Information Sciences 254 (2014) 19–38

Contents lists available at ScienceDirect

Information Sciences

journal homepage: www.elsevier.com/locate/ins

Semantic security against web application attacks

Abdul Razzaq [*], Khalid Latif, H. Farooq Ahmad, Ali Hur, Zahid Anwar, Peter Charles Bloodsworth

School of Electrical Engineering and Computer Science, National University of Science and Technology, Islamabad, Pakistan

A R T I C L E   I N F O

Article history:
Received 17 May 2011
Received in revised form 4 July 2013
Accepted 6 August 2013
Available online 19 August 2013

Keywords:
Application security
Semantic security
Semantic rule engine

A B S T R A C T

In this paper, we propose a method of detecting and classifying web application attacks. In contrast to current signature-based security methods, our solution is an ontology based technique. It specifies web application attacks by using semantic rules, the context of consequence and the specifications of application protocols. The system is capable of detecting sophisticated attacks effectively and efficiently by analyzing the specified portion of a user request where attacks are possible. Semantic rules help to capture the context of the application, possible attacks and the protocol that was used. These rules also allow inference to run over the ontological models in order to detect, the often complex polymorphic variations of web application attacks. The ontological model was developed using Description Logic that was based on the Web Ontology Language (OWL). The inference rules are Horn Logic statements and are implemented using the Apache JENA framework. The system is therefore platform and technology independent.

Prior to the evaluation of the system the knowledge model was validated by using OntoClean to remove inconsistency, incompleteness and redundancy in the specification of ontological concepts. The experimental results show that the detection capability and performance of our system is significantly better than existing state of the art solutions. The system successfully detects web application attacks whilst generating few false positives. The examples that are presented demonstrate that a semantic approach can be used to effectively detect zero day and more sophisticated attacks in a real-world environment.

1. Introduction

Web application security protects the confidentiality, integrity and availability of resources in cyber space. Web services and applications have shaped a new landscape of information sharing which has increased the productivity of e-business. On the other hand however, the number of cyber-threats has also increased tremendously due to the growing popularity of web applications [40,79,28,10,37]. Enormous efforts have been made to mitigate these attacks through various security mechanisms in the form of scanners, intrusion detection systems, encryption devices, and web application firewalls. These traditional security solutions often apply a signature based approach and are only effective therefore against “well-known” security flaws where signatures are already present in the solution database. Signature-based systems maintain a database of the easily identifiable “signatures” of known attacks and apply pattern matching algorithms for attack detection. Unchecked input validation is a major source of attacks at the web application level.

* Corresponding author. Tel.: +92 51 9085 2400.
E-mail addresses: abdul.razzaq@seecs.edu.pk (A. Razzaq), khalid.latif@seecs.edu.pk (K. Latif), farooq.ahmad@seecs.edu.pk (H.F. Ahmad), ali.hur@seecs.edu.pk (A. Hur), zahid.anwar@seecs.edu.pk (Z. Anwar), peter.bloodsworth@seecs.edu.pk (P.C. Bloodsworth).

0020-0255/$ - see front matter © 2013 Elsevier Inc. All rights reserved.
http://dx.doi.org/10.1016/j.ins.2013.08.007

According to the Open Web Application Security Project (OWASP) [50], four out of the current top ten vulnerabilities are related to input validation. Failure to protect web applications from invalid inputs, can often cause costly security breaches to organizations. Hacking incidents can result in the theft of sensitive data, defacement of web sites, privilege escalation, and unauthorized access to a system. A hacker might also be able to inject malicious code to bypass or modify the intended functionality of a program without the user knowing. For example, in an XSS attack, the user is deceived into clicking on a link which points to a trusted site, but it actually contains a malicious script which was written by a hacker. For example:

```
http://www.citibank.com <script> http://www.evil.com/payload?c= '+document.cookie</script>
```

Encoding schemes also play an important role in deceiving an attack vector and commonly facilitate the malicious intention of the hacker. For example in case of a directory traversal attack the string '../' is considered as a malicious string but it can also be presented in hexadecimal encoding "%252E%252E%252F". Similarly there are hundreds of ways to launch SQL Injection attack. A few samples are shown in Fig. 1.

Traditional security solutions such as web scanners, provide the first line of defense against attack and detect well-known security flaws using threat signatures. Scanners lack semantics and are thus unable to make intelligent decision upon data leakage or business logic flaws [24]. This frequently results in false alarms being raised and such approaches also fail to detect novel and critical vulnerabilities [45]. Signature based solutions maintain a white and black list of processes. These contain the signatures of benign inputs and also those of malicious attack vectors. The use of such lists requires continuous updating of threat signatures. Failing to do so may result in limited or no protection against zero day attacks and the generation of too many false positive and false negative alerts [56]. Furthermore, most network solutions ignore the payload and only scan the headers of a user request. Due to a lack of information regarding the application context, network solutions are commonly ineffective against application level attacks. A huge amount of effort has been expended by the security industry to mitigate these attacks through various state of the art security mechanisms in the form of scanners, intrusion detection systems, encryption devices, and firewalls. These measures have unfortunately so far been unable to achieve the security level that is necessary for web applications. There is a clear need therefore for a different approach.

Semantic-based systems are designed to intelligently understand the application's context, the data and the nature of possible attacks. Such systems validate the input to it at both the syntactic and semantic levels. Syntax based validation commonly applies size or content restrictions. Semantic based validation on the other hand mainly focuses on specific data types, formats and an understanding of potentially malicious commands with respect to their context and likely consequences. In recent years such semantic approaches have shown promise in terms of providing rich representations of web application domain knowledge [29]. Viswanathan and Krishnamurthi [77] proposed a personalization approach for making semantic relationship paths by capturing the user's interest level in various domains through their web browsing history. Rubiolo et al. [62] presented a technique that was based on an Artificial Neural Network model, for knowledge discovery through ontology matching on the Semantic Web. The concept of semantic annotations was applied recently to develop an identity management system which was deployed as a firewall for protecting digital assets [1].

Semantic annotations and ontologies can also help us to capture the precise specification of a security model in order to improve its prevention and reaction capabilities [33,38,75,43,55]. For example, intrusive behaviors can be systematically modeled to the required level of granularity. However the actual power and utility of this is determined by the expressiveness of the ontology in modeling attack scenarios in a generalized way and at a certain level of abstraction. The ontology models developed in the prior art mainly focused on lightweight representations of the attributes of the attacks in a taxonomic structure. These models seriously lack the necessary ontological modeling and subsequent reasoning capabilities. Moreover such systems focus on applications within the network layer or access control mechanisms for digital assets.

The method proposed in this paper for detecting and classifying web application attacks differs from current signature-based security solutions. We have developed an ontology based technique that specifies web application attacks using the

```
" 'OR 1=1'
" OR    1      =%091
" OR 'Password' = 'Password'
" OR 1%3D1
" OR 1 /* comments */ = 1
" OR  2 > 1
" OR 'Script' > 'R'
" OR 'Script' < 'Y'
" OR 'Script' = N' Script '
" OR 'Script' = 'Scri'+'pt'
" OR 'Script' LIKE 'Scr%'
" OR 'Script' IN ('Script')
" OR 'Script' BETWEEN 'R' AND 'T'
```

Fig. 1. Heterogeneity in a tautology attack using SQL Injection.

context of consequence, semantic rules and specifications of application protocols. The system is capable of detecting sophisticated attacks effectively and efficiently by analyzing the specified portion of user request where attacks are possible. Semantic rules can help the system to capture the context of the application, the likely attacks it may face and the protocol that was used. These rules are also utilized for reasoning by inference upon ontological models for detecting complex polymorphic variations of web application attacks. The proposed ontological model specifies web application attacks especially OWASP's most critical top 10 attacks, using the context of consequence, specification of protocol and semantic rules. These are structured as logical statements in the form of Horn logics [14].

Our approach semantically analyzes the HTTP [23] traffic (almost 60% of network stream [27]) but can be extended for other application protocols such as SMTP [53], HTTPS [57], FTP [54] and others. The system analyzes the specific portion of headers and payloads of the user requests. The ontological knowledge base of the proposed system is created using OWL-DL [46]. Ontology driven software systems are capable of demonstrating a shared understanding of structured information about the concepts within specific domain. They can provide reasoning capabilities and a greater ability to analyze information automatically. An ontology also allows the various semantic relationships among different concepts to be specified, thereby reducing interoperability issues, encouraging reuse, and allowing the system to evolve over time. The proposed system stores the concepts of different entities such as protocol, data, attack and relationships in the form of an ontology. This allow reasoning to be carried out which is designed to improve the detection ability of the system and also makes it more efficient and robust. The specific signature free rules are generated by capturing the context of the domain and the relationships between the entities. The data model of the proposed system has been implemented using the Resource Description Framework Schema (RDFS) [41] and OWL-DL. Pellet, an OWL-DL reasoner [70] has been used to facilitate the inference process. The Jena API [12] along with the SWRL (Semantic Web Rule Language) [35] has been used as development framework for reasoning. The knowledge model is validated using OntoClean [30] to remove inconsistency, incompleteness and redundancy in the specification of ontological concepts.

The paper makes the following contributions:

- In this paper, we propose a method of detecting and classifying web application attacks. In contrast to current signature-based security techniques, our solution is an ontology based approach that specifies web application attacks using the context of consequence, semantic rules and specifications of application protocols.
- The system is capable of detecting sophisticated attacks effectively and efficiently by analyzing the specified portion of user requests where attacks are possible. The semantic rules allow us to capture the context of the application, the attack and the protocol that was used. These rules are also utilized for reasoning by inference upon ontological models in order to detect complex polymorphic variations of web application attacks.
- Our semantic approach is capable of representing a shared understanding of structured information about the concepts within a specific domain, provides reasoning features and the ability to automatically analyze information. It also specifies semantic relationships between concepts, improves interoperability and encourages reuse and evolution over time.
- Most of the tradition security solutions such as ModSecurity [59] and Snort [60], lack an inference capability; that is why these systems depend upon attack signatures (rules) for each header and request body where parameters are present. Moreover they have to search sequentially to locate parameters that are present. This can mean looking at each header and request body. Thus they require more attack signatures and it often takes more time to search for an attack vector. Whereas our approach with its set of rules and inference capability overcomes these limitations. The mechanism uses inference rules and reduces the search space via a simultaneous search of the parameters from headers and the request body. It then applies a single semantic rule on these parameters for validation.
- We present examples and results which demonstrate that our semantic approach can be used to detect zero day and more sophisticated attacks in real-world programs.

This paper is organized as follows. Section two, includes some related work in the information security and semantic security domains. Section three, discusses the semantic approach to web application security, provides a brief overview of the protocol and explains the web attack ontology model. Section four, considers the architecture of the proposed system. In section five, the system evaluation and performance has been discussed followed by a conclusion and some directions that the work make take in the future.

2. Related work

A literature survey has been carried out in order to motivate a critical analysis of the various state of the art solutions for web application security. The characteristics of some of the best known techniques and their weaknesses are now discussed in the following section.

2.1. Anomaly based Intrusion Detection Systems (IDS)

Anomaly based intrusion detection systems analyze the behavior of the input stream against an established profile and classify all abnormal behavior as malicious [42]. Song et al. [71] proposed an anomaly detection technique that can automatically tune and optimize the values of parameters without predefining them through labeling. The detection ability of such systems however depends heavily upon the training of the data model. A poorly refined model reduces the detection performance. Anomaly based IDS are commonly unable to provide a fool proof solution to application level attacks because such models are also used to secure database servers against SQL Injection attacks. Pinzon et al. [52] has proposed a multi-agent hybrid architecture for detecting SQL Injection attacks. Chen et al. [13], put forward an unsupervised learning framework named the "community anomaly detection system" to detect insider threats based on the access logs of systems. Xu et al. [36] developed an anomaly management framework for firewalls by adopting a rule-based segmentation technique to identify policy anomalies and derive effective anomaly resolutions. Sheng et al. [69] has presented a behavior modeling approach that separates service features into operational and control behaviors for the automated verification of web services. Liu et al. [44] proposed a spam-detection framework for the detection of various kinds of Web spam, including novel ones, with the help of the user-behavior analysis. Fraiwan et al. [26] suggested a classification-based detection approach for the identification of web application attacks based on java script execution. Finally Rietta [58] concluded that application level IDS are effective for detecting novel attacks but are prone to high rates of false positives.

2.2. Signature based IDS

Signature based intrusion detection systems use the signatures of already known attacks or vulnerabilities and apply raw pattern matching algorithms [8] to detect security threats. Such detection mechanisms are applied at the both the network level, by analyzing network traffic [74] and at application level, by monitoring server logs [63]. Signatures of known attacks are often ineffective in preventing zero day attacks. Security systems can be easily defeated by the polymorphic nature of attacks. Due to an exponential increase in the variety of attacks, signature based IDS also face the challenge of a rapid increase in signature rules. Keeping the threat identity database up to date is a time consuming task. Snort, for instance, has more than 2500 signature rules [60]. Network IDS also have difficulties in working with encrypted communication on the web [83]. Many context based application IDS [3] face the problem of manually creating and updating signatures. Xu et al. [81] has presented an approach for the automated generation of security tests by using formal threat models to detect invalid inputs. Berghel [31] highlighted the issues related to identity theft and financial fraud vulnerabilities in banks, healthcare sectors and in government entities. Duan et al. [18] developed an effective spam zombie detection system that automatically detects compromised machines in a network by monitoring the outgoing messages from it. Vimercati et al. [76] provided a flexible solution for protecting the information accessible through Web applications by using an approach of credential-based access control and trust management. Shar and Tan [68] proposed a valid solution against Cross-Site Scripting vulnerability in web applications. Sun et al. [73] designed a user authentication protocol that leverages the short message service to thwart password stealing and reuse attacks. Antunes and Vieira [4] provided defense-in-depth approach to secure Web applications. Anomaly detection using a negative security model in a web application was tried by Auxilia and Tamilselvan [6] but it was found to only provide a partial solution and gives little protection against zero day attacks.

2.3. Data mining techniques or statistical IDS

Data mining techniques for anomaly detection [80] provide frameworks for the detection of web application attacks based on statistical methods. They often fail however to analyze malicious payloads by using the application's context. Some network based IDS [20,59] used data mining techniques, these methods consider character frequency and their occurrence probabilities in malicious data. Such approaches lack the semantics that are necessary to understand the contextual nature of an attack and its consequences. Balzarotti et al. [7] combined static and dynamic analysis techniques to identify weaknesses in the sanitization procedures of web applications which allow an attacker to bypass security controls. This analysis approach showed very limited effectiveness where custom sanitization procedures were put in place and typically generated many false positive results. Shahzad et al. [67] has proposed building a genetic footprint by mining the information in the kernel control blocks of a process to enable the detection of malicious processes at run time. In contrast to signature-based techniques of detecting malware, Santos et al. [64] has proposed a method that is based on the frequency of opcode sequences. This can be used for the detection of malware which other data-mining based approaches cannot detect. Amin et al. [2] introduced a new email-filtering technique based on email's persistent-threat and recipient-oriented features that outperforms traditional detection methods. A security risk analysis model has been proposed by Feng et al. [21], in order to identify the causal relationships among risk factors and analyzed the complexity and uncertainty of vulnerability propagation. Stein et al. [72] implemented the fuzzy versions of the Bayesian formulas to deal with the inconsistent information between the data and prior knowledge. Yao et al. [82] have described an anomaly detection methodology which uses a proximity graph and a PageRank algorithm. Static analysis has demonstrated that it can detect taint-style vulnerabilities in web applications [39]. It provides a high analysis speed whilst generating a low number of false positives but the scope of approach is limited in terms of web application attacks that were considered and the methodology used.

2.4. Ontology based IDS

Ontology-based IDS solutions are becoming increasingly used in information security. Raskin et al. [55] developed an ontology for the data integrity of web resources and advocated the use of ontologies for information security. They claim that by using an ontology, intrusive behavior can be systematically described with any level of detail that is necessary. Ontologies help in reducing the large diversity of concepts to a smaller list of properties. The precise specification of security know-how can also be helpful in improving prevention and reaction capabilities. Landwehr et al. [43] presented a categorized taxonomy of intrusion according to location, means and genesis. Ning et al. [48] considered a hierarchical model for the specifications of attacks through the examination of attack characteristics and attributes. McHugh [47] focused on the classification of attacks according to protocol layers and Guha and Mukherjee [32] considered the analysis of each layer of the TCP/IP protocol stack as the foundation for an attack taxonomy. Santos et al. [65] used an ontology for detecting spam messages by capturing the context of an e-mail message and its internal semantics. The major problem in the systems mentioned previously is that the ontology is just used to depict a simple representation of the attributes of the attack. These systems lack the reasoning ability that is necessary to intelligently protect a system. This is due to their taxonomical structure and focus on the network layer rather than application layer. Consequently they neglect some of the most critical web application attacks.

Denker et al. [16] based the control access on an ontology that was developed in DAML + OIL [34]. Ontologies have not been fully utilized because they are often limited simply to representing attack attributes. Undercoffer et al. [75] presented the concept of a target-centric ontology for intrusion detection and protection from network level attacks. Eastlake et al. [19] also presented an ontological model for intrusion detection during network layer attacks. From a taxonomy point of view, intrusion detection can have characters, classifications and languages that intelligently describe instances of a taxonomy and convey information regarding an attack or intrusion. The actual power and utility of the ontology is determined by the facts that it can express, the relationships between collected data and the use of such relationships to deduce particular data which represents an attack of a particular type [75].

Hung and Liu [38] suggested a new approach for designing and developing an intrusion detection application by using an ontology. The system expresses the intrusion detection in terms of the end users domain and allows a non-expert person to model the intrusion detection system easily by using the terminologies and concepts of intrusion detection. An Ontology of Information Security developed by Herzog et al. [33], models assets, threats, vulnerabilities, countermeasures and their relations. It also supports querying and acquisition of new knowledge through inference and rule-based reasoning using OWL Reasoner. It does not however address web application threats such as SQL Injection and XSS attacks. Fenz et al. [22] focus on the utilization of a security ontology that can support the ISO/IEC 27001 certification and maintenance of security guidelines or policies. This work does not though actually address any web application vulnerabilities. Compton et al. [15] proposed W3C Semantic Sensor Network ontology using OWL 2 to describe sensors and observations. Rowe et al. [61] presented a behavior ontology that captured the user behavior within a given context and inferred the role of a user by using semantic-rule based methodology. Park and Kang [51] proposed an automatic rule acquisition procedure using rule ontology and demonstrated that ontology-based rule acquisition approach works in a real-world application. Domingo-Ferrer et al. [17] presented a knowledge-based numerical mapping for nominal attributes that captures and quantifies their underlying semantics. Arogundade et al. [5] proposed an ontology for the formal representation of eliciting safety and security requirements. The ontological approach helped to avoid any ambiguity and inconsistency in capturing safety and security requirements. Eastlake and Undercoffer et al. [19,75] have provided better approaches by using an ontology to capture the domain knowledge of an application, but they carry some overhead due to lack of a viable search space reduction mechanism. Such solutions are commonly provided in a general form for network level attacks and ignore web application attacks.

3. Ontology models for web security

An ontology is the explicit specification of the conceptualization of a domain which captures its context. Ontological models are flexible in defining, nearly any concept to the desired level of detail. They are also fairly easy to extend and the logical foundations of modern ontology languages allows reasoning over concepts within a knowledge base. Reasoning over the instances of the data within the domain can be used to infer new knowledge and to integrate data. Moreover, ontological models can be easily shared, refined and reused between entities in a domain. A brief description of some important ontological models are described in subsequent sections.

3.1. Protocol ontology model

The protocol ontology model provides the foundation for developing an attack ontology model and is important in detecting attack scenarios. In this model, application layer communication protocols are modeled as semantic networks and as a conceptual category. The class Protocol, from conceptual modeling point of view; it is parent of all the specific application layer protocol classes such as HTTP, HTTPS, FTP, DNS, Telnet and SMTP. The model includes the additional concepts of Message, Request, and Response. Fig. 2 depicts the interconnection of various protocol concepts.

3.2. Attack ontology model

The attack model describes several important security aspects. It contains a number of concepts including: web application attacks; communication protocols used by the attack; encoding schemes that provide the wrapper to attack for decep-

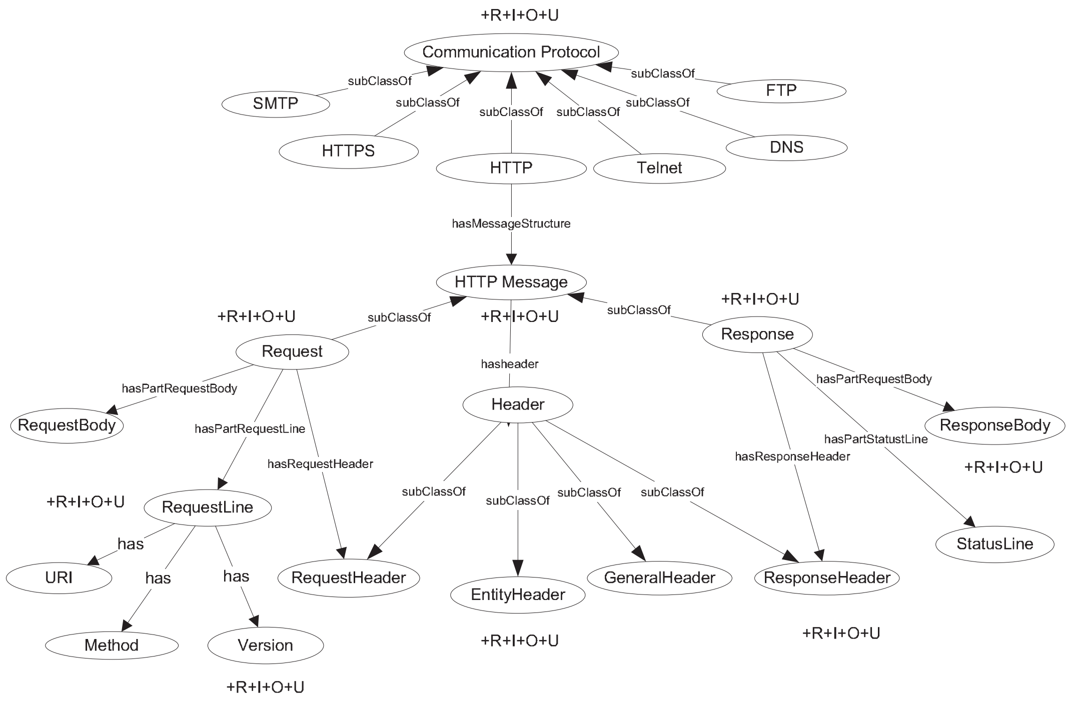

Fig. 2. Portion of protocol ontology validated through OntoClean.

tion purpose; vulnerabilities that are exploited by a malicious input; system components that are affected by these attacks; the consequences of each attack and control applied to mitigates such attacks. The model has the ability to represent security aspects with various levels of detail which range from the abstract to more specific concepts. The model can be reused and easily extended over time as web application attacks continue to evolve. The main classes of the ontological model include: the Attack Class which has the properties, hasCommunication, deceivedBy, causedBy, directedTo, resultingIn, and mitigatedBy, which are defined by classes Protocol, Encoding Scheme, Malicious Input, System Component, Consequence, and Control respectively. The Vulnerability Class, models the features which are exploited by malicious input and has been further subclassified. The Control Class has the subclasses Rule and Protocol Specification Rule, those are also further subclassified, as is shown in Fig. 3.

The Rule Class has the properties, hasCondition, executedIn, triggers and hasMetadata defined by the Classes, Condition, Phase, Action and RuleMetaData, respectively. The Condition Class has a specific value which is represented in the form of a regular expression, that matches with malicious portions of a user's request and as result, an action is triggered by a rule. A Phase Class indicates the stages in which the rule is to be executed. Phases have various stages, i.e. request headers, request body, response headers, response body and logging. The Action Class has a range of values including: discard the request, generate warning alert, log the current request, ignore the action, or allow the message through. A RuleMetaData Class has been created and has properties which include ruleID, revisionNumber, severityLevel and the message to be displayed. The logical construct about each attack can be expressed as: the Tautology attack is a subclass of the Attack (SQL Injection) that exploits the SQL Injection vulnerability in the application. This attack uses the communication Protocol (HTTP), is deceived by the Encoding Scheme (UTF-16) and effects the System Component (Database). The attack is caused by malicious input (Tautology Condition), resulting in consequence (Authentication by pass), and Mitigated by the Control (Ruleid-1234). This logical construct can be expressed in the OWL-DL as is shown in Eq. 1 and 2.

$$\begin{aligned} & \mathit{SQLInjectionAttack} \quad \sqsubseteq \mathit{Attack} \sqcap \exists \mathit{hasCommunication}.\mathit{Protocol}(\mathit{HTTP}) \\ & \sqcap \exists \mathit{deceivedBy}.\mathit{EncodingScheme}(\mathit{UTF}16) \\ & \sqcap \forall \mathit{effects}.\mathit{SystemComponent}(\mathit{Database}) \\ & \sqcap \exists \mathit{resultingIn}.\mathit{Consequence}(\mathit{Authenticationbypass}) \end{aligned} \tag{1}$$

$$\begin{aligned} & \mathit{TautologyAttack} \sqsubseteq \quad \mathit{SQLInjectionAttack} \sqcap \forall \mathit{causedBy}.\mathit{MaliciousInput}(\mathit{TautologyCondition}) \\ & \sqcap \forall \mathit{exploits}.\mathit{Vulnerability}(\mathit{SQLInjection}) \sqcap \exists \mathit{mitigatedBy}.\mathit{Control}(\mathit{Rid}4321) \end{aligned} \tag{2}$$

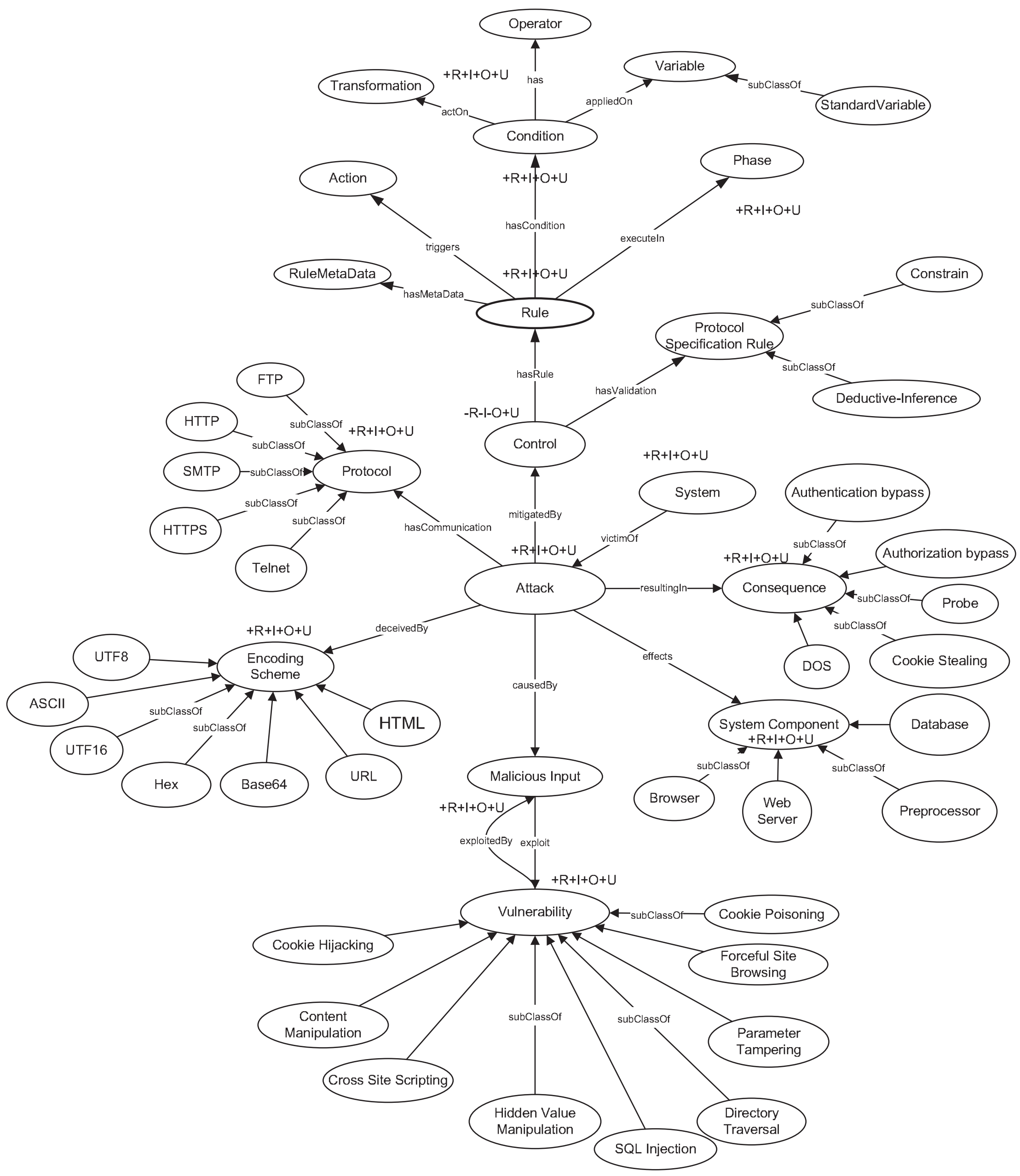

Fig. 3. A Portion of the attack ontology validated through OntoClean.

The verification and validation of the various model components has been carried out using the OntoClean methodology [30]. This technique formally analyzes the ontological models by providing a logical basis for arguing against some of the most common modeling pitfalls. It allows us to discover common misuses of subsumption and assists us in validating models by exposing any inconsistencies that they might contain. Models are validated against some Meta properties which include: rigidity (rigid (+R), anti-rigid (R), non-rigid (−R)), identity criterion (+I), not caring identity (−I), unity criterion (+U), anti-unity (U) and supplies identity criterion (+O). Our protocol and attack ontology models were both validated using OntoClean.

3.3. Protocol specification rules

The protocol specification rules are basically constraints on the protocol ontology. These rules are embedded in the model as constraints or modeled by using SWRL (Semantic Web Rule Language) in the protocol ontology, we call them constraint rules. An example of a protocol specification rule for HTTP messages is shown in Fig. 4. The protocol specification rules are as follows: The rule, named SecRule001, specifies that a HTTP request which has a GET method and also has a Request-Body will be treated as malicious. SecRule002 describes that a HTTP request which has a POST method but is without a Request-Body, should also be treated as a threat. SecRule003 shows that a HTTP request with a DELETE method and a Request-Body, will be treated as potentially damaging. SecRule004 asserts that a HTTP request with a HEAD method and a Request-Body, will be treated as a malicious request. The following rule, SecRule005 states that a HTTP request with a GET method and an Entity header, will be treated as a threat. Finally SecRule006 describes that if a HTTP request does not have exactly one Request Line, then it will be treated as malicious.

3.4. Rule generation mechanism

The Control Class handles the detection and mitigating of web application attacks by using the semantic and protocol specification rules. In terms of semantic rules, the detection mechanism semantically analyzes user requests by using the inferred knowledge that is generated as a result of inference over the ontological model. The inference process drives the detection rules and uses a customized Horn-logic grammar for parsing them. Semantic rule generation briefly includes the following steps: (1) Ontological model is obtained by inference the ontology stored in RDF-tuples. (2) Inferred knowledge is generated on the basis of specific rule template and rule grammar. (3) Detection rules are generated through querying the inferred knowledge and stored in a cache. A query consists of a triple (3-tuple) pattern and is used to retrieve results from a database of semantic rules and ontology models. In many cases, new triples are generated by the inference process and qualify to be considered in the result-set. Such a capability encourages more meaningful results which are therefore more likely to satisfy the user's requirements. This feature is provided by effectively using semantic knowledge in the form of an ontology.

The rule template acts as an abstract description of the attack scenario as shown in Fig. 5. This template describes the basic information flow of a user request from an external network, to web servers using the specified protocol and port. It displays the basic attack information which includes "msg", an "effects" variable that represents the system component that is affected by this attack and a "content" variable which highlights the malicious portion of the request. The "consequence" variable describes the consequence of the attack and a "condition" variable stores a regular expression that matches the malicious portion of the request as the result of some "action" that is triggered by a rule. The rule template also describes some meta data properties, i.e. rule ID, phase, revision number and severity level of the attack. In order to detect a specific attack situation the template has to be instantiated in accordance with the inferred knowledge about attack and mitigation mechanisms. Each rule instance specifies the conditions under which malicious activity is detected and the subsequent action to be taken. More specifically each rule contains a condition with some value for detecting a specific attack from a portion of the user request and the corresponding mitigation action.

Fig. 6 shows instances of rules for detecting Cross Site Scripting and SQL Injection attacks. Both instances describe the basic information flow of the user's request from external network to HTTP servers using the HTTP protocol and port 80. Firstly the system displays some msg information about the attack, such as "Cross site scripting HTML Image tag set to java script attempt". The effects variable represents that "client browser" is affected by this attack and content variable highlights the malicious portion of the request within the "SCRIPT tags". The consequence variable details that the consequence of the attack is "accessing sensitive information or identity theft". The condition variable has a "regular expression" that matches with the malicious portion of the user request and as a result the action "discard" is triggered by a rule. The Rule ID is 1234,

```
[SecRule001:(?r rdf:type ex:HTTPRequest)(?m rdf:type ex:HTTPMethod)(?r ex:hasMethod ?m)
(?m ex:hasMethodName "GET")(?b rdf:type ex:Request_Body)(?r ex:hasBody ?b)
 --> (?r rdf:type ex:MaliciousRequest)]
[SecRule 002: (?r rdf:type ex:HTTPRequest) (?m rdf:type ex:HTTPMethod) (?r ex:hasMethod ?m)
(?m ex:hasMethodName "POST")(?b rdf:type ex:Request_Body) !(?r ex:hasBody ?b)
--> (?r rdf:type ex:MaliciousRequest)]
[SecRule 003: (?r rdf:type ex:HTTPRequest)(?m rdf:type ex:HTTPMethod)(?r ex:hasMethod ?m)
(?m ex:hasMethodName"DELETE")(?b rdf:type ex:Request-Body)(?r ex:hasBody ?b)
--> (?r rdf:type ex:MaliciousRequest)]
[SecRule 004: (?r rdf:type ex:HTTPRequest)(?m rdf:type ex:HTTPMethod)(?r ex:hasMethod ?m)
(?m ex:hasMethodName"HEAD")(?b rdf:type ex:Request-Body)(?r ex:hasBody ?b)
--> (?r rdf:type ex:MaliciousRequest)]
[SecRule 005: (?r rdf:type ex:HTTPRequest)(?m rdf:type ex:HTTPMethod)(?r ex:hasMethod ?m)
(?m ex:hasMethodName "GET")(?e rdf:type ex:EntityHeader)(?r ex:containEntityHeader ?e)
--> (?r rdf:type ex:MaliciousRequest)]
[SecRule 006: (?r rdf:type ex:Request) (?l rdf:type ex:RequestLine)
(?r ex: \exist!=1 hasRequestLine ?l) ) -->(? r rdf: type ex: MaliciousRequest)]
```

Fig. 4. Protocol specification rules.

```
[alert tcp $EXTERNAL_NET any -> $communication protocol _SERVERS $ communication protocol _PORTS{
msg:" string of information about attack";
effects: system components;
content: "string of malicious input";
consequences: consequences of attack;
condition: "Regular expression";
control: detection rule with ids; phase: phase name;
severity level: critical;
action: rule trigger some action; rev: revision number}]
```

Fig. 5. Rule template as abstract description of the attack scenarios.

```
[alert tcp $EXTERNAL_NET any -> $HTTP_SERVERS $HTTP_PORTS{
msg:"Cross site scripting HTML Image tag set to java script attempt";
effects: Web server (client browser);
content:"<SCRIPT SRC=http://ha.ckers.org/xss.js></SCRIPT>";
consequences: accessing sensitive information| identity theft;
condition: "\w*((%3C)|(\<)|[a-z0-9\%]+(\%3E)|(\>)";
control: Rid-1234; phase: (request header | request body| logging);
severity level: critical;
action: (discard | log); rev:4}]

[alert tcp $EXTERNAL_NET any -> $HTTP_SERVERS $HTTP_PORTS {
msg:"SQL Injection attempt";
effects: Web server (database);
content: " ‘or 1=1--";
consequences: Authentication by pass;
condition: "\w*(\‘)|(\s((%6F)|o|(%4F))((%72)|r|(%52)))|((\%3D)|(=))[^\n]*((\%27)|(\‘))|((\-\-)";
control: Rid-4321; phase: (request header | request body| logging);
severity level: critical;
action: (discard | log) ; rev:3}]
```

Fig. 6. Instances of rules for attack detection.

```
[rule1:  (?a ex:hasPart ?b)  (?b ex:hasPart ?c) ->(?a ex:hasPart ?c) ]
[rule2:  (?a ex:hasPart ?b) (?b ex:contains ?c) ->(?a ex:contains ?c)]
[rule3: (?r rdf:type ex:HTTPRequest)(?m rdf:type ex:MaliciousPortion)(?m ex:infects ?p)
(?r ex:immediatePart ?x)(?y ex:subClassOf ?x)(?y ex:contains ?p) -> (?m ex:infects ?y)]
```

Fig. 7. Deductive inference rules.

the valid phases are "request header, request body or logging". The revision number of the rule is 4 and the severity level of the attack is critical. Similarly the second example provides complete information about the SQL Injection attack.

3.5. Inference process

The role of the inference process is to reason about the hidden information in the knowledge base and to create new assertions from it. A portion of the HTTP ontology has been used in order to understand the inference process. This was carried out with the help of simple Horn-style semantic rules. A brief description of an interpretation of the rules are shown in Fig. 8. The deductive inference rules shown in Fig. 7, operate on a portion of the HTTP ontology as depicted in Fig. 8 and derives new assertions from it. Rule one states that the HTTP Request has a part called Request Line and that the Request Line has a part named Query String. This implies (through the transitive property) that HTTP Request has part Query String. Similarly Rule two states that HTTP Request has a part called Query String and that Query String contains Parameters. This implies (through the transitive property) that a HTTP Request contains Parameters. Rule three states that "r" is the HTTP Request type and "m" is specific Malicious Portion of attack vector and that "m" infects the Parameters. The Immediate part of "r" is the Request Line, Request Body and Request Header. Whereas, the Cookie and the Referrer headers are the sub classes of the Request Header which contains Parameters. As the Malicious Portion "m" infects Parameters this implies (through the inference process), that "m" also infects the Cookie and the Referrer header.

Most traditional security solutions such as ModSecurity and Snort, lack an inference capability this is why they rely so heavily on applying attack signatures[1] (rules) to each parameter that is present. Moreover these solutions search sequentially in order to locate parameters in almost each header and request body. This results in processing more attack signatures and as a

[1] A static text expression used to represent the pattern of malicious data generated by known web application attacks.

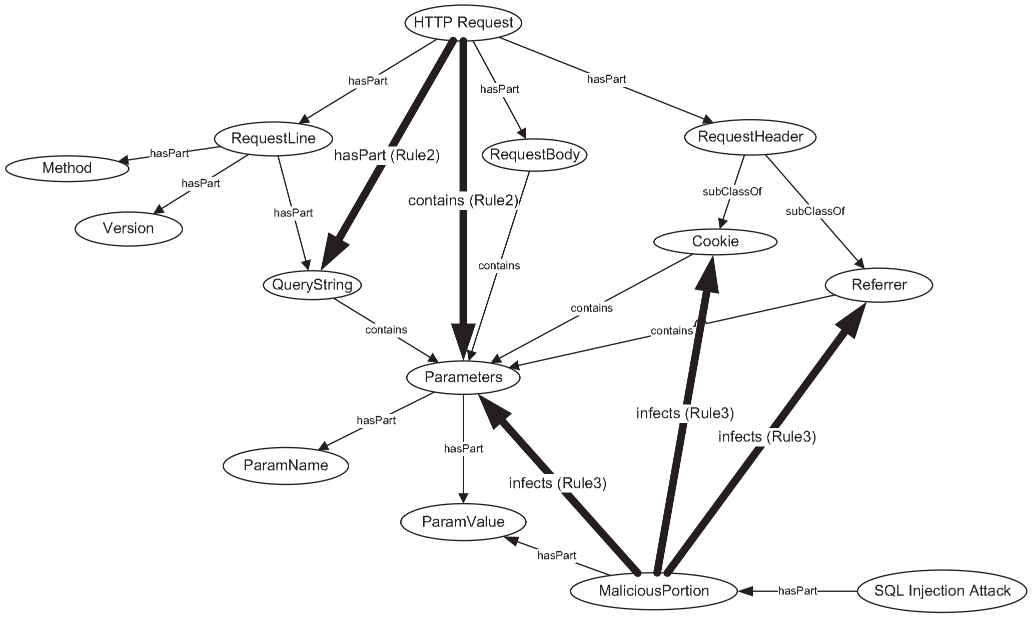

Fig. 8. Inference process and derivation of new assertions.

consequence the performance of the system may be compromised. Through the inference capability, these current limitations can be overcome. The inference mechanism helps to reduce the search space through a simultaneous search of parameters from headers and request bodies and it can then apply a single rule on these parameters for validation.

4. Semantic approach to web application security

The semantic approach to web security uses a defense in depth strategy by providing detection mechanisms at multiple layers. The first layer of defense uses the ontological model of the communication protocol and provides protection against protocol specification related attacks. The second layer normalizes the user requests by decoding them in a standardised format which helps to mitigate encoding based attacks. The third layer uses an ontological model of well known attacks via a negative security model and the fourth layer harnesses the knowledge of application profiles through a positive security model. The rule generation mechanism applies inference[2] upon these models and generates semantic rules[3] for the detection of web application attacks. A description of the attack detection process is presented in the following subsections:

4.1. Process of attack detection

The multi-layer defense approach is intended to handle the validation and sanitization of user requests and responses. In the initial phase, all user requests are parsed in a standardized message format, according to the information (ontological models) and validation specifications of the communication protocol. Protocol specifications ensure that the user request must contain all of the mandatory headers (such as Host header) and that the value of each header should be in the correct format (for example, the Content-Length must be numeric.) Similarly the referrer field and cookie headers are analyzed against valid URL and cookie values respectively according to the protocol ontological model. If the request follows the protocol specification, then it is further analyzed, otherwise it is discarded. At this stage most of the requests that violate the specification (invalid requests/ protocol based attack vectors) are filtered out, thus saving a large amount of processing time. Valid requests are normalized by decoding the request into a standardised format. The normalization process helps to reduce the scope for encoded attacks and translates encoded attack vectors into normal strings. For example, if user contains the encoded string in the form, "%3Cscript%3E%20alert (%22 This %20 is%20 cross%20 site% 20 script%22) %20%3C%2Fscript%3E". Normalization process will decode the string into,

```
"<script> alert (‘This is cross site script’) </script>"
```

[2] The process of deriving the logical consequences of assumed ontological assertions or the process of arriving at some conclusion that, though is not logically derivable from the assumed assertion, holds some degree of truth relative to the assertions.

[3] Logical statements which reuse concepts defined in the ontology models.

The normalization process enhances threat detection by providing the decoded data for further analysis. Each normalized request is then analyzed in greater detail using the positive and negative security models. If the request is valid, it passes to the web application, otherwise it is discarded. Similarly, responses from user requests are also sanitized (the process of eliminating dangerous constructs/important information) to avoid information leakage attacks.

4.2. Example of attack detection using semantic rules

In order describe in greater detail how the semantic rule based detection mechanism works, a couple of example threats have been selected. The first is a Web cache poisoning attack and the second is an HTTP response splitting threat. These will be discussed in the following paragraphs. The semantic rule that is necessary is shown in Fig. 9. This rule imposes the condition that if a request does not possess exactly one start line then the request is malicious. We will now consider how this can be applied against two example attacks.

4.2.1. Web cache poisoning attack

The attacker targets the cache service which is used by an organization to reduce the load on internet bandwidth. This server can be a cache server on the LAN or other application server caching the static Web pages. The attacker sends three different HTTP requests embedded in single request as shown in Fig. 10.

The attacker sends three different HTTP requests.

Request 1: POST http://www.example.com HTTP/1.1.

Request 2: GET http://www.example.com/maliciouspage.html HTTP/1.1.

Request 3: GET http://www.example.com/pagetoattack.html HTTP/1.1.

The cache server concludes that requests 1 and 3 are valid and the Web server which strictly follows the HTTP parsing rules, considers request 1 and 2 to be correct. The cache server therefore stores this response against request 3. When a normal user makes a request, the cache server responds with the wrong page (this is cache poisoning).

4.2.2. HTTP response splitting attack

In a HTTP Response splitting attack, attackers craft malicious requests that also contain a malicious response within it. This forces the web server to generate two responses, one for normal requests and the other as the attacker desires. In this attack the server embeds some malicious script within the user request by using HTTP response headers. In the following example that shown in Fig. 11, the HTTP response splitting attack vector is a composition of HTTP request along with a complete HTTP response.

Many Web application solutions have created signatures for these kinds of attacks. ModSecurity for example, has lengthy signatures for request smuggling, cache poising, credential hijacking and response splitting attacks. Our single rule, as shown in Fig. 9, can detect these attacks effectively and efficiently. In order to evaluate and explain the complete attack detection and rule generation mechanism, a system architecture has been designed that shows the interaction of the various components to achieve the desired functionalities.

4.3. Semantic based Security System

The proposed security mechanism can be deployed as a web proxy which protects servers hosting multiple web applications. All user requests are analyzed by the system before they are delivered to the particular web application. Every user request to the back-end server is also inspected and validated. For each user request a separate session is created and then monitored. The response of the user request is also sanitized by intercepting it from the back-end server and forwarding it to the client. The message is only forwarded if does not contain an error message which exposes information about the web server, application server, or the database being used.

The system architecture is composed of three main components: an Initial Validation Module, a System Analyzer, and a Detection Rule Generator. The System Analyzer is the core component which semantically analyzes the user request by applying knowledge of Positive and Negative Security Models. It also uses semantic rules that are generated through an inference process which runs over the Knowledge base module. The Knowledge base is a repository holding concepts describing different web attacks, communication protocol and application profiles in the form of ontological models (triple (3-tuple) patterns). Ontology models can be refined and expanded over time, according to the requirements of web applications. Each module is further subdivided into sub modules as shown in the high-level system architecture in Fig. 12. The individual components are covered in greater detail in following subsections.

4.3.1. Initial validation

The main role of this component is to filter out user requests that violate the protocol specifications. If a user request contains malicious data (for example, part of the user request contains harmful or unauthorized contents which can possibly have a negative effect on the web application or the application server), the request will be denied and recorded for further processing. This component is further divided into the following sub-components.

4.3.1.1. Request interceptor. This takes the user request from the network stream and sends it to the Parser.

```
[SecRule 006: (?r rdf:type ex:Request) (?l rdf:type ex:RequestLine)
(?r ex: \exist!=1 hasRequestLine ?l) ) -->(? r rdf: type ex: MaliciousRequest)]
```

Fig. 9. Protocol specification based semantic rule for attack detection.

```
POST http://www.example.com HTTP/1.1
Host: example.com
Connection: Keep-Alive
Content-Type: application/x-www-form-urlencoded
Content-Length: 0
Content-Length: 44
[CRLF]
GET http://www.example.com/maliciouspage.html HTTP/1.1
Host: example.com
Nothing: [space after the " Nothing:", but no CRLF]
GET http://www.example.com/pagetoattack.html HTTP/1.1
Host: example.com
Connection: Keep-Alive
[CRLF]
```

Fig. 10. Web cache poisoning attack.

```
http://www.example.com/redirpath.jsp?page=?foobar%0d%0aContent-
Length:%200%0d%0a%0d%0aHTTP/1.1%20200%20OK%0d%0aContent-
Type:%20text/html%0d%0aContent-
Length:%2019%0d%0a%0d%0a<html>Hacked</html>
```

Fig. 11. HTTP response splitting attack.

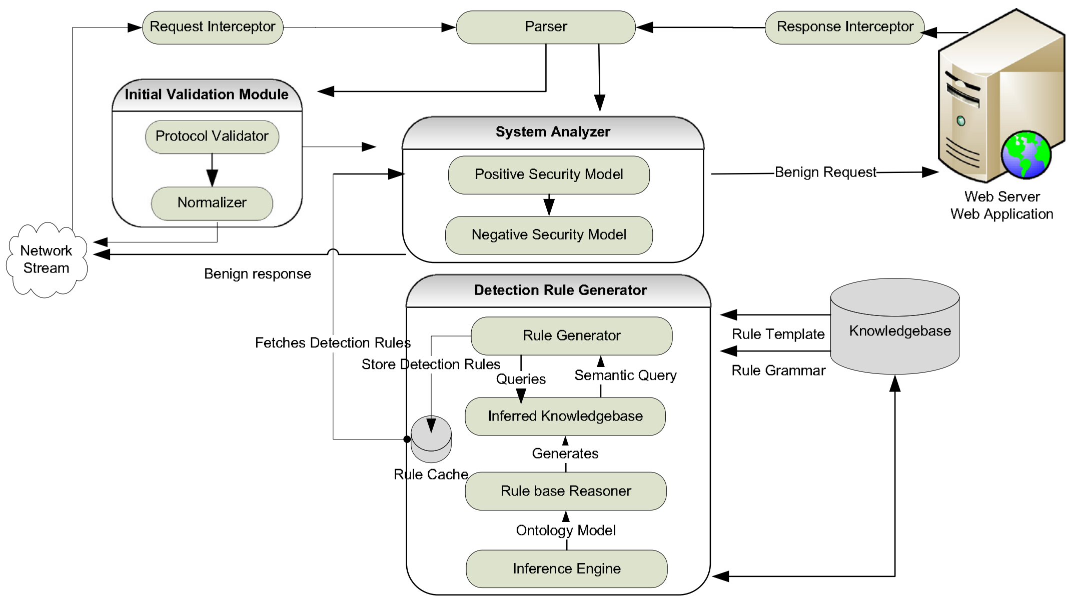

Fig. 12. The semantic base security system.

4.3.1.2. Response interceptor. The Response Interceptor takes the response of a user request from the web server and gives it to the Parser.

4.3.1.3. Parser. This component parses the user request in a standardized message format, according to the information (ontological description) that is saved in the knowledge base and sends the user request object to the Protocol Validator. It also takes the response of to a user request from the web server and passes it to System Analyzer.

4.3.1.4. Protocol Validator. This sub-component ensures that user requests and responses comply with RFCs. It applies the protocol specification rules to ensure that user requests contain all the required headers and that the value of each header is correct. For instance, the value for Content-Length must be numeric. Similarly, the referrer field and cookie header are compared against valid URL and cookie values respectively using the protocol ontology model. This module validates protocol compliance, mandatory headers such as the host header, headers fields, encoding schemes, methods used, length checking for URIs, query strings, POST data, object types (file types) and acceptable response codes. The user request is then passed onto the Normalizer.

4.3.1.5. Normalizer. This sub-component helps to normalize user requests by translating it into a standardized format, i.e. the encoded attack vectors are turned into a normal string. The normalized request is then sent to the System Analyzer for further analysis. This component also facilitates the Analyzer by providing decoded data thus increasing the efficiency and the detection ability of the system.

4.3.2. System Analyzer

The main task of the System Analyzer is to request analysis. It retrieves the detection rules from the cache and applies these on user requests using the positive and negative security models. These enable the system to detect anomalies in requests. If a request is benign it is passed to the web application otherwise it is rejected.

4.3.2.1. Positive security model. A Positive Security model stores the profile of an application. This includes the set of metadata elements, policies and guidelines that define a particular application. It details the characteristics of inputs to web application forms, specific resource authorizations and a list of functions that are permitted. A "positive" security model identifies scenarios with a known degree of trust, allowing access only to trusted resources. It is also known as white list, a list of accepted items which are allowed and rejected otherwise. The benefit of using a positive model is that new attacks, not originally anticipated by the developer, may be more easily prevented. Positive security is often managed via web application forms that provide a user interface through which they can interact with the system. Securing the form input field values can significantly reduce the attack space. Positive security rules can be generated by ensuring that only valid user input can be given to a form. This approach is often effective and more efficient because it leads to an early detection of a possible attack. This saves processing time in comparison with negative models, which need to scan a long list of terms before they can process a request.

4.3.2.2. Negative security model. This model contains the signature of known web application attacks. In contrast to the positive security model this scheme defines what is not allowed (otherwise known as a blacklist) and implicitly allows everything else. The negative security model monitors requests for anomalies, unusual behavior, and common web application attacks. This model provides a web application with a rule set (signatures), to ensure critical protection against web application attacks.

4.3.3. Detection Rule Generator

The Detection Rule Generator component comprises the following components:

4.3.3.1. Rule Generator. The Rule Generator automatically creates rule instances for web application attacks using ontological models. The rule generation mechanism can be summarized in the following steps:

- Rule base reasoning is managed by the inference engine using the ontology model, that is saved in the Knowledge base.
- The inferred knowledge model produced by the inference engine is queried by the rule generator in order to produce detection rules.
- A Semantic Query[4] is generated by the query builder using a rule template.
- The rule generator passes the semantic query onto the inferred knowledge model and populates the rule template.
- Rules are uniquely stored in the rule cache.
- Detection rules are taken by the System Analyzer from the rule cache.
- The System Analyzer examines incoming user requests and the outgoing responses to them.

In order to improve the performance of the system, rule instances are only created once during the life cycle of the system. The rule cache is populated with rule instances during the system start up phase. The system can efficiently handle multiple user requests by applying the rules that are available in cache and this takes minimal processing time. Generating and applying a new rule each time would badly affect the performance of the system and this is why it has been avoided in our design. New rules are produced, or the existing instances are re-generated if there is a change in one of the ontology models. This process is performed in an asynchronous thread to further improve the system efficiency, as the attack detection components are given a larger share of the computational and storage resources.

4.3.3.2. Rule based reasoner and inferred knowledgebase. The task of reasoning is to check the internal consistency of the knowledgebase, the correctness of data instances and the assertions that are present. During the reasoning process, new facts

[4] It consists of triple (3-tuple) patterns and is used to retrieve results from a database of semantic rules and ontology models.

are deduced and implicitly derived from the existing knowledgebase. Reasoning can be classified into logic based context reasoning, rule-based reasoning, deductive and inductive reasoning. The Rule based Reasoner is a customized generic reasoner that supports user defined rules. Forward chaining, backward chaining and hybrid execution strategies of reasoning are supported. These reasoning methods can be applied to web application system for deducing new knowledge and enhancing existing knowledgebase of system.

One of the advantages of using an ontology is that it provides a reasoning capability. It has the constructs that are necessary to enable software systems to reason over the knowledge base. New knowledge (inferred knowledge) is also created as a result of implicit derivation of the ontological statements and assertions (such as restrictions and truth conditions). The inferred facts are derived from the ontology model which in turn is based on concepts and their relationships.

4.3.3.3. Inference engine. The inference engine can be thought of as the "brain" of the system. It reasons about attack information in the knowledgebase and creates new assertions based on it. It also manages the execution control strategy in order to ensure that the most appropriate rules are acted upon when certain conditions are met regarding the user request. The inference engine, processes the Knowledgebase, extracts the required data and passes it to the rule base reasoner using the Jena API. It also classifies hierarchies and infers relationships between concepts which are not explicitly mentioned in the domain ontology. For example, the referrer field which is the part of the header of a HTTP request, is not normally checked for malicious scripts. The system's inference engine will however cause this field to be checked if it is confirmed as being necessary by traversing relationships in the ontology. This mechanism clearly shows that the system intelligently analyzes user requests for possible attacks.

4.3.3.4. Knowledgebase. The knowledgebase is an ontology store which holds a repository of attacks, a protocol ontology model, a set of semantic rules and a group of application profiles.

5. Evaluation

A defense in depth strategy has been implemented by applying a multi-layered detection and prevention mechanism in the proposed system. All malicious requests related to protocol validation are detected in the initial validation module, the remaining threats are detected in the analysis phase using the positive and negative security models. In this way the system reduces the processing overhead, as the further analysis of requests is unnecessary if a threat is detected in initial phase. Several features enhance the efficiency and effectiveness of the system. The system designed in such a way that it only needs to analyze the portions of a user request in which an attack is possible. This avoids the need for a full sequential search and reduces the overall processing time. The encoding scheme ontology also helps to handle the polymorphic nature of novel attacks.

Semantic rules provide attack metadata such as the level of severity of the attack, its effect on system throughput and efficiency, and the contents of the alert message. Alert generation is modeled in the action part of the model and is processed by the analysis engine. The performance of the system is improved by only generating rule instances once during the life cycle of the system. The rule cache is populated with rule instances during the system loading phase. New rules are generated or the existing instances are re-generated if there is a change in the underlying ontology models. This process is performed in an asynchronous thread to further improve the system efficiency, as attack detection components are given a larger share of the computational and storage resources.

5.1. System setup environment and hardware specification

In order to evaluate the system three machines have been used to construction a prototype environment. Web applications were hosted on an IIS 7.0 server [78]. The other servers hosted a firewall (including the proposed system) and a vulnerability scanner machine. The PHP based Joomla [11] content management system was selected as a test application for use during the evaluation phase. The hardware specifications of the servers are shown in Table 1.

5.1.1. Testing criteria

The system was evaluated through comparison with other state of the art solutions on the basis of detection rate, false alarm rate, features and performance. The evaluation of the system was performed on standard vulnerable web application "Web Goat" developed by OWASP for testing and evaluation of web application attacks. Attack vectors were collected from various security cheat sheets, including the web application security testing cheat sheet maintained by OWASP [49]. OWASP's WebScarab tool was used for the launching and analysis of web application attacks. Our system is compared with two state of the art security solutions including ModSecurity (open source) and Profense (trial version).

We reused the criteria which was set out by Sarasamma [66] for calculating the detection rate and false alarm rate:

- False Positive = FP: the total number of records that are classified as anomalous.
- False Negative = FN: the total number of anomalous records that are classified as normal.
- Total Noramal records = TN: the total number of normal records.

Table 1
Hardware specification of the systems in benchmarking environment.

System	Application	Hardware Specification
IIS 7.0 Server (Application hosting)	Web application	RAM – 2 GB, Processor – 3.0 GHz Intel Core 2 Duo
Firewall Machine – Proposed system	Linux Centos 5.3	RAM – 1 GB, Processor – 3.0 GHz
Firewall Machine – ModSecurity	Windows XP with service pack 2	RAM – 1 GB, Processor – 3.0 GHz
Firewall Machine – Profense	Windows XP with service pack 2	RAM – 1 GB, Processor – 3.0 GHz
Scanner Tool Machine	Paros Proxy	RAM – 2 GB, Processor – 3.0 GHz Intel Core 2 Duo

- Total Attack vectors = TA: the total number of attack records.
- Detection Rate = [(TA-FN)/TA] $*$ 100
- False Alarm Rate = [FP/TN] $*$ 100

The detection rate of the proposed system is shown in Fig. 13. A comparison of this result against ModSecurity and Profense is shown in Table 2. The system was also compared with other state of the art technologies on the basis of features present as depicted in Table 3. In doing this we made use of the features list that was developed by Brown et al. [9].

5.1.1.1. System performance. The overall performance of the system was tested using JMeter. Empirical results show that the maximum throughput and response time of the system is 1400 hits/s and 375 ms respectively. This compares well against

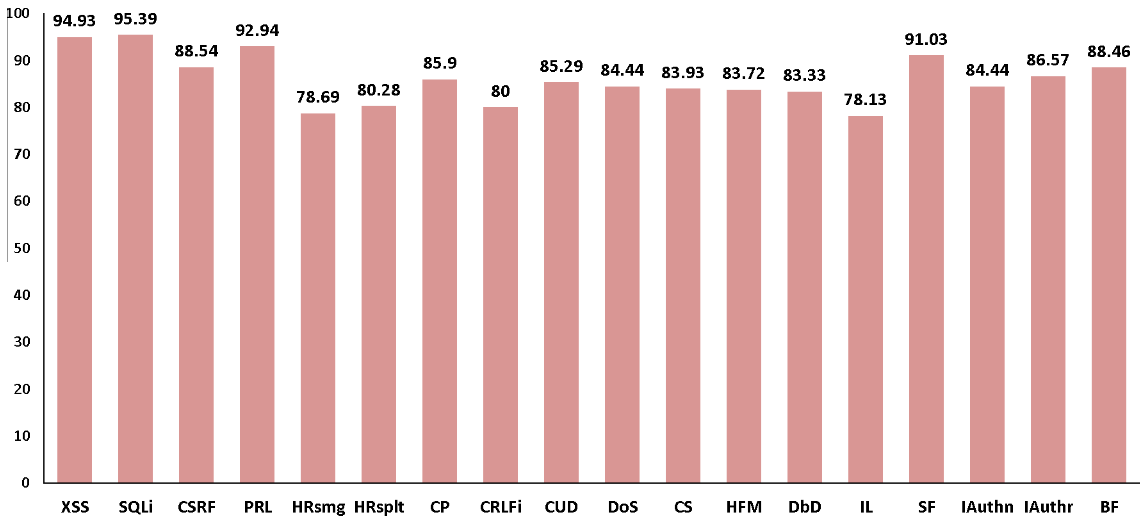

Fig. 13. Detection rate of proposed system.

Table 2
Detection Rate Comparison of Proposed System with ModSecurity and Profense in %.

Web Application Attacks	Proposed System	ModSecurity	Profense
Cross Site Scripting (XSS)	94.93	92.23	78.38
SQL Injection (SQLi)	95.39	79.95	71.00
Cross site Request Forgery (CSRF)	88.54	67.71	51.04
Predictable Resource Location (PRL)	92.94	85.88	47.06
HTTP Request smuggling (HRsmg)	78.69	49.18	78.69
HTTP Response splitting (HRsplt)	80.28	59.15	63.38
Cache Poisoning (CP)	85.90	73.08	58.97
CRLF Injection (CRLFi)	80.00	31.11	51.11
Cross User Defacement (CUD)	85.29	64.71	67.65
Denial of Services (Dos)	84.44	75.56	55.56
Content Spoofing (CS)	83.93	60.71	58.93
Hidden Field Manipulation (HFM)	83.72	79.07	51.16
Drive by Download-Gumbler (DbD)	83.33	11.11	16.67
Information Leakage (IL)	78.13	59.38	65.63
Session Fixation (SF)	91.03	67.95	64.10
Insufficient Authentication (IAuthn)	84.44	75.56	75.56
Insufficient Authorization (IAuthr)	86.57	65.67	65.67
Brute Force (BF)	88.46	61.54	57.69

Table 3
Feature based comparison of the system with state of art solutions:[(Proposed System:Sys), (Barracuda:Bar), (Secure Sphere:SSp), (F5 Networks:F5N), (Breach Security:BSe), (Profense:Pro), (ModSecurity:Mod)].

Appliance	Sys	Bar	SSp	F5N	BSe	Pro	Mod
Load balancing		✓		✓		✓	
Traffic shaping				✓			
High availability	✓	✓	✓	✓	✓	✓	✓
SSL acceleration and offloading	✓	✓		✓			
Connection pooling	✓	✓					
Cache and compression		✓				✓	✓
Pre-loaded policies and signatures	✓	✓	✓	✓	✓	✓	✓
Semantic support	✓						
Automated rule generation	✓						

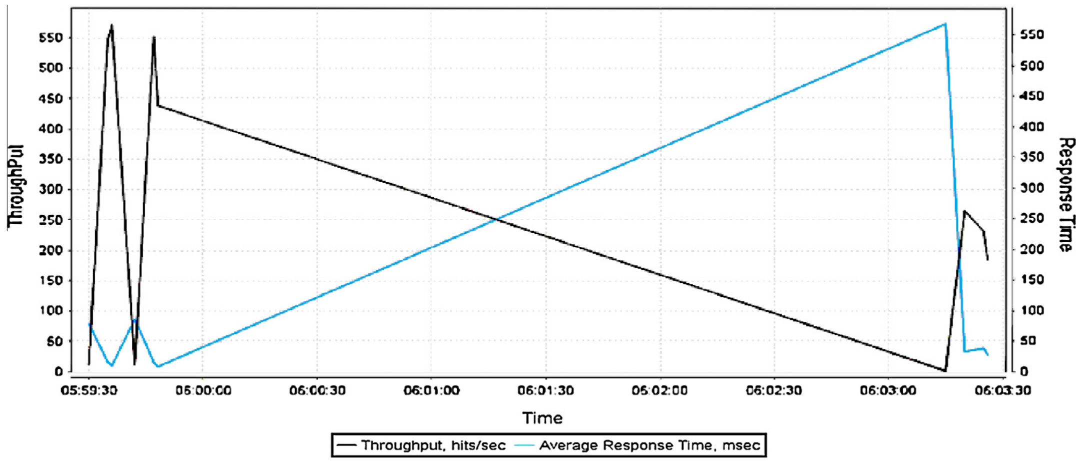

Fig. 14. Max throughput and response time of ModSecurity.

ModSecurity which has a maximum throughput of 560 hit/s and a response time of 555 ms on our benchmark environment. The results are depicted in Figs. 14 and 15.

5.2. Test cases

The test cases which are used for the SQL Injection and Cross site scripting attacks are illustrated along with list of some simple attack vectors in the following subsection.

5.2.1. SQL injection

SQL Injection is a code injection technique that exploits a security vulnerability occurring in the database layer of an application. The vulnerability is present when user input is incorrectly filtered for string literal escape characters embedded in SQL statements. It can also happen when user input is not strongly typed (in the absence of unchecked run-time errors) and thereby unexpectedly executed. This attack is an instance of a more general class of vulnerability that can occur whenever a programming or scripting language is embedded inside another.

Pre-conditions:

- Web Goat should be installed.

Execution steps:

- Open Web Goat.
- Go to Injection Flaws.
- Select String SQL Injection.
- Paste your signature in the text box named “Enter Your Last Name.”
- Paste your parameter in the text box.
- If it is not successful it will give the message “404 Forbidden Error” and if it is successful then it will send message “Whoops you entered”.

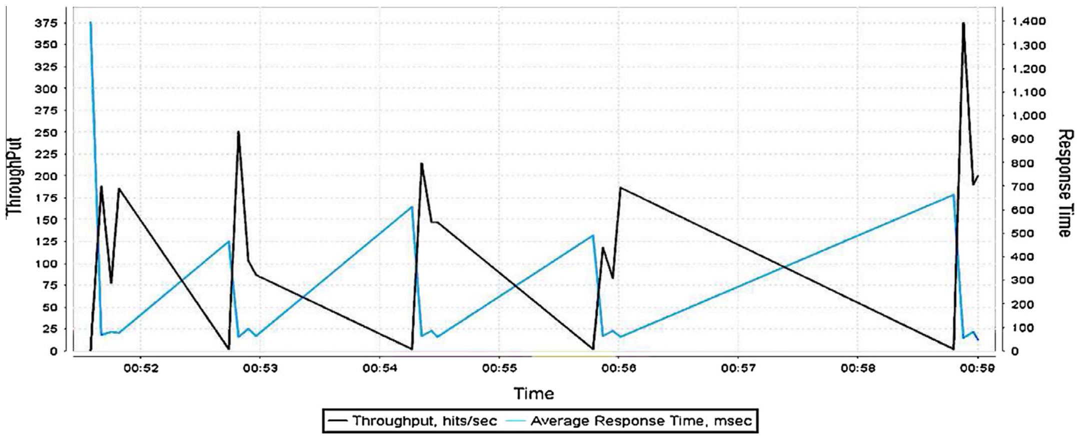

Fig. 15. Max throughput and response time of the proposed system.

Attacks vectors:

- DROP sampletable; --.
- DROP sampletable; #.
- SELECT/*avoid-spaces*/password/**/FROM/**/Members.
- admin' --.
- ' or 1 = 1 --.
- ' or 1 = 1#.

Post condition:

- Response will be 404 Forbidden message.

5.2.2. Cross Site Scripting (XSS)

Cross-site scripting (XSS) is a type of computer security vulnerability. It is typically found in web applications and enables attackers to inject a client-side script into web pages viewed by other users. An exploited cross-site scripting vulnerability can be used by hackers to bypass the access controls of a system. Recently, cross-site scripting attacks were found to account for roughly 80% of all the security vulnerabilities that were detected by Symantec [25]. The Impact of these attacks may vary from a petty nuisance to a significant security risk, depending on the sensitivity of the data handled by the vulnerable site, and the nature of security controls implemented by the site's owner.

Pre conditions:

- Web Goat should be installed.

Execution steps:

- Open Web Goat.
- Go to Cross Site Scripting.
- Select Cross site tracing.
- Paste your signature in the text box named "Purchase ".
- Paste your parameter in the text box.
- If it is not successful it will give the message "404 Forbidden Error" and if it is successful then it will send message "Whoops you entered".

Attacks vectors:

- <SCRIPT SRC = http://ha.ckers.org/xss.js> </SCRIPT \>
- <IMG SRC="javascript:alert ('XSS');">
- <IMG SRC = JaVaScRiPt:alert ('XSS')>
- <IMG SRC = javascript:alert (& quot;XSS& quot;)>

- <IMG SRC='javascript:alert ("RSnake says, 'XSS"')'>
- <IMG "> <SCRIPT>alert ('XSS')<\SCRIPT>">
- <IMG SRC = javascript:alert (String from CharCode (88,83,83)) >

Post condition:

- Response will be 404 Forbidden message.

The experimental results show that the detection capability and performance results of our system are significantly better than the existing state of the art solutions. The system successfully detects web application attacks whilst generating very few false positives. The average detection rate is 86% as compared to ModSecurity and Profense firewalls which have detection rate of 65% and 60% respectively. Additionally, our empirical results show that maximum throughput and response time of the system is 1400 hits/s and 375 ms respectively. This compares well against ModSecurity with 560 hit/s and 555 ms respectively in our benchmark environment. This is because our system is capable of detecting sophisticated attacks effectively and efficiently by analyzing the specified portions of user requests in which attacks are possible. An inference mechanism also reduces the search space by simultaneously searching parameters from headers and request bodies and then applying a single semantic rule on these parameters for validation. The examples that have been presented demonstrate that the semantic approach can be used to detect zero day and more sophisticated attacks in a real-world environment.

6. Conclusions and future work

Cyber security concerns have increased rapidly in recent years and are raising serious doubts regarding an emerging part of the global economy. The survival of e-businesses and the privacy of individual's data are becoming increasingly important as the world becomes ever more interconnected. Web applications are lucrative targets for would be hackers and crackers given their revenue generating potential. State of the art technologies have proven ineffective and are unable to provide robust security mechanisms at the application level. The semantics based detection system that has been proposed in this work is capable of making intelligent decisions by keeping the context of a domain application in view. It provides an effective and efficient security mechanism against web application attacks by capturing the context of a web application and its underlying protocols. This semantic based solution takes into account different categories of attacks; a range of encoding schemes used by hackers; the location of the attack; the impact on the system of components affected by the attack; the specification of the application layer communication protocols and polymorphic rules for mitigating attacks.

The system supports complex and zero-day attack detection. Semantic analysis is performed on the incoming and outgoing messages. The semantic rule generation is dynamic in the sense that new rules are autonomously generated through inference upon attack vectors in the knowledge base. The proposed system was compared with other open source systems, such as ModSecurity and Profense. Future work will include comparing it against commercial solutions and to develop the system into a fully commercial product. Moreover a generic rule generation mechanism could be created that can provide interoperability with existing security solutions. In this way rules could be interchanged between our system and other security applications and vice versa.

References

[1] Waleed Alrodhan, Identity management systems, Digital Identity and Access Management: Technologies and Frameworks (2011) 209.
[2] Rohan Amin, Julie Ryan, Johan van Dorp, Detecting targeted malicious email, Security & Privacy, IEEE 10 (3) (2012) 64–71.
[3] A. Anitha, V. Vaidehi, Context based application level intrusion detection system, in: International conference on Networking and Services, 2006, ICNS'06, IEEE, 2006, p. 16.
[4] Nuno Antunes, Marco Vieira, Defending against web application vulnerabilities, Computer (2012) 66–72.
[5] O.T. Arogundade, A.T. Akinwale, Z. Jin, X.G. Yang, Towards an ontological approach to information system security and safety requirement modeling and reuse, Information Security Journal: A Global Perspective 21 (3) (2012) 137–149.
[6] M. Auxilia, D. Tamilselvan, Anomaly detection using negative security model in web application, in: 2010 International Conference on Computer Information Systems and Industrial Management Applications (CISIM), IEEE, 2010, pp. 481–486.
[7] D. Balzarotti, M. Cova, V. Felmetsger, N. Jovanovic, E. Kirda, C. Kruegel, G. Vigna, Saner: composing static and dynamic analysis to validate sanitization in web applications, in: IEEE Symposium on Security and Privacy, 2008, SP 2008, IEEE, 2008, pp. 387–401.
[8] R.S. Boyer, J.S. Moore, A fast string searching algorithm, Communications of the ACM 20 (10) (1977) 762–772.
[9] T.D. Brown, Getting to the core of the matter, Strength & Conditioning Journal 28 (2) (2006) 50.
[10] P.W. Brunst, Terrorism and the internet: New threats posed by cyberterrorism and terrorist use of the internet, A War on Terror?: The European Stance on a New Threat, Changing Laws and Human Rights Implications (2009) 51.
[11] Tom Canavan, Joomla! Web Security, Packt Pub Limited, 2008.
[12] Jeremy J Carroll, Ian Dickinson, Chris Dollin, Dave Reynolds, Andy Seaborne, Kevin Wilkinson, Jena: implementing the semantic web recommendations, in: Proceedings of the 13th international World Wide Web Conference on Alternate Track Papers & Posters, ACM, 2004, pp. 74–83.
[13] You Chen, Steve Nyemba, Bradley Malin, Detecting anomalous insiders in collaborative information systems, IEEE Transactions on Dependable and Secure Computing 9 (3) (2012) 332–344.
[14] Manuel Clavel, José Meseguer, Miguel Palomino, Reflection in membership equational logic, many-sorted equational logic, horn logic with equality, and rewriting logic, Theoretical Computer Science 373 (2007) 70–91.
[15] Michael Compton, Payam Barnaghi, Luis Bermudez, Raul Garcia-Castro, Oscar Corcho, Simon Cox, John Graybeal, Manfred Hauswirth, Cory Henson, Arthur Herzog, et al., The SSN ontology of the w3c semantic sensor network incubator group, Web Semantics: Science, Services and Agents on the World Wide Web, 2012.

[16] G. Denker, L. Kagal, T. Finin, M. Paolucci, K. Sycara, Security for daml web services: Annotation and matchmaking, The Semantic Web-ISWC 2003 (2003) 335–350.

[17] Josep Domingo-Ferrer, David Sánchez, Guillem Rufian-Torrell, Anonymization of nominal data based on semantic marginality, Information Sciences (2013).

[18] Zhenhai Duan, Peng Chen, Fernando Sanchez, Yingfei Dong, Mary Stephenson, JamesMichael Barker, Detecting spam zombies by monitoring outgoing messages, IEEE Transactions on Dependable and Secure Computing 9 (2) (2012) 198–210.

[19] D. Eastlake, J. Reagle, D. Solo, M. Bartel, J. Boyer, B. Fox, B. LaMacchia, E. Simon, Xml-Signature Syntax and Processing, 2002.

[20] M. Ektefa, S. Memar, F. Sidi, L.S. Affendey, Intrusion detection using data mining techniques, in: 2010 International Conference on Information Retrieval & Knowledge Management, (CAMP), IEEE, 2010, pp. 200–203.

[21] Nan Feng, Harry Jiannan Wang, Minqiang Li, A security risk analysis model for information systems: causal relationships of risk factors and vulnerability propagation analysis, Information Sciences (2013).

[22] S. Fenz, G. Goluch, A. Ekelhart, B. Riedl, E. Weippl, Information security fortification by ontological mapping of the iso/iec 27001 standard, in: 13th Pacific Rim International Symposium on Dependable Computing, 2007, PRDC 2007, IEEE, 2007, pp. 381–388.

[23] Roy Fielding, Jim Gettys, Jeffrey Mogul, Henrik Frystyk, Larry Masinter, Paul Leach, Tim Berners-Lee, Hypertext transfer protocol–http/1.1, 1999.

[24] E. Fong, R. Gaucher, V. Okun, P.E. Black, Building a test suite for web application scanners, in: Proceedings of the 41st Annual Hawaii International Conference on System Sciences, IEEE, 2008. pp. 478–478.

[25] Marc Fossi, Gerry Egan, Kevin Haley, Eric Johnson, Trevor Mack, Téo Adams, Joseph Blackbird, Mo King Low, Debbie Mazurek, David McKinney, et al., Symantec Internet Security Threat Report Trends for 2010, vol. 16(20), 2011.

[26] Mohammad Fraiwan, Rami Al-Salman, Natheer Khasawneh, Stefan Conrad, Analysis and identification of malicious javascript code, Information Security Journal: A Global Perspective 21 (1) (2012) 1–11.

[27] Steffen Gebert, Rastin Pries, Daniel Schlosser, Klaus Heck, Internet access traffic measurement and analysis, in: Traffic Monitoring and Analysis, Springer, 2012, pp. 29–42.

[28] F.L. Greitzer, A.P. Moore, D.M. Cappelli, D.H. Andrews, L.A. Carroll, T.D. Hull, Combating the insider cyber threat, Security & Privacy IEEE 6 (1) (2008) 61–64.

[29] Tom Gruber, What is an ontology, Encyclopedia of Database Systems 1 (2008).

[30] Nicola Guarino, Christopher Welty, Evaluating ontological decisions with ontoclean, Communications of the ACM 45 (2) (2002) 61–65.

[31] Hal Berghel, Identity theft and financial fraud: Some strangeness in the proportions, Computer 45 (1) (2012) 86–89.

[32] B. Guha, B. Mukherjee, Network security via reverse engineering of tcp code: vulnerability analysis and proposed solutions, Network, IEEE 11 (4) (1997) 40–48.

[33] A. Herzog, N. Shahmehri, C. Duma, An ontology of information security, Techniques and Applications for Advanced Information Privacy and Security: Emerging Organizational, Ethical, and Human Issues (2009) 278–301.

[34] I. Horrocks et al, Daml + oil: adescription logic for the semantic web, IEEE Data Engineering Bulletin 25 (1) (2002) 4–9.

[35] Ian Horrocks, Peter F Patel-Schneider, Harold Boley, Said Tabet, Benjamin Grosof, Mike Dean, et al., Swrl: a semantic web rule language combining owl and ruleml, W3C Member Submission 21 (2004) 79.

[36] Hongxin Hu, Gail-Joon Ahn, Ketan. Kulkarni, Detecting and resolving firewall policy anomalies, IEEE Transactions on Dependable and Secure Computing 9 (3) (2012) 318–331.

[37] Jian Hua, Sanjay Bapna, How can we deter cyber terrorism?, Information Security Journal: A Global Perspective 21 (2) (2012) 102–114

[38] S. Hung, S. Liu. A User-Centric Intrusion Detection System by Using Ontology Approach, 2006.

[39] N. Jovanovic, C. Kruegel, E. Kirda, Static analysis for detecting taint-style vulnerabilities in web applications, Journal of Computer Security 18 (5) (2010) 861–907.

[40] J. Kannan, P. Maniatis, B.G. Chun, Secure data preservers for web services, in: Proceedings of the 2nd USENIX Conference on Web Application Development, USENIX Association, 2011, pp. 3–3.

[41] Graham Klyne, Jeremy J Carroll, Brian McBride, Resource description framework (RDF): concepts and abstract syntax, W3C Recommendation 10 (2004).

[42] C. Krugel, T. Toth, E. Kirda, Service specific anomaly detection for network intrusion detection, in: Proceedings of the 2002 ACM symposium on Applied computing, ACM, 2002, pp. 201–208.

[43] C.E. Landwehr, A.R. Bull, J.P. McDermott, W.S. Choi, A taxonomy of computer program security flaws, ACM Computing Surveys (CSUR) 26 (3) (1994) 211–254.

[44] Yiqun Liu, Fei Chen, Weize Kong, Huijia Yu, Min Zhang, Shaoping Ma, Liyun Ru, Identifying web spam with the wisdom of the crowds, ACM Transactions on the Web (TWEB) 6 (1) (2012) 2.

[45] G.W. Manes, D. Schulte, S. Guenther, S. Shenoi, Netglean: a methodology for distributed network security scanning, Journal of Network and Systems Management 13 (3) (2005) 329–344.

[46] Deborah L McGuinness, Frank Van Harmelen, et al, Owl web ontology language overview, W3C Recommendation 10 (2004-03) (2004) 10.

[47] J. McHugh, Testing intrusion detection systems: a critique of the 1998 and 1999 darpa intrusion detection system evaluations as performed by lincoln laboratory, ACM Transactions on Information and System Security 3 (4) (2000) 262–294.

[48] P. Ning, S. Jajodia, X.S. Wang, Abstraction-based intrusion detection in distributed environments, ACM Transactions on Information and System Security (TISSEC) 4 (4) (2001) 407–452.

[49] OWASP, Web application security testing cheat sheet, 2011. <www.owasp.org>.

[50] Top10 OWASP, Top 10–2010, The Ten Most Critical Web Application Security Risks, The Open Web Application Security Project, 2010.

[51] Sangun Park, Juyoung Kang, Using rule ontology in repeated rule acquisition from similar web sites, IEEE Transactions on Knowledge and Data Engineering 24 (6) (2012) 1106–1119.

[52] Cristian I Pinzon, Juan F De Paz, Alvaro Herrero, Emilio Corchado, Javier Bajo, Juan M Corchado, idmas-sql: Intrusion detection based on mas to detect and block sql injection through data mining, Information Sciences (2011).

[53] Jon Postel, Simple mail transfer protocol, Information Sciences (1982).

[54] Jon Postel, Joyce Reynolds, File Transfer Protocol, 1985.

[55] V. Raskin, C.F. Hempelmann, K.E. Triezenberg, S. Nirenburg, Ontology in information security: a useful theoretical foundation and methodological tool, in: Proceedings of the 2001 Workshop on New Security Paradigms, ACM, 2001, pp. 53–59.

[56] A. Razzaq, A. Hur, M. Masood, K. Latif, H.F. Ahmad, H. Takahashi, Foundation of semantic rule engine to protect web application attacks, in: 2011, 10th International Symposium on Autonomous Decentralized Systems (ISADS), IEEE, 2011, pp. 95–102.

[57] Eric Rescorla, A Schiffman, The Secure Hypertext Transfer Protocol, 1999.

[58] F.S. Rietta, Application layer intrusion detection for sql injection, in: Proceedings of the 44th Annual Southeast Regional Conference, ACM, 2006, pp. 531–536.

[59] Ivan Ristic, Modsecurity: Open Source Web Application firewall, 2010.

[60] M. Roesch, et al., Snort-lightweight intrusion detection for networks, in: Proceedings of the 13th USENIX conference on System administration, Seattle, Washington, 1999, pp. 229–238.

[61] Matthew Rowe, Miriam Fernandez, Sofia Angeletou, Harith Alani, Community analysis through semantic rules and role composition derivation, Web Semantics: Science, Services and Agents on the World Wide Web, 2012.

[62] M. Rubiolo, M.L. Caliusco, G. Stegmayer, M. Coronel, M. Gareli Fabrizi, Knowledge discovery through ontology matching: an approach based on an artificial neural network model, Information Sciences 194 (2012) 107–119.

[63] T. Ryutov, C. Neuman, K. Dongho, Z. Li, Integrated access control and intrusion detection for web servers, IEEE Transactions on Parallel and Distributed Systems 14 (9) (2003) 841–850.
[64] Igor Santos, Felix Brezo, Xabier Ugarte-Pedrero, Pablo G Bringas, Opcode sequences as representation of executables for data-mining-based unknown malware detection, Information Sciences (2011).
[65] Igor Santos, Carlos Laorden, Borja Sanz, Pablo G Bringas, Enhanced topic-based vector space model for semantics-aware spam filtering, Expert Systems with applications 39 (1) (2012) 437–444.
[66] S.T. Sarasamma, Q.A. Zhu, J. Huff, Hierarchical kohonenen net for anomaly detection in network security, IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics 35 (2) (2005) 302–312.
[67] Farrukh Shahzad, M. Shahzad, Muddassar Farooq, In-execution dynamic malware analysis and detection by mining information in process control blocks of linux os, Information Sciences (2011).
[68] Lwin Khin Shar, Hee Beng Kuan Tan, Defending against cross-site scripting attacks, Computer 45 (3) (2012) 55–62.
[69] Quan Z Sheng, Zakaria Maamarb, Lina Yaoa, Claudia Szaboa, Scott Bournea, Behavior modeling and automated verification of web services, Information Sciences (2012).
[70] Evren Sirin, Bijan Parsia, Bernardo Cuenca Grau, Aditya Kalyanpur, Yarden Katz, Pellet: A practical owl-dl reasoner, Web Semantics: Science, Services and Agents on the World Wide Web 5 (2) (2007) 51–53.
[71] Jungsuk Song, Hiroki Takakura, Yasuo Okabe, Koji Nakao, Toward a more practical unsupervised anomaly detection system, Information Sciences (2011).
[72] M. Stein, M. Beer, V. Kreinovich, Bayesian approach for inconsistent information, Information Sciences (2013).
[73] Hung-Min Sun, Yao-Hsin Chen, Yue-Hsun Lin, opass: A user authentication protocol resistant to password stealing and password reuse attacks, IEEE Transactions on Information Forensics and Security 7 (2) (2012) 651–663.
[74] C.Y. Tan, S.R. Tan, B. Morel, 19601 Information Warfare, in: 19601 Information Warfare, Carnegie Mellon, 2011, pp. 15.
[75] J. Undercoffer, J. Pinkston, A. Joshi, T. Finin, A target-centric ontology for intrusion detection, in: 18th International Joint Conference on Artificial Intelligence, 2004, pp. 9–15.
[76] Sabrina De Capitani Di Vimercati, Sara Foresti, Sushil Jajodia, Stefano Paraboschi, Giuseppe Psaila, Pierangela Samarati, Integrating trust management and access control in data-intensive web applications, ACM Transactions on the Web (TWEB) 6 (2) (2012) 6.
[77] V. Viswanathan, Ilango Krishnamurthi, Ranking semantic relationships between two entities using personalization in context specification, Information Sciences (2012).
[78] Mike Volordarsky, Olga M Londer, Brett Hill, Bernard Cheah, Steve Schofield, Carlos Aguiar Mares, Kurt Meyer, et al, Internet Information Services (IIS) 7.0 resource kit, Microsoft Press, 2010.
[79] M. Vrancianu, L.A. Popa, Considerations regarding the security and protection of e-banking services consumers' interests, The Amfiteatru Economic Journal 12 (28) (2010) 388–403.
[80] X.F. Wang, J.L. Zhou, S.S. Yu, L.Z. Cai, Data mining methods for anomaly detection of http request exploitations, Fuzzy Systems and Knowledge Discovery (2005) 484. 484.
[81] Dianxiang Xu, Manghui Tu, Michael Sanford, Lijo Thomas, Daniel Woodraska, Weifeng Xu, Automated security test generation with formal threat models, IEEE Transactions on Dependable and Secure Computing 9 (4) (2012) 526–540.
[82] Zhe Yao, Philip Mark, Michael Rabbat, Anomaly detection using proximity graph and pagerank algorithm, IEEE Transactions on Information Forensics and Security 7 (4) (2012) 1288–1300.
[83] Xin Zhao, Atul Prakash, Wsf: An http-Level firewall for Hardening Web Servers, 2005.