# A Metamodel for Web Application Injection Attacks and Countermeasures

Hannes Holm and Mathias Ekstedt

Industrial Information and Control Systems
Royal Institute of Technology
Osquldas väg 10, 100 44 Stockholm, Sweden
{hannesh,mathiase}@ics.kth.se

**Abstract.** Web application injection attacks such as cross site scripting and SQL injection are common and problematic for enterprises. In order to defend against them, practitioners with large heterogeneous system architectures and limited resources struggle to understand the effectiveness of different countermeasures under various conditions. This paper presents an enterprise architecture metamodel that can be used by enterprise decision makers when deciding between different countermeasures for web application injection attacks. The scope of the model is to provide low-effort guidance on an abstraction level of use for an enterprise decision maker. This metamodel is based on a literature review and revised according to the judgment by six domain experts identified through peer-review.

**Keywords:** Cyber security, web applications, enterprise architecture.

## 1 Introduction

Cyber security is a critical concern for enterprises as successful cyber attacks can result in severe economic deficits due to losses of data confidentiality, integrity or availability. Depending on the IT asset in question and the intent of the attacker, there are various cyber attacks that can be considered. For example, an attacker could try to harvest sensitive data through seemingly legitimate emails (i.e., phishing) or exploit a cross site scripting vulnerability (XSS) in a web application. A XSS vulnerability allows an attacker to execute client side script in the web browser of any visitor of the website (which could lead to a range of issues, including complete control of the visitors computer).

Of the various types of cyber attacks available, code injection is often considered the most troubling type of attack [1]. That is, to introduce code into a computer program or system by taking advantage of unchecked assumptions the system makes about its inputs. Code injection attacks can be classified into binary code injection attacks and source code injection attacks [2]. A binary code injection involves insertion of malicious code in a binary program to alter how the program behaves, and is generally carried out through a buffer overflow [3]. Source code injection attacks

S. Aier et al. (Eds.): TEAR 2012 and PRET 2012, LNBIP 131, pp. 198–217, 2012.

involve interaction with applications written in programming languages that do not require compilation, e.g., JavaScript, PHP and SQL statements. As source code injections primarily concerns Web Applications (WA) we hereafter refer to this attack type as web application injections, or WA injections.

WA injections includes a number of different attack types, for example, injections using SQL statements (i.e., SQL injections), XSS and OS Command Injection. These are highly critical software flaws according to OWASP [4] and SANS 2011 Top 25 [5] (which sample all known IT security vulnerabilities).

While large amounts of research have been committed to studying WA injections and organizations such as MITRE, SANS and OWASP have developed security awareness programs to help organizations to mitigate the issue, application developers are still unable to implement effective countermeasures to mitigate these vulnerabilities [6].

One possible reason behind the frequent occurrence of WA injection vulnerabilities is that most IT security related matters involve security tools such as specific vulnerability scanners, static code analyzers and intrusion detection systems. As the security landscape on an abstraction level of tools is rapidly changing and there are various tools available for the same purpose, it is difficult for practitioners to understand what security measures that are worth employing, given different scenarios. This is especially the case if the practitioner has a managerial position such as Chief Security Officer (CSO); this type of practitioner needs to consider the security of system-of-systems as a whole.

Enterprise architectures are typically very complex structures that involve various aspects of relevance other than web applications. Consequently, WA injections only constitute a small part of the overall "security puzzle". Thus, the effort spent to manage countermeasures for this attack type is often very limited and information on the general effectiveness of different countermeasures would be valuable to enterprises.

There are works that have attempted to quantify the general effectiveness of different types of countermeasures. However, every such study has been conducted in the presence of various assumptions that are likely to affect their validity. For example, [7] studied the effectiveness of eight WA firewalls and intrusion prevention systems but did not differentiate between what types of WA injection attacks they prevented. As such, an assumption was made that the tools would be equally effective against all types of such security flaws. Another common assumption is regarding the severity of the concerned vulnerability; most studies do not differentiate between vulnerabilities of the same category (e.g., different cross site scripting (XSS) vulnerabilities), even though vulnerabilities within the same category clearly can be of different importance in practice. For example, CVE-2010-3753 and CVE-2008-5718 are both OS Command Injection vulnerabilities; however, only one of them can provide high level privileges if successfully exploited (CVE-2008-5718).

This paper presents an Enterprise Architecture (EA) metamodel that can be used to aid enterprise decision makers deciding upon different countermeasures for WA injection attacks. A hypothesized metamodel was constructed through a literature

review; this metamodel was then revised by interviews with six domain experts identified through peer-review.

The rest of the paper unfolds as follows: Section 2 describes the literature review used to construct the hypothesized metamodel and Section 3 describes its result. Section 4 presents the methodology for gathering expert judgment. Section 5 presents the findings from the expert study and the revised EA metamodel. Section 6 critically examines these findings. Finally, Section 7 concludes the paper.

## 2 A Literature Review of Web Application Injection Attacks

While there are several categorizations describing different areas of WA injection attacks there is no holistic work on the topic. Thus, there is a need to compile the domain theory on WA injection attacks in order to construct a valid metamodel. This chapter describes the method and result of this literature review.

### 2.1 A Methodology for Categorizing Variables

In order to assemble the currently available work in the field there is a need to both have a way to classify it and a way to collect it. This chapter describes these aspects.

#### 2.1.1 Classifying Current Approaches

Hansman and Hunt [8] present a taxonomy for categorizing network and computer attacks in general that is influenced by Howard's taxonomy [9]. This well established taxonomy is constructed along a set of dimensions, which in combination gives a holistic view of the variables of interest for cyber attacks in general.

**Table 1.** Used categories from the taxonomy by [8]

| Criterion | Description |
|---|---|
| Main means of attack | The attack vector of the cyber attack, e.g., if it is a physical attack or a brute-force password attack. |
| Vulnerabilities and exploits | The vulnerabilities and exploits that the attack uses are either known or unknown to the public at large (i.e., shared on the public domain such as on the US National Vulnerability Database [10]). |
| Result of the attack | The outcome of an attack (denial of service, corruption of information, theft of service, disclosure of information, and/or subversion) |
| Countermeasures | How to defend against the attack. |

As this taxonomy is well established and sufficiently comprehensive to capture the whole domain of WA injections it is chosen to compare the currently available approaches. The criteria target(s) of the attack, Damage, Cost, and Propagation are however not included in the categorization utilized in this study as they significantly involve context-dependent attributes such as the actual targeted software, something which is not useful for the concept of generalizing the attack type. As such, four criteria are used when comparing current WA injection categorization approaches; these criteria are described in Table 1.

### 2.1.2 Collecting Current Approaches

The online databases ACM, IEEE, SCOPUS and ISI Web of Knowledge were chosen as primary sources for collecting current approaches regarding categorizing different types of WA injections. Articles published between January 2000 and March 2012 found using keywords related to the topic of the study had their titles studied. Example searches include "XSS", "SQL injection", "PHP injection", "Web Application attacks" and "XPath attacks". Through this approach a set of articles possibly related to the topic of the study were gathered. A second brief study of the abstracts of these articles delimited this set even further. The final set of papers were thoroughly read. In addition to this approach, any significant work discussed in any of the studied papers was also chosen for further study. This approach resulted in a collection of 12 works that each covers at least one of the four criteria in Table 1.

## 2.2 The Main Means of Attack

Six out of the 12 gathered works [2, 11–15] involves classifying different types of WA injections.

All the studied works [2, 11–15] in one way or another discuss means of attack in terms of programming languages. This paper continues this tradition, using a categorization similar to that of [2]. It is possible to inject data through two different types of languages – *domain specific languages (DSL)* and *dynamic languages (DL)*. These criteria can furthermore be more detailed in terms of actual programming languages, e.g., SQL (DSL), XPath (DSL), JavaScript (DL) or PHP (DL). For example, an SQL injection vulnerability due to an unsanitized input parameter in a PHP application can be exploited through input using a DSL (SQL command). In the same way, a vulnerable exec() variable in a PHP application can be exploited through a dynamic language (an OS command injection through a PHP script). While it certainly is possible to go into further detail, e.g., regarding SQL injection tautologies [14], this would significantly add complexity to the categorization – thus not viable for the purpose of the present study.

## 2.3 Vulnerabilities and Exploits

Three out of 12 studied works discuss topics related to known and unknown WA injection vulnerabilities and exploits [11–13].

Pietraszek and Berghe [11] propose that the CVE (Common Vulnerabilities and Exposures) identification number should be included in the vulnerability information, if applicable. That is, the vulnerability can be known to the public at large. This is contingent to the taxonomy of Hansman and Hunt [8]. Vorobiev and Han [13], and Sidharth and Liu [12] propose that vulnerabilities can be found through querying the WA implementation of Universal Description, Discovery and Integration (UDDI) or Web Service Description Language (WSDL). While this type of information is shared on the public domain, it still involves finding vulnerabilities in a WA, rather testing vulnerabilities known to the public at large (for instance, shared on the US National Vulnerability Database (NVD)).

The categorization used in this paper distinguish, as [11], between *known* and *unknown* vulnerabilities. A vulnerability known to the public at large is an easy target for an attacker as it often also has publicly known exploits, or at least ideas for how to conceive an exploit available. Also, a known vulnerability likely has, or is soon to have, a software patch or work-around remediating it available. This makes it an important vulnerability to manage as it (typically) requires little skill to exploit but (typically) is easily mitigated. There are numerous WA injection vulnerabilities publicly available. For example, the NVD presently describes more than 3300 SQL injection vulnerabilities and almost 3400 XSS vulnerabilities.

### 2.4 Result of the Attack

Four of the 12 studied articles discuss possible results of WA injection attacks [2, 11, 14, 16].

One commonly applied categorization for describing the outcome of an attack is through its impact on confidentiality, integrity and availability (CIA) [2, 16]. While the concept of CIA is somewhat holistic, it can be difficult to relate to as most attacks affect a combination of these criteria. The two remaining works [11, 14] describes attacks on significantly lower abstraction levels (e.g., SQL Union Queries and SQL PiggyBacked Queries). Employing such an abstraction level is however not useful given the purpose of the study; it would simply be too detailed.

A useful categorization for the purpose of this paper is the five criteria proposed by [8]. These criteria constitute a holistic and usable view of possible results from a WA attack. All of the studied approaches [2, 11, 14, 16] are possible to map to it. That is, theft of service, corruption of information, subversion, disclosure of information, and denial of service.

### 2.5 Countermeasures

Nine of the 12 studied works classify different types of countermeasures [2, 11, 12, 14, 17–21]. This section describes the classifications by these authors. Many properties of these categorizations are similar. However, no categorizations fully overlap and the used terminology is highly varied. For example, developing a software using a “secure” API is referred to as *new API’s* [2], *new query development paradigms* [14], and *serialization API’s* [11]. This section summarizes existing

attempts to categorize countermeasures against WA injection attacks, and suggests addition of a variable that is not covered by these.

The two main types of countermeasures in the used categorization are static and runtime approaches, as discussed by [2, 14, 21]. An important distinction between static and runtime countermeasures is that runtime countermeasures do not suggest patches for vulnerabilities in the application codebase, but rather make exploitation of existing vulnerabilities more difficult. In the same fashion, static countermeasures detects (and recommends patches for) vulnerabilities in the application code base, but cannot thwart attacks against any remaining issues. Thus, static measures are often useful before deployment of a WA when code patching is reasonably simple to perform and runtime measures after deployment when code patching can be costly to perform. There is also a combination of them, hybrid approaches, which involve both static analysis for vulnerabilities and run-time analysis of incoming requests. An example of this type of tool is AMNESIA [22].

*Static approaches* involve measures to find and remove vulnerabilities in the application codebase. This category includes black box testing, disabling unnecessary responses, software patching, type-safe API's, and static code analysis.

**Black box testing** [14, 20, 21] involves running automated scanners or fuzzers on deployed WAs without viewing server-side source code. One such example is WAVES [23].

**Disabling unnecessary responses** [12, 14, 17, 19] involves removing any application response messages that are not needed to provide its desired service. For example, any SQL database errors should be eliminated, unnecessary WSDL and UDDI information should be removed and Web server software query responses should be limited. If this countermeasure is successfully implemented it forces the attacker to use "blind" techniques.

**Software patching** is not discussed by any of the studied papers. However, it is clear that it is of importance towards the success of an attack – many organizations do not aim to "reinvent the wheel" and instead deploy commercial-of-the-shelf (COTS) software. Such applications are maintained through software updates by developers which address known vulnerabilities found in their products. Typically, software patching is implemented through an automated patch management tool such as Shavlik [24].

**Type-safe API's** [2, 14] involves using a development environment that is built to function in a secure and reliable fashion. In essence, this countermeasure defines a rule set for allowed code and how different parts of an application exchange information. For instance, how a PHP application is allowed to communicate with an SQL database. If a developer writes code that does not comply with the rule set defined within the type-safe API an error is produced, notifying the developer of the proper syntax as defined by the API. An examples of this type of countermeasure is SQL DOM [25].

**Static code analysis** [2, 11, 14, 20, 21] involves detecting vulnerabilities by analyzing the application source code. That is, to learn of the control or data would flow at runtime without actually executing the code. An example static code analysis tool is Pixy [26].

*Runtime approaches* involve detecting and thwarting ongoing attacks through, e.g., a set of predefined signatures. This category includes content based rejection, query modification and intrusion detection systems.

**Content based rejection** [2, 14, 21] involves analyzing the structure of requests to see if they conform to a model of expected queries. If not, the request is considered to be malicious and as such rejected. One such approach involves creating two *grammatical representations* of input statements using finite state machines or parse trees, one with and one without user input. If the representations do not match the user input is considered to involve a malicious command. An example of this approach is SQLGuard [27]. The perhaps most common type of content based rejection countermeasure in practice is *proxy filters* [12, 14, 17–21] (e.g., application firewalls and gateways) which intercepts calls to WAs to check if requests are malicious (i.e., if they match blacklisted signatures). An example of such countermeasure is Cisco Application Velocity [28]. Thirdly, a popular approach is *dynamic taint analysis*; to mark certain input (e.g., POST) as dangerous and evaluate if it is used in a malicious fashion. An example of this approach is SecuriFly [29].

**Query modification** [2, 11, 14, 20, 21] involves countermeasures aimed to modify queries using predefined functions such as cryptographic keys [2, 14] or through escaping characters [11, 20, 21]. This is also the main difference from content based rejection – query modification accepts modified versions of all input. Both methods as such naturally have pros and cons. An example query modification countermeasure is SQLrand [30].

**Intrusion detection systems (IDS)** [14, 20, 21] involves detecting source code injection attacks. This category differs from the other runtime approaches in the sense that an IDS merely detect, and not thwart, malicious requests. As such, if an IDS is setup to thwart detected issues this categorization treats it as a content based rejection technique (or query modification technique in case it accepts modified input). Intrusion detection systems can be both signature and anomaly based [31]. A signature based IDS have a predefined set of signatures for malicious requests and alarms if a request matches such a signature. An anomaly based IDS is trained on what type of requests that are “normal” and can thus in theory differentiate regular traffic from malicious traffic. A common WA IDS is Apache Scalp [32].

## 3 Hypothesized Metamodel

An EA model describes an organization in terms of the artifacts of business and IT, as well as their interrelationships. An EA metamodel is a description language used when creating EA models. Various EA metamodels have been proposed, for example, general metamodels such as ArchiMate [33] and metamodels for analysis of specific properties such as modifiability [34] and data accuracy [35]. The metamodel presented in this paper is based on the concepts of an existing EA metamodel for cyber security risk analysis, namely, the Cyber Security Modeling Language (CySeMoL) [36]. This section gives a brief overview of the concepts of CySeMoL of relevance to this paper. The reader is referred to [36] for a more detailed description of CySeMoL.

### 3.1 The Cyber Security Modeling Language

The CySeMoL covers a variety of attacks such binary code injections, flooding attacks, abuse of obtained privileges and social-engineering attacks (it does however not cover WA injections). The main objective of CySeMoL is to allow users to create models of their architectures and make calculations on the likelihood of different attacks being successful. Security expertise is not required from the user as the model includes theory on how attributes in the object model depend on each other. In other words, users must only model their system architectures and properties.

The entities in CySeMoL includes various IT components such as `Operating System` (e.g., Windows XP) and `Firewall`, processes such as `Security Awareness Program` (i.e., IT security training) and `Zone Management` (i.e., security maintenance of network zones), and personnel (`Person`). Each entity has a set of attributes that can be either attacks or countermeasures. These attributes are related in various ways. For example, the credentials of personnel can be social engineered – but the likelihood of this attack being successful depends on whether the person has undergone security awareness training or not. Each attribute in CySeMoL has a binary range (True / False), i.e., the likelihood of an attack being successful and the likelihood of a countermeasure being functional.

### 3.2 A Hypothesized Metamodel

The attributes found during the literature review (cf. Section 2) can be mapped to four entities: the WA itself (`WebApplication`), the process for developing the WA (`SoftwareDevelopmentProcess`), the process for maintaining the WA (`SoftwareMaintenanceProcess`) and whether there is an intrusion detection system monitoring the WA (`IntrusionDetectionSystem`). An overview of the metamodel can be seen in Fig. 1. Each entity is associated with a set of attributes with binary ranges (i.e., true or false). An attribute can be either an attack (a means of compromising the entity) or a countermeasure (a means to counter attacks). `WebApplication` is the only entity which is associated to attacks – an attacker can achieve an intended *result* (e.g., denial of service) by exploiting a known or unknown domain-specific language or dynamic language *vulnerability*. The presence of vulnerabilities in turn depend on the presence of *countermeasures* applied during development and maintenance of the WA, and if there is an intrusion detection system present, i.e., whether `IntrusionDetectionSystem.Deployed = True`. Due to these relational dependencies, the user of the model only needs to specify the states of attributes without parents (i.e., attributes without any arrows directed towards them), which in practice means the countermeasures.

The countermeasures corresponding to `SoftwareDevelopmentProcess` and `SoftwareMaintenanceProcess` next to completely overlap. However, two measures differ between them: Type-safe API's are only used during development of a WA as they require the application codebase to be written using specific constraints. Similarly, automated patch management is a tool that only can be used for finished applications. Also, automated patch management can unlike the remainder of the countermeasures due to its nature only mitigate known vulnerabilities.

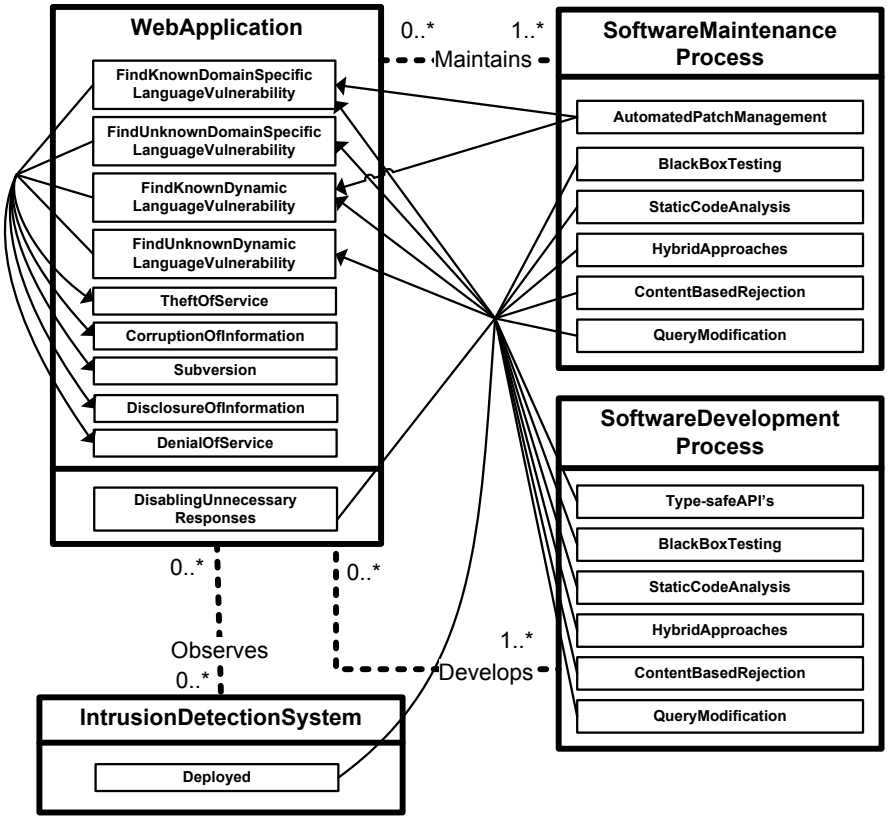


**Fig. 1.** A hypothesized metamodel for WA injections

## 4 Methodology for Revising the Hypothesized Metamodel

The complexity of this research means that it will be difficult to validate the metamodel using an experimental approach. Given such a scenario expert judgment can be justified as a means of estimation [37]. This study utilizes a combination of interviews and a workshop in order to revise the hypothesized metamodel (cf. Fig. 1).

### 4.1 Population and Sampling

In terms of the expert categories described in [38] individuals that are expert judges are desirable. Studies of experts' calibration have concluded that experts are well calibrated in situations with learnability and with ecological validity [39]. Learnability comes with models over the domain, the possibility to express judgment in a coherent quantifiable manner and the opportunity to learn to from historic predictions and outcomes. Ecological validity is present if the expert is used to making judgments of the type of questions they are asked. An individual that has significant and up to date

professional experience from working with WA security testing should likely possess both learnability and ecological validity. This type of individual can be seen as the population of the present study.

The respondents used during the present study were identified through peer-review by prominent members of Swedish OWASP chapters. Each of the six individuals participating in the study had significant and fresh experience from professional work with WA security.

### 4.2 Data Collection

This study utilizes a combination of three semi-structured interviews and a workshop with three individuals, all carried out face-to-face. Due to the complexity of the matter effort was spent to enforce reliability of results. That is, the original layout and scope of the data collection was somewhat changed according to the focus area(s) of the respondents. For example, no answers were forced, the scales were allowed to be switched for a ranking system, and the respondents were allowed to traverse from the original scope if needed. For example, if they wanted to discuss a particular countermeasure in greater detail. As a consequence, more time was spent on those matters the respondents perceived to be of greater importance for the topic of the study.

The interviews and the workshop all had the same general approach. A summary of this methodology is described below. The approach consisted of two main objectives: (i) to study what aspects of the hypothesized metamodel that should be revised, and (ii) to estimate the general effectiveness of different countermeasures given the revised metamodel defined by the respondent(s), i.e., what attribute relations that should be present in the metamodel.

#### 4.2.1 Revision of Hypothesized Metamodel

The first part of the interview or workshop concerned describing the topic of the study and the outline of the event. After this the respondents were given a graphical description of the proposed metamodel (cf. Fig. 1) and introduced to the concepts of it. The second part concerned the countermeasures of the metamodel. Effort was spent to identify if the abstraction levels of the countermeasures were reasonable given the scope of the study and if any countermeasures should have been changed, removed or added. A specific focus during this phase was placed on that the countermeasure must be applicable in practice, i.e., it must be readily available to practitioners and reasonably effortless to deploy and manage. The third part concerned the difference between known and unknown vulnerabilities; if this concept should be changed, and elicit the dependencies between countermeasures and the variables of this type. The fourth part involved if the employed types of WA injection attacks were useful (i.e., attacks for domain specific and dynamic language vulnerabilities), or if some aspects should have been revised. It also involved eliciting the dependencies between countermeasures and attack types – if any countermeasure was more competent at mitigating some attack types than others. The fifth part concerned the different

categorized results of successful attacks (e.g., denial of service); if anything should be revised, and whether any countermeasure was more viable for mitigating attacks of different outputs than others.

### 4.2.2 Estimations of the Effectiveness of Countermeasures

The sixth part concerned identifying dependencies between the different countermeasures (using the information identified in step 1-5). That is, what combination of approaches that provide significantly greater effectiveness (and which that do not). The seventh part involved quantitatively scoring each countermeasure according to its mean effectiveness and variance (using the dependencies and information identified in the previous steps) through a scale of 1-5. In terms of mean effectiveness, 1 meant "do not increase the difficulty of successful attack" and 5 "greatly increases the difficulty of successful attack", and in terms of variation 1 meant "very small variation" and 5 "very high variation". To decrease ambiguity, the respondents were told that the variation was "if you would pick two countermeasures at random from the countermeasure category, how much would their effectiveness typically differ?". This quantitative scoring was carried out for one countermeasure at a time until all had been scored.

### 4.2.3 Respondents Part of the Study

The first interview (I1) lasted for 1.5 hours. The respondent of this interview had 7 years of relevant professional experience and works with software penetration testing in general; finding vulnerabilities in software written in, for example, C++, JavaScript, and PHP. Respondent 1 had also significant previous professional experience from network penetration testing.

The second interview (I2) lasted for 2.5 hours and involved a respondent with 10 years of WA security experience that works within the area of WA security. For instance, penetration tests of software written in PHP, Perl or .NET, and communications with database solutions such as SQL. This individual also performed occasional network penetration tests.

The third interview (I3) lasted for 1.5 hours and involved a respondent with 12 years experience who did not presently work, but had previously done so, with software penetration testing. This individual works as the chief technology officer of an enterprise specializing in WA security. For instance, penetration tests of WAs and security awareness training of developers. This individual is required to inhibit knowledge of all of these aspects.

The workshop (WS) lasted for 3 hours and involved three respondents whom all performed similar work as the second respondent. These individuals had 7, 3, and 3 years of professional experience from WA security. Notable is that the two respondents with 3 years of professional experience of WA injections had worked extensively for many years on the matter before having it as a main profession.

# 5 A Metamodel Revised by Expert Judgment

This section concerns data collected through three interviews and a workshop. To make results more pedagogical, the opinions by the three respondents of the workshop are unified at all occurrences where complete consensus was reached between them. Agreement among experts is also used as a basis for revising the hypothesized metamodel (cf. Fig. 1). Consensus was chosen for this purpose as it has been shown to outperform competing indicators of expert calibration [40].

## 5.1 Changes to the Metamodel Prescribed by the Experts

This section describes the revisions that the experts recommended for the hypothesized metamodel.

### 5.1.1 Type of Attacker

This variable is not part of the hypothesized metamodel. However, the type of attacker in question came up very early during each session – the skill level of the studied attacker was perceived to greatly affect the effectiveness of the countermeasures. Each of these discussions resulted in two basic categories of attackers: Advanced Persistent Threats (*APT*) and *Noise*. An APT is an experienced attacker that knows how to cover its tracks [41]. Noise, or a script kiddie, is an attacker that has a very limited cyber security experience and depend a lot on automated tools created by more experienced hackers [41]. It is expected that the effort required to defend against an APT is much higher than for a noise attack [41].

### 5.1.2 Changes to Attack Types

None of the respondents thought that the effectiveness of any of the studied countermeasures were significantly dependent on the type of attack that is conducted – at least not on a level that would suit the purpose of the current study. For example, the respondents of the workshop denoted that it can be more difficult to find dynamic language vulnerabilities (e.g., OS Command injections) than domain-specific language vulnerabilities (e.g., SQL injections) as there traditionally are no error messages provided during the probing. Such vulnerabilities are likely easier to find through white box analysis rather than black box analysis. However, given the purpose of the study, the respondents did not perceive it is useful to detail these aspects. Consequently, this category is removed from the metamodel.

### 5.1.3 Changes to Results of Attacks

As for the *type of attack*, all respondents thought that the variables of the *attack result* category (cf. Section 2.4) should be aggregated into a single variable. That is, the different countermeasures are not significantly more effective at preventing, for example, denial of service attacks compared to attacks which aim to corrupt information. Thus, this category is removed from the metamodel.

### 5.1.4 Changes to Vulnerabilities

All of the respondents perceived that there was a significant difference in effectiveness of the countermeasures depending on whether the vulnerability and exploit was known to the public at large or not. As such, the qualitative structure regarding this variable is the same as the variables presented in Section 2.3.

### 5.1.5 Changes to Countermeasures

The results from each data collection event are fairly similar in terms of recommended changes to the concepts of the hypothesized metamodel (cf. Fig. 1). There are however a few notable suggestions by the experts.

**Table 2.** Recommended revisions to countermeasures

| Countermeasure | I1 | I2 | I3 | WS |
|---|---|---|---|---|
| Software patching | None | None | None | None |
| Disabling unnecessary responses | None | None | None | Revise[a] |
| Black box testing | None | None | None | Revise[b] |
| Type-safe API's | Remove | None | None | None |
| Static code analysis | None | None | None | - |
| Hybrid approaches | Remove | Remove | Remove | Remove[c] |
| Content based rejection | None | Revise[d] | Revise[d] | Revise[d] |
| Query modification | Remove | Remove | Remove | None |
| Intrusion detection systems | None | Revise[e] | Revise[e] | Revise[e] |
| Disabling unnecessary services | Add | - | - | - |
| Developer security training | Add[f] | Add | Add | Add[f] |
| A formalized development process | - | - | Add | - |

[a] Change for Configuration management.
[b] Change for Vulnerability scanning.
[c] Effective, but not used in practice.
[d] Change for Web Application Firewall (prevent).
[e] Change for Web Application Firewall (detect).
[f] Important, but difficult to estimate and categorize.

One notable recommendation was to replace Content based rejection and Intrusion Detection Systems for Web Application Firewalls (WAF), a common type of countermeasure that often is employed after deployment of a WA. A WAF can be configured to automatically prevent detected attacks or to report of their occurrence to human operators.

Another notable revision prescribed by the experts was removal of hybrid approaches. The experts believed that this would be an effective solution in the future, but not something that was practically available at the time of the study.

A third notable revision made by suggestion of the experts was replacing query modification by the broad category of developer security training. A developer that have been security trained can be perceived to both be able to produce more secure

code and be able to apply countermeasures such as static code analysis in a more effective way [14, 20].

## 5.2 Estimates on the Significance of Relations between Attributes

This section describes the estimates made by the experts on the effectiveness of the countermeasures, both alone and in combination with others'. I.e., it analyzes the significance of the attribute relations of the metamodel.

### 5.2.1 Effectiveness of Individual Countermeasures

The quantitative estimates made by the respondents can be seen in Table 3. These estimates are made under the assumption that no countermeasure other than the one studied is present. The effectiveness of the countermeasures was studied based on the revised metamodel as seen by the experts (cf. Section 5.1). As the respondents prescribed removing attributes related to *the type of attack* and *the results of attacks* the hypothesized dependencies regarding these attributes were not analyzed. While the experts made estimations for both known and unknown vulnerabilities, data is only provided for known vulnerabilities. This as the experts depicted exactly the same values for them - the only difference is that software patching per definition is not effective for unknown security issues.

**Table 3.** Estimates on the effectiveness of countermeasures by experts

| | Mean effectiveness (1-5) | | | | | | | | Mean variance (1-5) | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | Noise | | | | APT | | | | Noise | | | | APT | | | |
| Countermeasure | I1* | I2 | I3 | WS | I1* | I2 | I3 | WS | I1 | I2 | I3 | WS | I1 | I2 | I3 | WS |
| Software patching** (SP) | A | 4 | 5 | 5 | A | 1 | 4 | 2 | 4 | 5 | 1 | 1 | 4 | 5 | 1 | 2.5 |
| Disabling unnecessary responses (DUR) | D | 2 | 5 | 4 | D | 2 | 3 | 4 | 4 | 5 | 2 | 4 | 4 | 5 | 2 | 4 |
| Black box testing (BBT) | B | 3 | 3 | 4 | B | 3 | 2 | 4 | 2 | 2 | 2 | 4.5 | 2 | 2 | 3 | 3.5 |
| Type-safe API's (API) | - | 4 | 5 | 5 | - | 4 | 3 | 5 | - | 4 | 1 | 1 | - | 4 | 2 | 1 |
| Static code analysis (SCA) | E | 4 | 5 | - | E | 3 | 4 | - | 2 | 2 | 2 | - | 2 | 2 | 3 | - |
| Web Application firewall, reject (WAFr) | C | 4 | 5 | 3.5 | C | 3 | 5 | 2 | 3 | 3 | 2 | 3 | 3 | 3 | 2 | 2 |
| Web Application firewall, detect (WAFd) | F | 1 | 5 | 2 | F | 1 | 4 | 1 | 3 | 3 | 3 | 2 | 3 | 3 | 3 | 2 |
| Developer security training (DST) | - | 4 | 4 | - | - | 4 | 4 | - | - | 4 | 4 | - | - | 4 | 4 | - |

* Ranked from A: most effective to F: least effective.
** Only effective for known vulnerabilities.

Notable is that the first interview respondent did not feel comfortable providing quantitative estimates, and thus instead ranked the countermeasures according to their effectiveness (from A: most effective to F: least effective).

The consensus is rather low regarding mean effectiveness and mean variance for most countermeasures. There is however agreement regarding some aspects: all respondents perceive software patching to be the most effective countermeasure given known vulnerabilities and noise attacks. Each respondent perceives black box testing

to be fairly effective against noise attacks, and all interview respondents seemingly agree that the variance in effectiveness between tools in this category is fairly low. It is however clear that all attribute relations are required to be modeled.

### 5.2.2 Effectiveness of Countermeasures in Combination

Oftentimes, the effectiveness of one countermeasure can be thought of as dependent on the presence of another. None of the respondents however perceived that the dependencies between countermeasures would depend on the type of attacker in question. As a consequence, two dependence matrixes were scored during each data collection event: one for known vulnerabilities (cf. Table 4) and one for unknown vulnerabilities. As for the estimates on the effectiveness of individual countermeasures, the only difference between the two dependence matrixes analyzed by the respondents is that software patching is not applicable for unknown vulnerabilities. As a consequence, all combinations including software patching in the matrix for unknown vulnerabilities were denoted as not effective by all respondents. This is also the reason for why it is not shown in the paper.

In Table 4, a "0" means that the combination of countermeasures is not perceived to result in a significant increased effectiveness. A "+" means that the combination is perceived to result in a significantly increased effectiveness. A "*" means that a data collection event did not detail the perceived dependency, i.e., it was not part of the metamodel proposed by the respondent(s) (cf. Section 5.1). Interview 1 is the first symbol in each cell, interview 2 the second, interview 3 the third, and the workshop the fourth. For example, the combination between type-safe API's and prevention based WA firewalls has the symbols "*++0". That is, respondent 1 (I1) did not have type-safe API's in the perceived metamodel; and thus the first symbol is"*". The second and third respondents (I2 and I3) perceived a significant increased effectiveness of the combination, and thus the second and third symbols are "+". The respondents of the workshop (WS) did not perceive the combination to result in any significant increased effectiveness, and as such the fourth symbol is "0".

**Table 4.** Dependencies between countermeasures regarding known vulnerabilities. "+" denotes perceived increased effectiveness, "0"denotes no perceived increased effectiveness, and "*"denotes that the combination was not scored.

| | SP | DUR | BBT | API | SCA | WAFr | WAFd |
|---|---|---|---|---|---|---|---|
| DUR | ++++ | | | | | | |
| BBT | 0+++ | ++++ | | | | | |
| API | *0++ | *+++ | *+++ | | | | |
| SCA | 00+* | +++* | +++* | *++* | | | |
| WAFr | +0++ | ++++ | +++0 | *++0 | +++* | | |
| WAFd | 00+0 | 00+0 | 00+0 | *0+0 | 00+* | 00+0 | |
| DT | *0+* | *++* | *++* | *++* | *++* | *++* | *++* |

Most combinations are perceived to have significant increased effectiveness. However, the perceived non-significance of combinations involving WAFd could seem curious. The reason behind this is that all respondents except the third interview respondent perceived that the attack would be successful long before the eventual response by operators seeing the alarm. The third respondent viewed a WAFd as a good indicator of overall threat, something the respondent believed to be of importance.

### 5.3 Revised Metamodel for Web Application Injections

An overview of revised metamodel, formulated from the consensus by the six respondents, can be seen in Fig. 2. All attributes except `Attacker.Skill` and `WebApplicationFirewall.Functionality` have state-spaces of {*True*, *False*}. `Attacker.Skill` can take the states {*Noise*, *Advanced Persistent Threat*} and `WebApplicationFirewall.Functionality` can take the states {*Prevent*, *Detect and report*}.

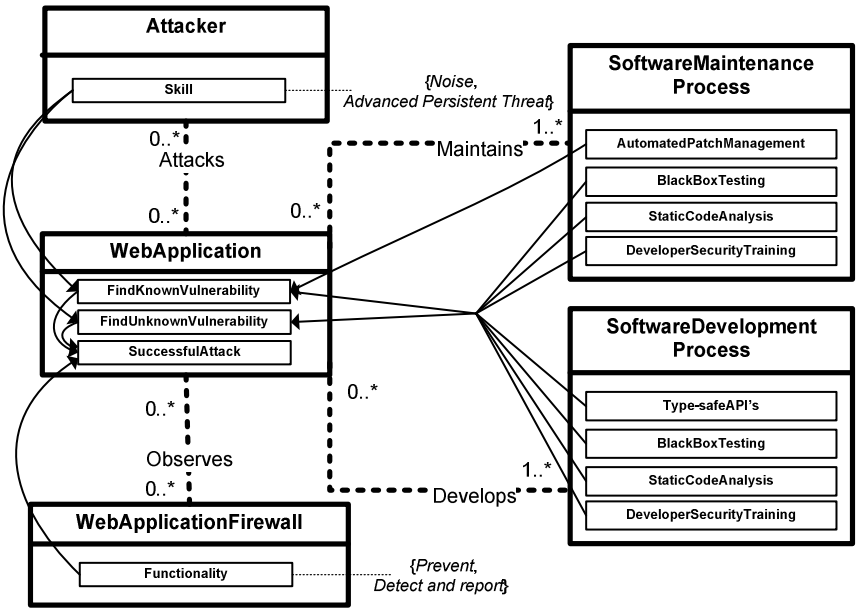


**Fig. 2.** A revised metamodel of WA injections

As can be seen, while the entity `Attacker` has been added, the revised metamodel is significantly smaller than the hypothesized metamodel (cf. Fig. 1). A less detailed metamodel is preferable due to two main reasons: (i) it requires less effort to manage by the modeler and (ii) it performs better when simulating attacks using the CySeMoL calculation engine (due to a smaller total state-space).

## 6 A Critical Discussion of Research Findings

While the hypothesized metamodel is constructed from peer-reviewed domain theory, the revised metamodel is subject to potential bias, for example, bias due to the chosen sample [42]. The findings from this study are based on assessments by a small number of individuals. Even though these individuals were selected based on recommendations by their peers and had significant experience on the topic it is difficult to say that their estimates are representative for the population at large. Nevertheless, while it is important to recognize these delimitations, the results provided by this paper give valuable input that no previous study has analyzed.

Another bias that is important to address is the bias due to the data collection methodology [42]. Moser [42] argue that there are three possible bias in terms of data collection methodology through interviews: (i) bias due to the interaction of interviewer and respondent, (ii) bias due to factors connected with the questionnaire, and (iii) bias due to factors connected with the setup and circumstances of the interview. The author describes a list of factors that are of importance in order to limit bias due to these three factors; these suggestions were consulted when formulating interview questions and collecting data during the present study. A few significant decisions made due to recommendations by [42] are described below.

The interviews and the workshop were carried out using the same procedure (cf. Section 4.2), using a structured procedure. Also, no respondent had any previous affiliation with the interviewer. These aspects should serve to reduce the threat of interview bias.

To handle the complexity of the research purpose, the questionnaire was broken down into a sequence of different topics (cf. Section 4.2). The sub-session corresponding to each of these topics were introduced by the interviewer at the beginning of them.

The outcomes of the interviews and the workshop were presented to the corresponding respondents to enable them to correct any issues. No respondent found any issues which they wanted to address.

Another potential bias is that respondents, if pressured, can provide answers which they do not really believe in [43]. This is of particular significance to a study such as the present, with complex high-level questions that can be perceived as difficult to answer. To counter this issue, no answers were forced. Furthermore, the format of the estimates could be changed to better suit the respondent. These options were utilized twice in the present study: the respondents of the workshop did not feel comfortable addressing static code analyzers, and the first respondent did not feel comfortable with the measurement scale of “mean effectiveness”. As a consequence, the interview instrument was revised during these occasions to accommodate their needs.

## 7 Conclusions

While decision support on an abstraction level of actual security tools (e.g., WAVES or AMNESIA) is useful in the sense that it provides *accurate* information it would

end up with a significant number of options which would require large amounts of resources to parse – something which rarely is available, especially as WA injection attacks only constitute one small piece of the "security puzzle". The present work presents a metamodel that can aid enterprise decision makers with a language for modeling WA injections and estimating the effectiveness of different countermeasures for the attack type. However, in order to decide upon a specific WA security solution, practitioners' are naturally in need to consider more precise and valid knowledge.

The results also indicate that some countermeasures do seem to outperform others (i.e., some attribute relations are more significant than others). The expert judgment indicate that type-safe API's is the most effective approach – given that there is a possibility to manipulate the software code base. Under other circumstances, things are a bit different. Software patching is the most effective means of handling publicly known security issues and noise attacks. Static code analysis is the most effective for known security issues and APT, and for unknown security issues for both noise and APT. Hybrid based countermeasures are not a useful method of countermeasure as of yet, but are perceived to be a more viable solution in the future. It is also often times perceived to be useful to employ combinations of different countermeasures.

Finally, it is important to acknowledge that the results presented in this paper only provide tentative findings - in order to enable sound conclusions regarding the topic of the study there is a need to perform further studies with more samples that can be perceived representative of the population.

## References


1. Tsipenyuk, K., Chess, B., McGraw, G.: Seven pernicious kingdoms: A taxonomy of software security errors. IEEE Security & Privacy 3, 81–84 (2005)
2. Mitropoulos, M.D., Karakoidas, V., Louridas, P., Spinellis, D.: Countering Code Injection Attacks: A Unified Approach. Information Management & Computer Security 19, 3 (2011)
3. One, A.: Smashing the stack for fun and profit (1996), http://ezano-secu.fr/securite/Applicatif/Smashing_the_stack_for_fun_and_profit.pdf
4. OWASP: 2010 OWASP Top 10 (2010)
5. Martin, B., Brown, M., Paller, A., Kirby, D., Christey, S.: 2011 CWE/SANS Top 25 Most Dangerous Software Errors (2011)
6. Scholtea, T., Balzarottib, D., Kirdac, E.: Have things changed now? An empirical study on input validation vulnerabilities in web applications. Computers and Security (2012)
7. Suto, L.: Analyzing the Effectiveness of Web Application Firewalls (2011)
8. Hansman, S., Hunt, R.: A taxonomy of network and computer attacks. Computers & Security 24, 31–43 (2005)
9. Howard, J.D.: An analysis of security incidents on the Internet 1989-1995 (1997)
10. NVD: National Vulnerability Database, http://nvd.nist.gov/
11. Pietraszek, T., Berghe, C.: Defending Against Injection Attacks Through Context-Sensitive String Evaluation. In: Valdes, A., Zamboni, D. (eds.) RAID 2005. LNCS, vol. 3858, pp. 124–145. Springer, Heidelberg (2006)

12. Sidharth, N., Liu, J.: IAPF: A Framework for Enhancing Web Services Security. The Computer Society (2007)
13. Vorobiev, A., Han, J.: Security attack ontology for web services. In: Second International Conference on Semantics, Knowledge and Grid, SKG 2006, p. 42. IEEE (2006)
14. Halfond, W., Viegas, J., Orso, A.: A classification of SQL-injection attacks and countermeasures. In: Int'l Symp. on Secure Software Engineering, Citeseer (2006)
15. Zuchlinski, G.: The Anatomy of Cross Site Scripting (November 2003)
16. Álvarez, G., Petrovi, S.: A new taxonomy of web attacks suitable for efficient encoding. Computers & Security 22, 435–449 (2003)
17. Stamos, A., Stender, S.: Attacking Web Services: The Next Generation of Vulnerable Enterprise Apps. In: BlackHat 2005 (2005)
18. Klein, A.: Blind XPath Injection. Whitepaper from Watchfire (2005)
19. Ghourabi, A., Abbes, T., Bouhoula, A.: Experimental analysis of attacks against web services and countermeasures. In: Proceedings of the 12th International Conference on Information Integration and Web-based Applications & Services, pp. 195–201. ACM (2010)
20. Nystrom, M.: Sql injection defenses. O'Reilly Media, Inc. (2007)
21. Shin, Y., Williams, L.: Toward A Taxonomy of Techniques to Detect Cross-site Scripting and SQL Injection Vulnerabilities (2008)
22. Halfond, W.G.J., Orso, A.: AMNESIA: analysis and monitoring for NEutralizing SQL-injection attacks. In: Proceedings of the 20th IEEE/ACM International Conference on Automated Software Engineering, pp. 174–183. ACM (2005)
23. Huang, Y., Huang, S.: Web application security assessment by fault injection and behavior monitoring. In: Proceedings of the 12th International Conference on World Wide Web, pp. 148–159. ACM (2003)
24. Shavlik: Shavlik Technologies, http://www.shavlik.com/
25. McClure, R.A., Krüger, I.H.: SQL DOM: compile time checking of dynamic SQL statements. In: Proceedings of the 27th International Conference on Software Engineering, pp. 88–96 (2005)
26. Jovanovic, N., Kruegel, C., Kirda, E.: Pixy: A static analysis tool for detecting web application vulnerabilities (short paper). In: Proceedings aof the 2006 IEEE Symposium on Security and Privacy, pp. 258–263. IEEE Computer Society (2006)
27. Buehrer, G., Weide, B.W., Sivilotti, P.A.G.: Using parse tree validation to prevent SQL injection attacks. In: Proceedings of the 5th International Workshop on Software Engineering and Middleware, pp. 106–113. ACM (2005)
28. Cisco: Cisco Application Velocity System, http://www.cisco.com/en/US/products/ps6499/index.html
29. Livshits, B., Martin, M., Lam, M.S.: Securifly: Runtime protection and recovery from web application vulnerabilities (2006)
30. Boyd, S.W., Keromytis, A.D.: SQLrand: Preventing SQL Injection Attacks. In: Jakobsson, M., Yung, M., Zhou, J. (eds.) ACNS 2004. LNCS, vol. 3089, pp. 292–302. Springer, Heidelberg (2004)
31. Denning, D.E.: An intrusion-detection model. IEEE Transactions on Software Engineering, 222–232 (1987)
32. apache-scalp: Apache log analyzer for security, http://code.google.com/p/apache-scalp/
33. Lankhorst, M.: Enterprise architecture at work: Modelling, communication and analysis. Springer-Verlag New York Inc. (2009)

34. Lagerström, R.: Analyzing system maintainability using enterprise architecture models. Journal of Enterprise Architecture 3, 33–42 (2007)
35. Närman, P., Holm, H., Johnson, P., König, J., Chenine, M., Ekstedt, M.: Data accuracy assessment using enterprise architecture. Enterprise Information Systems 5, 37–58 (2011)
36. Sommestad, T., Ekstedt, M., Holm, H.: The Cyber Security Modeling Language: A Tool for Assessing the Vulnerability of Enterprise System Architectures. IEEE Systems Journal (to be available)
37. Cooke, R.: Special issue on expert judgment. Reliability Engineering & System Safety 93, 655–656 (2008)
38. Weiss, D.J., Shanteau, J.: Empirical Assessment of Expertise. Human Factors: The Journal of the Human Factors and Ergonomics Society 45, 104–116 (2003)
39. Bolger, F., Wright, G.: Assessing the quality of expert judgment: Issues and analysis. Decision Support Systems 11, 1–24 (1994)
40. Holm, H., Sommestad, T., Ekstedt, M., Honeth, N.: Indicators of expert judgment and their value: an empirical investigation in the area of cyber security. Expert Systems: The Journal of Knowledge Engineering (to be available)
41. Bodeau, D.J., Graubart, R., Fabius-Greene, J.: Improving Cyber Security and Mission Assurance Via Cyber Preparedness (Cyber Prep) Levels. In: 2010 IEEE Second International Conference on Social Computing, pp. 1147–1152. IEEE (2010)
42. Moser, C.: Interview bias. Review of the International Statistical Institute, 28–40 (1951)
43. Crespi, L.: The interview effect in polling. Public Opinion Quarterly 12, 99–111 (1948)