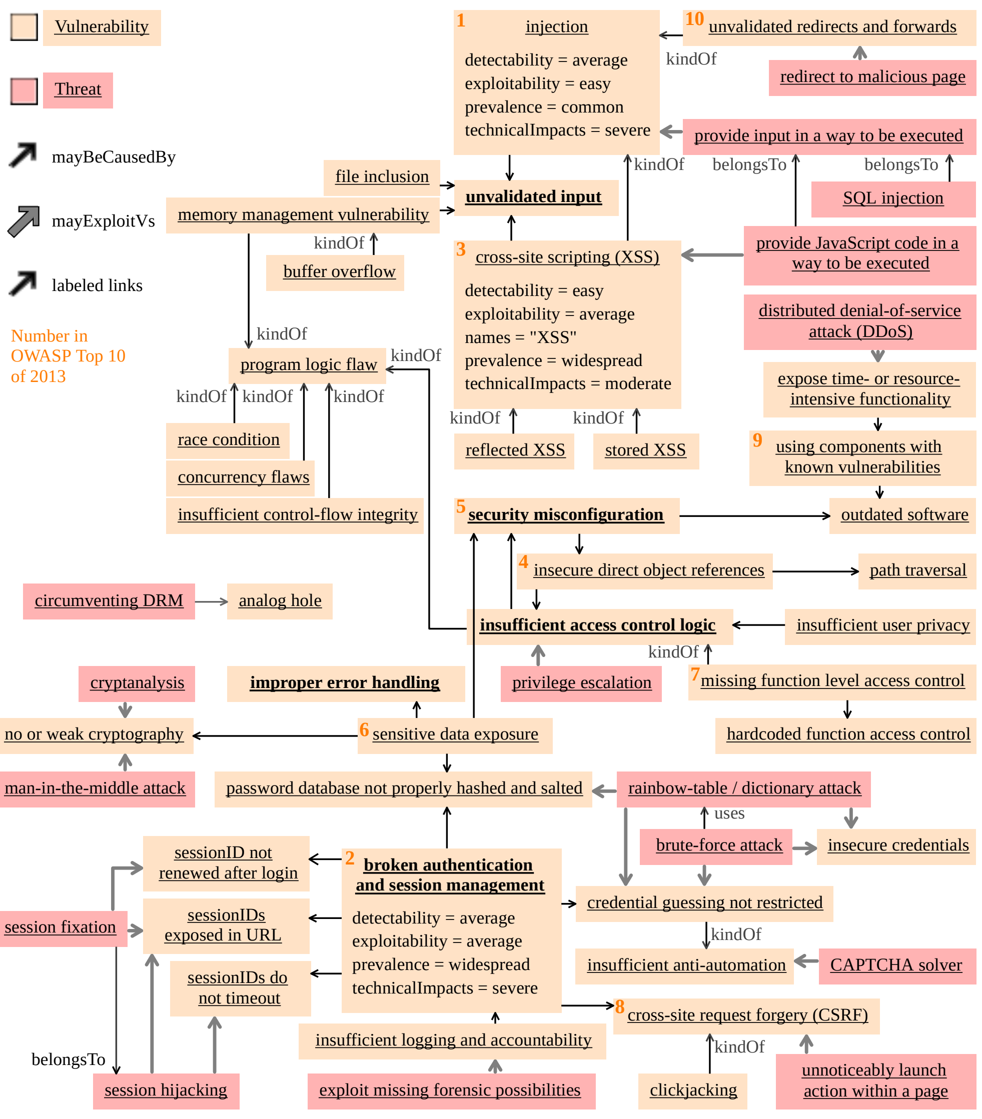

Vulnerability
Threat
mayBeCausedBy
mayExploitVs
labeled links
Number in OWASP Top 10 of 2013
1 injection
detectability = average
exploitability = easy
prevalence = common
technicalImpacts = severe
10 unvalidated redirects and forwards
kindOf
redirect to malicious page
provide input in a way to be executed
belongsTo
SQL injection
file inclusion
memory management vulnerability
unvalidated input
buffer overflow
3 cross-site scripting (XSS)
detectability = easy
exploitability = average
names = "XSS"
prevalence = widespread
technicalImpacts = moderate
provide JavaScript code in a way to be executed
distributed denial-of-service attack (DDoS)
expose time- or resource-intensive functionality
9 using components with known vulnerabilities
program logic flaw
race condition
concurrency flaws
insufficient control-flow integrity
reflected XSS
stored XSS
5 security misconfiguration
outdated software
4 insecure direct object references
path traversal
circumventing DRM
analog hole
insufficient access control logic
insufficient user privacy
privilege escalation
7 missing function level access control
hardcoded function access control
cryptanalysis
improper error handling
no or weak cryptography
6 sensitive data exposure
man-in-the-middle attack
password database not properly hashed and salted
rainbow-table / dictionary attack
uses
brute-force attack
insecure credentials
sessionID not renewed after login
2 broken authentication and session management
detectability = average
exploitability = average
prevalence = widespread
technicalImpacts = severe
credential guessing not restricted
session fixation
sessionIDs exposed in URL
sessionIDs do not timeout
insufficient anti-automation
CAPTCHA solver
8 cross-site request forgery (CSRF)
insufficient logging and accountability
belongsTo
session hijacking
exploit missing forensic possibilities
clickjacking
unnoticeably launch action within a page

Fig. 11. SecWAO Vulnerabilities

like clickjacking[45] or cross-site-scripting. Depending on the configuration of a concrete asset, some vulnerabilities, like `weak credentials`, can only be exploited in combination with others.

For us, attacks are threats that become reality for a concrete asset. In practice, threats and attacks often share the same name, as e.g., the *threat* of a brute-force *attack*. In [HSD07, figure 4], a threat classification is provided that uses the terms

[45] A web page is vulnerable to clickjacking if an attacker can hide it by layers with arbitrary contents to trick users into clicking on these layers and thus involuntarily interact with the hidden web page. [OWA14a]

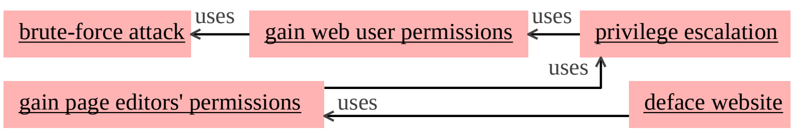


Fig. 12. A sequence of threats

'threat' and 'attack' interchangeably. The authors of [FE09] differentiate between "low level threats" and "top level threats". The former correspond to our definition of threats, the latter is the counterpart to what we express positively as security property (e.g., data disclosure – confidentiality).

Successful attacks can give rise to further vulnerabilities, as an attacker can use a found out password to search for vulnerabilities that are not exploitable from the outside. As the class `Threat` descends from `Method`, the roles `uses`, `steps`, `extends`, etc., can be used. Figure 12 shows an example of a sequence of threats that could become reality so that in the end a set of web pages are defaced. In this way, attack trees [Sch99][46] can be constructed. Such trees can grow exponentially[47], as threats depend on assets: storing illegal data might be a huge threat for servers with plenty of storage, whereas misconfiguring nuclear power plants might be less relevant for average server administrators.

With instances of SecEval's Context model, we can also represent relations of methods and notations. For example, we modeled an overview of current web application security testing tools. The interested reader can download[48] all models in the MagicDraw [No 15] or XMI [Obj05] format.

5 Conclusion

In this paper we have presented a novel ontology, namely SecWAO, for the area of secure web applications. The aim is to make web system developers aware of security issues and give hints how to secure their systems. To our knowledge, SecWAO is the first ontology that provides a systematic overview of the domain of secure web applications. It is based on the general Security Context model of SecEval that defines generic concepts like security property, threat, vulnerability, and asset and relates them with mechanisms like method, notation, or tool. In SecWAO we instantiate these elements, e.g., cryptography, input validation, authentication, session management, authorization, logging, error handling, and configuration management are some instances of security-related methods.

In [NVB+13], Neuhaus et al. state that the first question to be asked when dealing with an ontology is: "Can humans understand the ontology correctly?". According to our experience, SecWAO reaches this goal due to its clear structure and its common terminology. So far, we have used SecWAO in two ways: First, we structured a tutorial about practical IT-Security for master students according to SecWAO in the

[46] Although the term "attack tree" is widely used, the term "threat tree" would be more appropriate, as there is no need for attackers to exploit all theoretical possibilities.
[47] The growth of the ontology is a reason why SecWAO will never be completely finished; aside from the fact that the state of the art changes over time.
[48] SecEval and SecWAO. `http://www.pst.ifi.lmu.de/~busch/SecEval/`

winter term 2014/2015. Zooming into SecWAO's UML diagrams on the slides allowed giving further explanations without losing track of the context. The students acknowledged the helpfulness of SecWAO for studying, although they noted that readability naturally depends on a projector with high resolution and a large projection surface.

Second, we integrated SecWAO into the UML-based Web Engineering approach (UWE) [LMU15]. The UWE notation is defined using the UML profile mechanism and SecWAO allowed us to systematically establish UML stereotypes and tags for modeling secure web applications. As a result, UWE enables developers to document most important security design decisions in a graphical way. We expect that this structured and concise documentation facilitates the development and maintenance of web applications, in particular in the case of changes in the developer team.

By using SecWAO combined with UWE in future work, we aim at making security issues to first class citizens of web development and at supporting a secure web development life cycle where security issues are considered from the beginning and in all development phases: during requirements elicitation assets are determined and relevant security properties are chosen; at design time appropriate methods for ensuring the required security properties are selected, whereas other methods notations and tools are chosen in all phases; e.g., in the testing phase security engineers typically decide which tools are used for penetration testing. Finally, security of web applications is a dynamic field; every year there are new threats, new vulnerabilities and new security methods. In order to remain useful, SecWAO will have to relate these new aspects and it would also be worthwhile to integrate SecWAO into one of the general security ontologies. In the future, it may also be interesting to compare previous versions of SecWAO to analyze how the domain of web applications evolves over time.

References

[A. 11] A. Barth. HTTP State Management Mechanism. Version 1.2. Specification, Internet Engineering Task Force (IETF), 2011. URL: `https://tools.ietf.org/html/rfc6797`.

[BBB+85] F. L. Bauer, R. Berghammer, M. Broy, W. Dosch, F. Geiselbrechtinger, R. Gnatz, E. Hangel, W. Hesse, B. Krieg-Brückner, A. Laut, T. Matzner, B. Möller, F. Nickl, H. Partsch, P. Pepper, K. Samelson, M. Wirsing, and H. Wössner. *The Munich Project CIP, Volume I: The Wide Spectrum Language CIP-L*, volume 183 of *LNCS*. Springer, 1985. `doi:10.1007/3-540-15187-7`.

[BBD+81] F. L. Bauer, M. Broy, W. Dosch, R. Gnatz, B. Krieg-Brückner, A. Laut, M. Luckmann, T. Matzner, B. Möller, H. Partsch, P. Pepper, K. Samelson, R. Steinbrüggen, M. Wirsing, and H. Wössner. Programming in a wide spectrum language: A collection of examples. *Science of Computer Programming*, 1(1-2):73–114, 1981. `doi:10.1016/0167-6423(81)90006-X`.

[BEHS12] K. Beckers, S. Eicker, M. Heisel, and W. Schwittek. NESSoS Deliverable D5.2 – Identification of Research Gaps in the Common Body of Knowledge, 2012. URL: `http://www.nessos-project.eu/media/deliverables/y2/NESSoS-D5.2.pdf`.

[Bis02] M. Bishop. *Computer Security: Art and Science*. Addison-Wesley Professional, 1st edition, 2002.

[BJ95] M. Broy and S. Jähnichen, editors. *KORSO - Methods, Languages, and Tools for the Construction of Correct Software*, volume 1009 of *LNCS*. Springer, 1995.

[BK80] M. Broy and B. Krieg-Brückner. Derivation of invariant assertions during program development by transformation. *ACM Transactions on Programming Languages and Systems*, 2(3):321–337, 1980. `doi:10.1145/357103.357108`.

[BKW14a] M. Busch, N. Koch, and M. Wirsing. SecEval: An Evaluation Framework for Engineering Secure Systems. In *Proceedings of Modellierung*, volume P-225 of *LNI*, pages 337–352, 2014.

[BKW14b] M. Busch, N. Koch, and M. Wirsing. Systematic Evaluation of Engineering Approaches for Secure Software and Systems. In M. Heisel, W. Joosen, J. Lopez, and F. Martinelli, editors, *Advances in Engineering Secure Future Internet Services and Systems*, number 8431 in LNCS State-of-the-Art-Surveys, pages 234–265. Springer, 2014. `doi:10.1007/978-3-319-07452-8_10`.

[BLVG+08] C. Blanco, J. Lasheras, R. Valencia-Garcia, E. Fernandez-Medina, A. Toval, and M. Piattini. A systematic review and comparison of security ontologies. In *Third International Conference on Availability, Reliability and Security, 2008. ARES 08*, pages 813–820, 2008. `doi:10.1109/ARES.2008.33`.

[BMPW86] M. Broy, B. Möller, P. Pepper, and M. Wirsing. Algebraic implementations preserve program correctness. *Science of Computer Programming*, 7(1):35–53, 1986. `doi:10.1016/0167-6423(86)90004-3`.

[BSS11] D. Basin, P. Schaller, and M. Schläpfer. *Applied Information Security: A Hands-on Approach*. Springer, 2011. `doi:10.1007/978-3-642-24474-2`.

[BSW06] D. Boneh, E. Shen, and B. Waters. Strongly unforgeable signatures based on computational diffie-hellman. In M. Yung, Y. Dodis, A. Kiayias, and T. Malkin, editors, *Public Key Cryptography - PKC 2006*, volume 3958 of *LNCS*, pages 229–240. Springer, 2006. `doi:10.1007/11745853_15`.

[C+14] S. A. Chun et al. iSecure Lab, 2014. URL: `http://cis.csi.cuny.edu/~project/iSecure/`.

[CBK15] CBK. Common Body of Knowledge, 2015. URL: `http://nessos-project.eu/cbk`.

[Cen14] Cenzic. Application vulnerability trends report. Technical report, Cenzic, 2014. URL: `http://www.cenzic.com/downloads/Cenzic_Vulnerability_Report_2014.pdf`.

[DDN03] D. Dolev, C. Dwork, and M. Naor. Nonmalleable cryptography. *SIAM Review*, 45(4):727–784, 2003. `doi:10.1137/S0036144503429856`.

[DKF+03] G. Denker, L. Kagal, T. Finin, M. Paolucci, and K. Sycara. Security for daml web services: Annotation and matchmaking. In D. Fensel, K. Sycara, and J. Mylopoulos, editors, *The Semantic Web - ISWC 2003*, volume 2870 of *LNCS*, pages 335–350. Springer, 2003. doi:10.1007/978-3-540-39718-2_22.

[DKF05] G. Denker, L. Kagal, and T. Finin. Security in the semantic web using OWL. *Information Security Technical Report*, 10(1):51 – 58, 2005. doi:10.1016/j.istr.2004.11.002.

[DVOW92] W. Diffie, P. C. Van Oorschot, and M. J. Wiener. Authentication and authenticated key exchanges. *Designs, codes and cryptography*, 2(2):107–125, 1992. doi:10.1007/BF00124891.

[EYZ10] G. Elahi, E. Yu, and N. Zannone. A vulnerability-centric requirements engineering framework: analyzing security attacks, countermeasures, and requirements based on vulnerabilities. *Requirements Engineering*, 15(1):41–62, 2010. doi:10.1007/s00766-009-0090-z.

[FE09] S. Fenz and A. Ekelhart. Formalizing information security knowledge. In *Proceedings of the 4th International Symposium on Information, Computer, and Communications Security*, ASIACCS '09, pages 183–194, New York, NY, USA, 2009. ACM. URL: http://securityontology.securityresearch.at/, doi:10.1145/1533057.1533084.

[FFL13] D. Feledi, S. Fenz, and L. Lechner. Toward web-based information security knowledge sharing. *Information Security Technical Report*, 17(4):199–209, 2013. Special Issue: ARES 2012 7th International Conference on Availability, Reliability and Security. doi:10.1016/j.istr.2013.03.004.

[GB02] A. Gómez-Pérez and V. R. Benjamins, editors. *Knowledge Engineering and Knowledge Management. Ontologies and the Semantic Web, 13th International Conference, EKAW 2002, Spain, 2002, Proceedings*, volume 2473 of *LNCS*. Springer, 2002.

[Hes14] W. Hesse. Ontologie und Weltbezug - vom philosophischen Weltverständnis zum Konstrukt der Informatik (German). *Informatik Spektrum*, 37(4):298–307, 2014. doi:10.1007/s00287-014-0795-3.

[HSD07] A. Herzog, N. Shahmehri, and C. Duma. An ontology of information security. In *International Journal of Information Security and Privacy*, pages 1–23, 2007. URL: https://www.ida.liu.se/~iislab/projects/secont/, doi:jisp.2007100101.

[ISO13] ISO/IEC. 27001: Information technology - Security techniques - Information security management systems - Requirements. Technical report, International Organization for Standardization (ISO) and International Electrotechnical Commission (IEC), 2013.

[J. 12] J. Hodges and C. Jackson and A. Barth. HTTP Strict Transport Security (HSTS). Specification, Internet Engineering Task Force (IETF), 2012. URL: https://tools.ietf.org/html/rfc6797.

[K$^+$15] M. Krötzsch et al. Semantic web, 2015. URL: http://semanticweb.org.

[Kas14] Kaspersky. Kaspersky security bulletin. Technical report, Kaspersky, 2014. URL: http://securelist.com/files/2014/12/Kaspersky-Security-Bulletin-2014-EN.pdf.

[KB97] B. Krieg-Brückner. Seven Years of COMPASS, 1997. URL: http://www.informatik.uni-bremen.de/agbkb/forschung/formal_methods/completed_projects/compass/index_e.htm.

[KB$^+$04] Krieg-Brückner et al. MMiSS - MultiMedia Instruction in Safe Systems, 2004. URL: http://www.mmiss.de/.

[KFL$^+$04] B. Krieg-Brückner, U. Frese, K. Lüttich, C. Mandel, T. Mossakowski, and R. J. Ross. Specification of an Ontology for Route Graphs. In C. Freksa, M. Knauff, B. Krieg-Brückner, B. Nebel, and T. Barkowsky, editors, *Spatial Cognition IV: Reasoning, Action, Interaction, International Conference Spatial Cognition*, volume 3343 of *LNCS*, pages 390–412. Springer, 2004. doi:10.1007/978-3-540-32255-9_22.

[KHL$^+$02] B. Krieg-Brückner, D. Hutter, A. Lindow, C. Lüth, A. Mahnke, E. Melis, P. Meier, A. Poetzsch-Heffter, M. Roggenbach, G. Russell, J. Smaus, and M. Wirsing. MultiMedia Instruction in Safe and Secure Systems. In M. Wirsing, D. Pattinson, and R. Hennicker, editors, *Recent Trends in Algebraic Development Techniques, 16th International Workshop, WADT*, volume 2755 of *LNCS*, pages 82–117. Springer, 2002. `doi:10.1007/978-3-540-40020-2_4`.

[KLK07] A. Kim, J. Luo, and M. Kang. Security ontology to facilitate web service description and discovery. In S. Spaccapietra, P. Atzeni, F. Fages, M.-S. Hacid, M. Kifer, J. Mylopoulos, B. Pernici, P. Shvaiko, J. Trujillo, and I. Zaihrayeu, editors, *Journal on Data Semantics IX*, volume 4601 of *LNCS*, pages 167–195. Springer, 2007. `doi:10.1007/978-3-540-74987-5_6`.

[LH05] S. Lipner and M. Howard. The Trustworthy Computing Security Development Lifecycle. *Developer Network - Microsoft*, 2005. URL: `http://msdn.microsoft.com/en-us/library/ms995349.aspx#sdl2_topic2_5`.

[LMU15] LMU. Web Engineering Group. UWE Website, 2015. URL: `http://uwe.pst.ifi.lmu.de/`.

[M$^+$] Mossakowski et al. IFIP WG 1.3 - Foundations of System Specification. URL: `http://ifipwg13.informatik.uni-bremen.de/`.

[No 15] No Magic Inc. Magicdraw, 2015. URL: `http://www.magicdraw.com/`.

[NVB$^+$13] F. Neuhaus, A. Vizedom, K. Baclawski, M. Bennett, M. Dean, M. Denny, M. Grüninger, A. Hashemi, T. Longstreth, L. Obrst, S. Ray, R. Sriram, T. Schneider, M. Vegetti, M. West, and P. Yim. Towards ontology evaluation across the life cycle: The ontology summit 2013. volume 8, pages 179–194. IOS Press, 2013. URL: `http://ontolog.cim3.net/cgi-bin/wiki.pl?OntologySummit2013_Communique`.

[O$^+$15] M. Odersky et al. Scala, 2015. URL: `http://scala-lang.org/`.

[OAS05] OASIS. eXtensible Access Control Markup Language (XACML) Version 2.0, 2005. URL: `http://docs.oasis-open.org/xacml/2.0/access_control-xacml-2.0-core-spec-os.pdf`.

[Obj05] Object Management Group. XMI 2.1. Specification, OMG, 2005. URL: `http://www.omg.org/spec/XMI/`.

[Obj11] Object Management Group. Unified Modeling Language. Specification, OMG, 2011. URL: `http://www.omg.org/spec/UML/`.

[Ora15] Oracle. Java, 2015. URL: `http://www.java.com/`.

[OWA13] OWASP Foundation. OWASP Top 10 – 2013, 2013. URL: `http://owasptop10.googlecode.com/files/OWASPTop10-2013.pdf`.

[OWA14a] OWASP Foundation. Clickjacking, 2014. URL: `https://www.owasp.org/index.php/Clickjacking`.

[OWA14b] OWASP Foundation. Cross-Site Request Forgery (CSRF) Prevention Cheat Sheet, 2014. URL: `https://www.owasp.org/index.php/Cross-Site_Request_Forgery_(CSRF)_Prevention_Cheat_Sheet`.

[OWA14c] OWASP Foundation. List of useful HTTP headers, 2014. URL: `https://www.owasp.org/index.php/List_of_useful_HTTP_headers`.

[OWA14d] OWASP Foundation. OWASP Risk Rating Methodology, 2014. URL: `https://www.owasp.org/index.php/OWASP_Risk_Rating_Methodology`.

[OWA15a] OWASP Foundation. OWASP Data Validation, 2015. URL: `https://www.owasp.org/index.php/Data_Validation`.

[OWA15b] OWASP Foundation. Session fixation, 2015. URL: `https://www.owasp.org/index.php/Session_fixation`.

[OWA15c] OWASP Foundation. SQL Injection Prevention Cheat Sheet, 2015. URL: `https://www.owasp.org/index.php/SQL_Injection_Prevention_Cheat_Sheet`.

[OWA15d] OWASP Foundation. User Privacy Protection Cheat Sheet, 2015. URL: `https://www.owasp.org/index.php/User_Privacy_Protection_Cheat_Sheet`.

[Pau13] J. Pauli. *The Basics of Web Hacking: Tools and Techniques to Attack the Web*. Syngress, 1 edition, 2013.

[PHP14] PHP. Scripting Language, 2014. URL: `http://www.php.net/`.

[PP06] C. P. Pfleeger and S. L. Pfleeger. *Security in Computing, 4th Edition.* Prentice Hall, 4th edition, 2006.

[Rei14] M. Reithmayer. Tool support for a knowledge base for secure software engineering. Master's thesis, Ludwig-Maximilians-Universität München, 2014.

[RWT14] RWTH Aachen University. i* notation, 2014. URL: `http://istar.rwth-aachen.de`.

[Sch99] B. Schneier. Attack trees. *Dr. Dobb's journal*, 24(12):21–29, 1999. URL: `https://www.schneier.com/paper-attacktrees-ddj-ft.html`.

[She12] M. Shema. *Hacking Web Apps: Detecting and Preventing Web Application Security Problems.* Syngress, 1 edition, 2012.

[SK13] P. Salini and S. Kanmani. Ontology-based representation of reusable security requirements for developing secure web applications. *Int. J. Internet Technol. Secur. Syst.*, 5(1):63–83, 2013. `doi:10.1504/IJITST.2013.058295`.

[SSCW12] A. Souag, C. Salinesi, and I. Comyn-Wattiau. Ontologies for security requirements: A literature survey and classification. In M. Bajec and J. Eder, editors, *Advanced Information Systems Engineering Workshops*, volume 112 of *Lecture Notes in Business Information Processing*, pages 61–69. Springer, 2012. `doi:10.1007/978-3-642-31069-0_5`.

[Sym14] Symantec. Internet security threat report. Technical report, Symantec, 2014. URL: `http://www.symantec.com/de/de/security_response/publications/threatreport.jsp`.

[T. 08] T. Dierks and E. Rescorla. The Transport Layer Security (TLS) Protocol. Version 1.2. Specification, Internet Engineering Task Force (IETF), 2008. URL: `http://tools.ietf.org/html/rfc5246`.

[The15] The MITRE Corporation. Common Vulnerabilities and Exposures List (CVE), 2015. URL: `https://cve.mitre.org`.

[Tia14] J. Tiago. Angriffsrisiken minimieren mit Security-Headern (German). *Heise IX Kompakt Security*, (4/2014):64–53, 2014.

[UG96] M. Uschold and M. Gruninger. Ontologies: principles, methods and applications. *Knowledge Eng. Review*, 11(2):93–136, 1996. `doi:10.1017/S0269888900007797`.

[vTJ11] H. van Tilborg and S. Jajodia, editors. *Encyclopedia of Cryptography and Security.* Springer, 2011. `doi:10.1007/978-1-4419-5906-5`.

[W$^+$15] Wikimedia Foundation et al. Wikidata, 2015. URL: `https://www.wikidata.org`.

[W3C] W3C Recommendation. W3C Web Ontology Language (OWL). URL: `http://www.w3.org/2001/sw/wiki/OWL`.

[W3C14] W3C Last Call Working Draft. Content Security Policy Level 2, 2014. URL: `http://www.w3.org/TR/CSP2/`.

[WCG13] A. Wali, S. A. Chun, and J. Geller. A bootstrapping approach for developing a cyber-security ontology using textbook index terms. In *Eighth International Conference on Availability, Reliability and Security (ARES)*, pages 569–576, 2013. `doi:10.1109/ARES.2013.75`.

[Y$^+$15] E. Z. Yang et al. HTML purifier – standards-compliant HTML filtering, 2015. URL: `http://htmlpurifier.org/`.

All online resources were accessed in January 2015.