

Incorporation of Application Layer Protocol Syntax into Anomaly Detection

Patrick Düssel¹, Christian Gehl¹, Pavel Laskov^{1,2}, and Konrad Rieck¹

¹ Fraunhofer Institute FIRST

Intelligent Data Analysis, Berlin, Germany

² University of Tübingen

Wilhelm-Schickard-Institute for Computer Science, Tübingen, Germany

{duessel,gehl,laskov,rieck}@first.fraunhofer.de

Abstract. The syntax of application layer protocols carries valuable information for network intrusion detection. Hence, the majority of modern IDS perform some form of protocol analysis to refine their signatures with application layer context. Protocol analysis, however, has been mainly used for misuse detection, which limits its application for the detection of unknown and novel attacks. In this contribution we address the issue of incorporating application layer context into anomaly-based intrusion detection. We extend a payload-based anomaly detection method by incorporating structural information obtained from a protocol analyzer. The basis for our extension is computation of similarity between attributed tokens derived from a protocol grammar. The enhanced anomaly detection method is evaluated in experiments on detection of web attacks, yielding an improvement of detection accuracy of 49%. While byte-level anomaly detection is sufficient for detection of buffer overflow attacks, identification of recent attacks such as SQL and PHP code injection strongly depends on the availability of application layer context.

Keywords: Anomaly Detection, Protocol Analysis, Web Security.

1 Introduction

Analysis of application layer content of network traffic is getting increasingly important for protecting modern distributed systems against remote attacks. In many cases such systems must deal with untrusted communication parties, e.g. the majority of web-based applications. Application-specific attack can only be detected by monitoring the content of a respective application layer protocol.

Signature-based intrusion detection systems (IDS) possess a number of mechanisms for analyzing the application layer protocol content ranging from simple payload scanning for specific byte patterns, as in Snort [19], to protocol analysis coupled with a specialized language for writing signatures and policy scripts, as in Bro [15]. By understanding the protocol context of potential attack patterns, significant improvements in the detection accuracy of unknown application layer attacks can be achieved.

The main drawback of signature-based IDS is their dependence on the availability of appropriate exploit signatures. Rapid development of new exploits and their growing variability make keeping signatures up-to-date a daunting if not impossible task. This motivates investigation of alternative techniques, such as anomaly detection, that are capable to detect previously unknown attacks.

Incorporation of the protocol context into anomaly detection techniques is, however, a difficult task. Unlike a signature-based system which looks for a *specific pattern* within a *specific context*, an anomaly-based system must translate *general knowledge* about patterns and their context into a numeric measure of abnormality. The latter is usually measured by a distance from some typical “profile” of a normal event. Hence, incorporation of protocol syntax into the computation of distances between network events (e.g. packets, TCP connections etc.) is needed in order to give anomaly detection algorithm access to protocol context.

The idea behind the proposed method for syntactically aware comparison of network event is roughly the following. A protocol analyzer can transform a byte stream of each event into a structured representation, e.g. a sequence of token/attribute pairs. The tokens correspond to particular syntactic constructs of a protocol. The attributes are byte sequences attached to the syntactic elements of a protocol. Measuring similarity between sequences is well understood, for general object sequences [18], as well as byte streams of network events [16; 17; 23]. To compare sequences of token/attribute pairs we perform computation of sequential similarity at two levels: for sequences of tokens (with partial ordering) and byte sequence values of corresponding tokens. The resulting similarity measure, which we call the *attributed token kernel*, can be transformed into a Euclidean distance easily handled by most anomaly detection algorithms.

To illustrate effectiveness of the protocol syntax-aware anomaly detection, we apply the proposed method for detection of web application attacks. Such attacks, for example SQL injection, cross-site scripting (XSS) and other script injection attacks, are particularly difficult for detection due to (a) their variability, which makes development of signature a futile exercise, and (b) entanglement of attack vector within the protocol framework, which makes simple byte-level content analysis ineffective. Our experiments carried out on client-side HTTP traffic demonstrate a strong performance improvement for these kinds of attacks compared to byte-level analysis. The proposed method should be easily adaptable for other application layer protocols for which a protocol dissector is available.

2 Related Work

A large amount of previous work in the domain of network intrusion detection systems has focused on features derived from network and transport layer protocols. An example of such features can be found in the data mining approach of Lee and Stolfo [9], containing packet, connection and time window features derived from IP and TCP headers. The same work has pioneered the use of

“content” features that comprised selected application-level properties such as number shell prompts, number of failed login prompts, etc. deemed to be relevant for detection of specific attacks. Similar features comprising selected keywords from application layer protocols have been used by Mahoney and Chan for anomaly detection [12].

General content-based features using payload distribution of specific byte groups have been first proposed by Kruegel et al. [7] in the context of service-specific anomaly detection using separate normality models for different application layer protocols. Full distributions of byte values have been considered by Wang and Stolfo [23], extended to models of various languages that can be defined over byte sequences, e.g. n -grams [16; 22].

Incorporation of application-level protocol information into the detection process has been first realized in signature-based IDS. Robust and efficient protocol parsers have been developed for the Bro IDS [15]; however, until recently they were tightly coupled with Bro’s signature engine, which has prevented their use in other systems. The development of a stand-alone protocol parser binpac [14] has provided a possibility for combining protocol parsing with other detection techniques. Especially attractive features of binpac are incremental and bi-directional parsing as well as error recovery. These issues are treated in depth in the recent. Similar properties at a more abstract level are exhibited by the recent interpreted protocol analyzer GAPAL [1].

Combination of protocol parsing and anomaly detection still remains largely unexplored. By considering separate models corresponding to specific URI attributes in the HTTP protocol, Kruegel and Vigna [8] have developed a highly effective system for the detection of web attacks. The system combines models built for specific features, such as length and character distribution, defined for attributes of applications associated with particular URI paths. Ingham et al. [6] learn a generalized DFA representation of tokenized HTTP requests using delimiters defined by the protocol. The DFA inference and the n -grams defined over an alphabet of protocol tokens performed significantly better than other content-based methods in a recent empirical evaluation [5]. Our approach differs from the work of Ingham and Inoue in that our method explicitly operates on a two-tier representation – namely token/attribute pairs – obtained from a full protocol analyzer (binpac/Bro) which provides a more fine-grained view on HTTP traffic.

3 Methodology

A payload-based anomaly detection approach benefits from its ability to cope with unknown attacks. The architecture of our system which is specifically built for the requirements of anomaly detection at application layer is illustrated in Fig. 1. The following four stages outline the essential building blocks of our approach and will be explained in detail for the rest of this section.

1. **Protocol Analysis.** Inbound packets are captured from the network by Bro which provides robust TCP re-assembly and forwards incoming packets

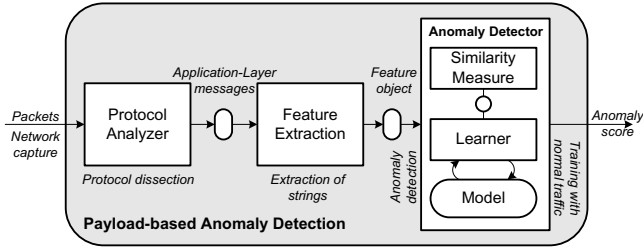


Fig. 1. Architecture of payload-based anomaly detection

to the *binpac* protocol analyzer. The latter extracts application-layer events at different levels of granularity, typical for common text protocols such as HTTP, FTP, SMTP or SIP. Initially, extraction starts at the level of request/response messages and can be further refined to specific protocol elements. A key benefit of using protocol dissectors as part of data pre-processing is the capability to incorporate expert knowledge into the feature extraction process. Details on protocol analysis can be found in Section 3.1.

2. **Feature Extraction.** Each parsed event is mapped into a feature vector which reflects essential characteristics. However, an event can be projected into byte-level or syntax-level feature spaces. Our approach allows to combine both. Details of the feature extraction process can be found in Section 3.2.
3. **Similarity Computation.** The similarity computation between strings is a crucial task for payload-based anomaly detection. Once a message is brought into a corresponding vectorial representation two events can be compared by computing their pairwise distance in a high-dimensional geometric space. We extend the common string similarity measures for token/attribute representations provided by a protocol parser, as explained in Section 3.4.
4. **Anomaly Detection.** In an initial training phase the anomaly detection algorithm learns a global model of "normality" which can be interpreted as a center of mass of a subset of training data. At detection time an incoming message is compared to a previously learned model and based on its distance an anomaly score is computed. The anomaly detection process is described in Section 3.3.

3.1 Protocol Analysis

Network protocols specify rules for syntax, semantics, and synchronization for bidirectional data communication between network endpoints. The protocol syntax is usually defined by an augmented Backus-Naur Form.

Our goal is to analyze network traffic based on the grammatical characteristics of an underlying protocol in order to detect network attacks. We perform the analysis at the granularity of protocol elements present in request/response messages. The HTTP protocol definition is a classical representative of such request/response protocols. Additionally, HTTP is the most frequently used protocol for web applications and so we limit our focus to this particular specification.

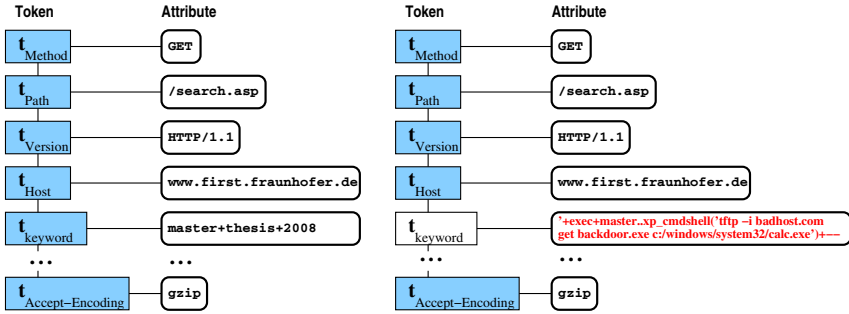


Fig. 2. Attributed token sequence of a benign and malicious HTTP request

Usually, a request is transmitted in a single TCP packet, although certain features of the TCP/IP (e.g. fragmentation) can make the process more complicated. An application protocol analyzer, e.g. binpac [14], allows one to transform the network event’s raw byte payload into a structured representation reflecting the syntactic aspects of an underlying application protocol. For example in the syntactic context of HTTP the sequence “Content-Length: 169” refers to a header “Content-Length” with attribute “169”.

We present two examples of HTTP connections to illustrate the effect of applying binpac’s grammar and parser to an application-level specification. The first example is a benign GET request containing common HTTP headers together with a CGI parameter.

```
GET /search.asp?keyword=master+thesis+learning HTTP/1.1\r\nHost: www.firs
t.fraunhofer.de\r\nUser-Agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows
NT 5.1; SV1)\r\nConnection: Keep-alive\r\nAccept-Encoding: gzip\r\n\r\n
```

The second example shows a SQL injection attack against a Microsoft SQL Server exploiting the vulnerable CGI parameter *keyword*.

```
GET /search.asp?keyword='+exec+master..xp_cmdshell('tftp -i badhost.com g
et backdoor.exe c:/windows/system32/calc.exe')+' HTTP/1.1\r\nHost: www.f
```

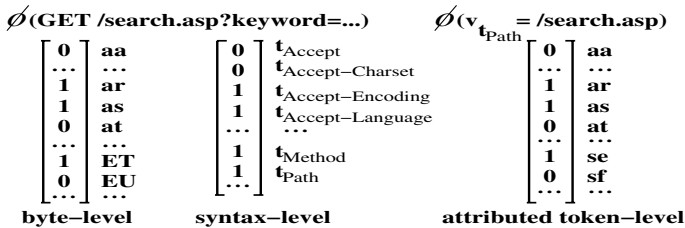


Fig. 3. Feature extraction for byte-level, syntax-level and attributed token-level

```
irst.fraunhofer.de\r\nUser-Agent: Mozilla/5.0\r\nConnection: Keep-alive\r\n\nAccept-Language: en-us,en\r\n\nAccept-Encoding: gzip\r\n\r\n
```

The token/attribute pairs generated by the protocol analyzer are shown in Fig. 2. One can clearly see that the difference between these two requests is given by the attributes attached to the token “keyword” of the parsed GET request.

3.2 Feature Extraction

Anomaly detection usually requires data to be in a vectorial representation. Therefore, the feature extraction process maps application layer messages, such as HTTP requests, into a feature space in which similarity between messages can be computed. Protocol dissection in the pre-processing stage allows to deploy feature extraction on two different levels explained in the following.

Byte-Level Features. An intuitive way to represent a message at byte level is to extract unique substrings by moving a sliding window of a particular length n over a message. The resulting set of feature strings are called n -grams. The first example in Fig. 3 shows the mapping of the benign HTTP request introduced in Section 3.1 into a binary 2-gram feature space.

Syntax-Level Features. Each message is transformed into a set of tokens. Although shown as a sequence of tokens in Fig. 2, these feature have only partial sequential order as the order of many (but not all) tokens in an HTTP request is not defined. An embedding of the benign HTTP request into a token feature space is exemplarily presented in Fig. 3.

With the extraction of byte-level features from token attributes a semantic notion is implicitly assigned to the corresponding protocol token. Note, that an explicit representation of application layer messages can easily become computational infeasible due to the combinatorial nature of byte sequences. Therefore, we use efficient data structures such as suffix trees or hash tables (“feature objects” in Fig. 1) that allow an implicit vectorial representation.

3.3 Anomaly Detection

Once, application layer messages are mapped into some feature space the problem of anomaly detection can be solved mathematically considering the geometric relationship between vectorial representations of messages. Although anomaly detection methods have been successfully applied to different problems in intrusion detection, e.g. identification of anomalous program behavior [e.g. 3; 4], anomalous packet headers [e.g. 11] or anomalous network payloads [e.g. 8; 16; 17; 22; 23], all methods share the same concept – *anomalies are deviations from a model of normality* – and differ in concrete notions of normality and deviation. For our purpose we use the one-class support vector machine (OC-SVM) proposed in [21] which fits a minimal enclosing hypersphere to the data which is characterized by a center c and a radius R as illustrated in Fig. 4.

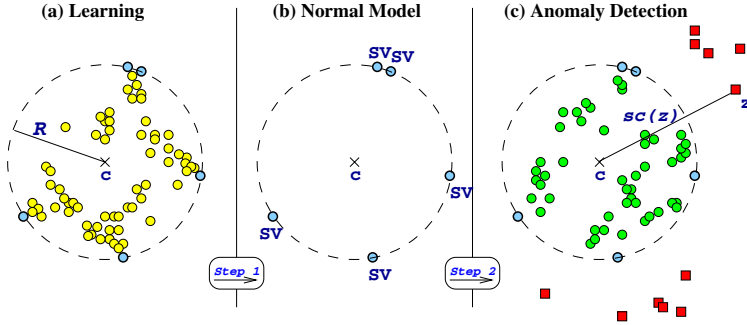


Fig. 4. Learning and anomaly detection using a one-class support vector machine

Mathematically, this can be formulated as a quadratic programming optimization problem:

$$\begin{aligned}
 & \min_{\substack{R \in \mathbb{R} \\ c \in \mathbb{R}^n}} R^2 + C \sum_{i=1}^n \xi_i \\
 & \text{subject to: } \|\phi(x_i) - c\|^2 \leq R^2 + \xi_i, \\
 & \xi_i \geq 0.
 \end{aligned} \tag{1}$$

By minimizing R^2 the volume of the hypersphere is minimized given the constraint that training objects are still contained in the sphere which can be expressed by the constraint in Eq.(1). A major benefit of this approach is the control of generalization ability of the algorithm [13], which enables one to cope with noise in the training data and thus dispense with laborious sanitization, as recently proposed by Cretu et al. [2]. By introducing slack variables ξ_i and penalizing the cost function we allow the constraint to be softened. The regularization parameter C controls the trade-off between radius and errors (number of training points that violate the constraint). The solution of the optimization problem shown in Eq. (1) yields two important facts:

1. The center $c = \sum_i \alpha_i \phi(x_i)$ of the sphere is a linear combination of data points, while α_i is a sparse vector that determines the contribution of the i -th data point to the center. A small number of training points having $\alpha_i > 0$ are called **support vectors** (SV) which define the model of normality as illustrated in Fig. 4.
2. The radius R which is explicitly given by the solution of the optimization problem in Eq. (1) refers to the distance from the center c of the sphere to the boundary (defined by the set of support vectors) and can be interpreted as a threshold for a decision function.

Finally, having determined a model of normality the anomaly score $sc(z)$ for a test object z can be defined as the distance from the center:

$$sc(z) = \|\phi(z) - c\|^2 = k(z, z) - 2 \sum_i \alpha_i k(z, x_i) + \sum_{i,j} \alpha_i \alpha_j k(x_i, x_j), \tag{2}$$

where the similarity measure $k(x, y)$ between two points x and y refers to a *kernel* function which allows to compute similarity between two data points embedded in some geometric feature space \mathcal{F} . Given an arbitrary mapping $\phi : \mathcal{X} \mapsto \mathcal{F}$ of a data point $x \in \mathcal{X}$ a common similarity measure can be defined from the *dot product* in \mathcal{F} :

$$k(x, y) = \langle \phi(x), \phi(y) \rangle = \sum_{i=1}^N \phi_i(x) \phi_i(y), \quad (3)$$

where $\phi_i(x)$ refers to the i -th dimension of a data point x mapped into \mathcal{F} .

3.4 Similarity Measures

Having defined a set of characteristic features we can develop similarity measures that, given two data points, returns a real number reflecting their similarity.

Spectrum Kernel. A natural way to define a kernel $k(s, u)$ between two application layer messages s and u is to consider n -grams that both messages have in common. Given the set \mathcal{A}^n of all possible strings of length n induced by an alphabet \mathcal{A} we can define a kernel function computing the dot product of both messages embedded into a $|\mathcal{A}|^n$ dimensional feature space.

$$k(s, u) = \langle \phi(s), \phi(u) \rangle = \sum_{w \in \mathcal{A}^n} \phi_w(s) \phi_w(u), \quad (4)$$

where $\phi_w(s)$ ¹ refers to a signum function that returns 1 if the w is contained in s and 0 otherwise. The kernel function $k(s, u)$ is referred to as *spectrum kernel* [10]. Using n -grams to characterize messages is intuitive but may result in a high-dimensional feature space. From an algorithmic point of view, it may seem that running a summation over all possible substrings $w \in \mathcal{A}^n$ can become computationally infeasible. Thus, special data structures such as tries, suffix trees or suffix arrays enable one to compute $k(s, u)$ in $O(|s| + |u|)$ time [20]. Interestingly, the Euclidean distance $d_{eucl}(s, u)$ which is of particular interest for anomaly detection can be easily derived from the above kernel formulation:

$$d_{eucl}(s, u) = \sqrt{k(s, s) + k(u, u) - 2k(s, u)}. \quad (5)$$

Attributed Token Kernel. In this section we address the problem of how to combine string similarity as defined in Section 3.4 and structural similarity of two application layer messages. Consider an alphabet $\mathcal{A} = t_1, t_2, \dots, t_z$ of tokens. Let $s = s_1, s_2, \dots, s_n$ and $u = u_1, u_2, \dots, u_m$, $s_i, u_j \in \mathcal{A}$, be the token sets of two application layer messages returned by a protocol analyzer. Let v_t^u denote an attribute attached to the token t found in a token set u . We can define a kernel function for attributes in the same way we have done it in Eq.(4):

$$k_t(s, u) = \begin{cases} k(v_t^s, v_t^u), & \text{if } t \text{ is found in both } s \text{ and } u \\ 0, & \text{otherwise.} \end{cases} \quad (6)$$

¹ Alternatively, the embedding function $\phi_w(s)$ may return the count or the term frequency of w in s .

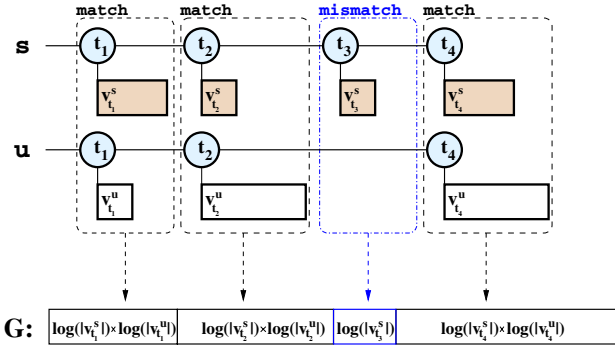


Fig. 5. Normalization of the similarity measure between attributed token sequences

Finally, we combine definitions (4) and (6) in the following way:

$$k(s, u) = \sum_{t \in s \cap u} \frac{\gamma(t)}{G} k_t(s, u), \tag{7}$$

where $s \cap u$ is an intersection of tokens in s and u , $\gamma(t)$ is a weight assigned to a particular token, and G is an overall normalization constant. The computation of the kernel function in Eq. (7) runs through all matching tokens and computes the similarity measure defined in Eq. (4) over associated attributes at byte level.

The weighting constant is defined as:

$$\gamma(t) = \log(|v_t^s|) \times \log(|v_t^u|),$$

and the normalization constant is defined as:

$$G = \sum_{t \in s \cup u} \gamma(t).$$

These constants are motivated by the need to normalize contributions from individual value sequences according to their lengths. The overall normalization constant also includes contributions from mismatching tokens. This allows the latter to indirectly influence the similarity measure by rescaling contributions from matching tokens; direct influence is not possible since it does not make any sense to compare value strings for mismatching tokens. For the rest of this paper we refer to the kernel function defined in Eq. (7) as *attributed token kernel*.

4 Experiments

We evaluate the impact of incorporation of protocol context into anomaly detection on two data sets containing HTTP traffic. Both data sets comprise exploits taken from the Metasploit framework² as well as from common security mailing lists and archives such as xssed.com, sla.ckers.org or Bugtraq.

² <http://www.metasploit.com/>

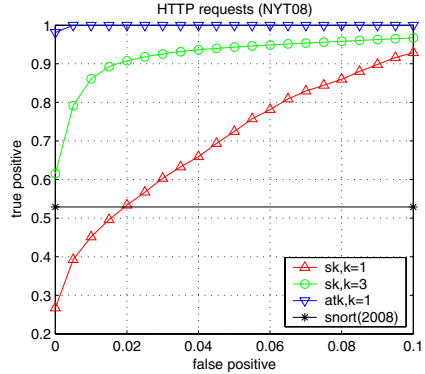
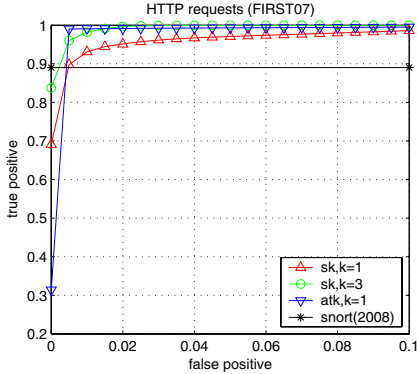
The first dataset (FIRST07) contains a sample of 16000 normal HTTP connections drawn from two months incoming traffic recorded at our institute’s web server. We mixed normal data with 42 attack instances of 14 different types, mostly **overflow-based attacks** (8 stack overflows, 4 heap overflows, 1 format string exploit, 1 web application attack). Except for using unsanitized data, this is a typical experimental protocol used in previous work, e.g. [5; 8; 23].

The second data set (NYT08) has been motivated by the observation that a traffic profile of a mostly static web site may be quite different from a profile of a site running web applications. In particular, a much more involved structure of URI and certain header fields can be expected in normal traffic, which may cause significantly higher false positive rates than the ones reported in evaluations on static normal traffic. Since we do not have access to a “pure” web application traffic, as e.g. the Google data used by Kruegel and Vigna [8], we have attempted to simulate such traffic using specially developed crawlers. Our request engine analyzes the structure and content of existing static traffic (in our case, the FIRST07 data) and generates valid HTTP requests using frequently observed protocol header elements while randomly visiting a target domain. We generated 15000 client-side connections accessing the *nytimes.com* news site and mixed the traffic with 17 instances of **web application attacks** (1 buffer overflow, 1 arbitrary command execution vulnerability, 3 Perl injection, 2 PHP include, 4 SQL injections and 6 XSS attacks). XSS attacks and SQL injections were launched against two prototypical CGI programs that were adapted to the structure of a “login”-site (8 CGI parameters) and a “search”-site (18 CGI parameters) found in the *nytimes.com* domain. In order to obtain a normal behavior for these sites we generated 1000 requests containing varying user names and passwords as well as search phrases and realistic parameter values for both prototypical “mirrors” of NYT sites. In contrast to previous work in which the structure of HTTP requests in exploits was used “as is”, we have also normalized the attacks by using the same typical headers injected by crawlers, in order to avoid attacks being flagged as anomalous purely because of programmatic but non-essential difference in their request structure.

In our experiments, a model was trained using 500 requests taken from a normal pool of data and subsequently applied to 500 unknown requests taken from a distinct test set mixed with attacks. The detection accuracy was measured in terms of area under receiver operating characteristic curve ($ROC_{0,1}$) for false positive rates $\leq 10\%$. For statistical reasons experiments were repeated and the detection accuracy was averaged over 50 repetitions.

4.1 Detection Accuracy: Byte-Level Versus Attributed Token-Level

In the experiment on FIRST07 we investigate the detection of overflow-based attacks in HTTP requests. As shown in Fig. 6(a), the spectrum kernel **sk** on binary 3-grams attains a detection rate of 82% at 0% false positives anomaly, which is comparable to Snort. The only attack that repeatedly suffered a high false positive rate (1.5%-2%) is a *file inclusion* (“php_include”) which is the only non-buffer-overflow attack in this data set. Similar results have been obtained



(a) ROC curve of OC-SVM on the FIRST07 data set (overflow attacks)

(b) ROC curve of OC-SVM on the NYT08 data set (web application attacks)

Fig. 6. Detection accuracy on intrusion detection datasets FIRST07 and NYT08

for the attributed token kernel except for some initial false positives due to proxy requests containing anonymized header attributes.

In the experiment on NYT08 we investigate the detection of attacks that are bound to specific parameters of a CGI program. The results presented in Fig. 6(b) reveal that the detection of *web application attacks* using byte-level similarity over n -grams is much more difficult than for overflow-based attacks (Fig. 6(a)). The OC-SVM achieves its best detection rate of approximately 64% at zero percent false positives using a spectrum kernel (**sk**) over 3-grams and a binary feature embedding. Moreover, it can be observed that the accuracy of byte-level detection rises by increasing the size of n -grams. As illustrated in Fig. 6(b) significant improvements can be obtained by incorporating structure of application layer messages into payload-based anomaly detection as illustrated in Fig. 2. It turns out that anomaly detection using the attributed token kernel (**atk**) achieves a 97% detection rate without suffering any false positive.

In order to assess the computational effort of our method we measured the runtime for involved processing steps. Parsing and feature extraction require on average 1.2ms per request; average kernel computation and anomaly detection take 3.5ms per request. The overall processing time of 4.7ms per request yields a throughput of approximately 212 HTTP requests per second.

4.2 Visualization of Discriminative Features

In order to illustrate the increase of discriminative power gained through incorporation of application layer context into anomaly detection Fig. 4.2 displays 1-gram frequency differences between 1000 normal HTTP requests and two different attacks, a buffer overflow and a SQL injection. A frequency difference of 1 arises from bytes that only appear in normal data points whereas bytes that can be exclusively observed in the attack instance result in a frequency difference of -1. Bytes with a frequency difference close to zero do not provide

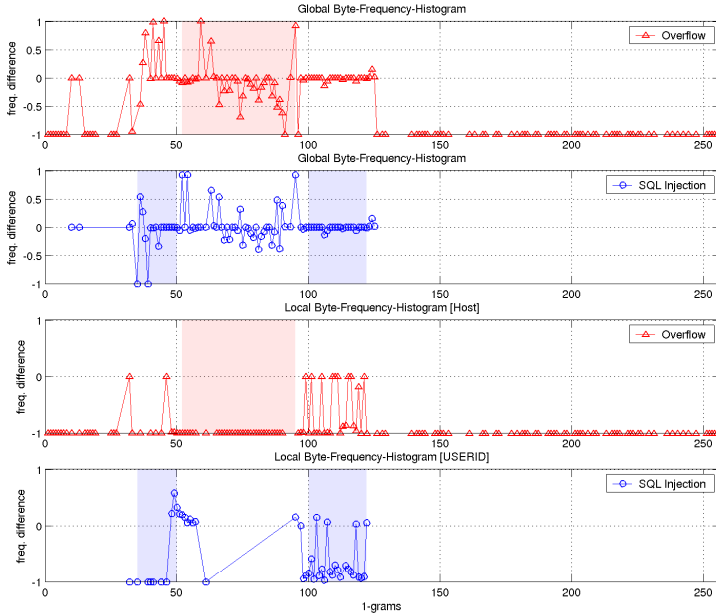


Fig. 7. Byte-frequency differences at request-level and token-level

discriminative information. The upper two charts display frequency differences for a buffer overflow attack (“edirectory_host_shikata_ga_nai”) and a SQL injection (“sql_union_injection”) that results from conventional byte-level analysis. The buffer overflow can be easily detected due to the presence of a large amount of non-printable characters (a large number of points at -1 for lower byte values and values above 128). On the other hand, the SQL injection contains mostly ASCII characters, which is reflected by close to zero frequency differences for most of the printable characters. As a consequence, the detection of this attack is relatively difficult. The lower two charts show the frequency differences for the same attacks within their appropriate application layer context. The frequency differences of the buffer overflow attack become even more obvious by examining the local byte distribution within the exploited request parameter “Host”. Similar clarification takes place for the SQL injection attack for which many previously normal bytes become clearly anomalous considering the CGI parameter “USERID”. This results in a major improvement of detection accuracy.

4.3 Comparison with Other Methods

To give a comparison to different intrusion detection techniques we examined a model-based anomaly detection method (**model-based AD**) proposed by Kruegel and Vigna [8] and the well-known signature IDS **Snort**³. The model-based detection method applies a number of different models to individual

³ VRT Certified Rules (registered user release) downloaded 07/15/2008.

Table 1. Comparison of fp-rates between model-based AD, Snort and OC-SVM

HTTP attacks (NYT08 dataset)	model-based AD			Snort	OC-SVM	
	ALM	ICD	Combined		sk	atk
edirectory_host_alpha_mixed	.0728	.0208	.0362	+	.0	.0
sql_l=1	.0112	.0379	.0214	-	.1798	.0002
sql_backdoor	.0	.0006	.0	+	.0004	.0
sql_fileloader	.0025	.0006	.0010	-	.0065	.0
sql_union_injection	.0	.0020	.0	-	.0047	.0
xss_alert	.0	.0	.0	+	.0388	.0
xss_dom_injection	.0	.0	.0	+	.0001	.0
xss_img_injection	.0	.0	.0	-	.0004	.0

parameters of HTTP queries. By learning various parameter models this method creates a "user profile" for each server-side program that is compared against incoming requests for a particular resource. To allow a fair comparison to our method we build models of not only CGI parameters in the URI, but also of header parameters and CGI parameters present in the request's body.

In our experiments the model-based AD comprises two models:

Attribute length model (ALM) estimates the attribute length distribution of CGI parameters and detects data points that significantly deviate from normal models.

Attribute character distribution model learns the *idealized character distribution* (ICD) of a given attribute. Using ICD instances are detected whose sorted frequencies differ from the previously learned frequency profile.

A comparison of false positive rates per attack class (percentage of false positives in test set given a detection of all instances from that attack class) is provided in Table 1. Anomaly detection using the attributed token kernel exhibits almost perfect accuracy of the 8 selected attacks, whereas both models of Kruegel and Vigna as well as their combination suffer from significant false positive rates for the first two attacks. Interestingly, Snort reported alarms for most of the cross site scripting and buffer overflow instances but failed to detect the majority of SQL injections which shows that specific exploits may require customization of signatures in order to provide a consistent protection.

5 Conclusions

In this paper, we have developed a general method for the incorporation of application layer protocol syntax into anomaly detection. The key instrument of our method is computation of similarity between token/attribute sequences that can be obtained from a protocol analyzer. A combined similarity measure is developed for such sequences which takes into account the syntactic context contained in tokens as well as the byte-level payload semantics.

The proposed method has proved to be especially useful for the detection of web application attacks. We have carried out experiments on realistic traffic

using the HTTP protocol analyzer developed in binpac. Although the additional effort of protocol analysis does not pay off for simple buffer overflow exploits, the detection rate for web application attacks has been boosted from 70% to 100% in the false positive rate interval of less than 0.14%. Surprisingly, our method has even outperformed a very effective model-based method of Kruegel and Vigna [8] that was specially designed for the detection of web application attacks.

Due to its generality, the proposed method can be used with any other application layer protocol for which a protocol analyzer is available. It can be deployed in a variety of distributed systems applications, especially the ones for which very few examples of potential exploits are currently known (e.g. IP multimedia infrastructure and SCADA systems).

Acknowledgements. This work was supported by the German Bundesministerium für Bildung und Forschung (BMBF) under the project ReMIND (FKZ 01-IS07007A).

References

- [1] Borisov, N., Brumley, D., Wang, H., Dunagan, J., Joshi, P., Guo, C.: Generic application-level protocol analyzer and its language. In: Proc. of Network and Distributed System Security Symposium (NDSS) (2007)
- [2] Cretu, G., Stavrou, A., Locasto, M., Stolfo, S., Keromytis, A.: Casting out demons: Sanitizing training data for anomaly sensors. In: *ieesp* (to appear, 2008)
- [3] Forrest, S., Hofmeyr, S., Somayaji, A., Longstaff, T.: A sense of self for unix processes. In: Proc. of IEEE Symposium on Security and Privacy, Oakland, CA, USA, pp. 120–128 (1996)
- [4] Gao, D., Reiter, M., Song, D.: Behavioral distance measurement using hidden markov models. In: Zamboni, D., Krügel, C. (eds.) RAID 2006. LNCS, vol. 4219, pp. 19–40. Springer, Heidelberg (2006)
- [5] Ingham, K.L., Inoue, H.: Comparing anomaly detection techniques for http. In: Kruegel, C., Lippmann, R., Clark, A. (eds.) RAID 2007. LNCS, vol. 4637, pp. 42–62. Springer, Heidelberg (2007)
- [6] Ingham, K.L., Somayaji, A., Burge, J., Forrest, S.: Learning dfa representations of http for protecting web applications. *Computer Networks* 51(5), 1239–1255 (2007)
- [7] Kruegel, C., Toth, T., Kirda, E.: Service specific anomaly detection for network intrusion detection. In: Proc. of ACM Symposium on Applied Computing, pp. 201–208 (2002)
- [8] Kruegel, C., Vigna, G.: Anomaly detection of web-based attacks. In: Proc. of 10th ACM Conf. on Computer and Communications Security, pp. 251–261 (2003)
- [9] Lee, W., Stolfo, S.: A framework for constructing features and models for intrusion detection systems. *ACM Transactions on Information Systems Security* 3, 227–261 (2000)
- [10] Leslie, C., Eskin, E., Noble, W.: The spectrum kernel: A string kernel for SVM protein classification. In: Proc. Pacific Symp. Biocomputing, pp. 564–575 (2002)
- [11] Mahoney, M., Chan, P.: PHAD: Packet header anomaly detection for identifying hostile network traffic. Technical Report CS-2001-2, Florida Institute of Technology (2001)

- [12] Mahoney, M., Chan, P.: Learning nonstationary models of normal network traffic for detecting novel attacks. In: Proc. of ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD), pp. 376–385 (2002)
- [13] Müller, K.-R., Mika, S., Rätsch, G., Tsuda, K., Schölkopf, B.: An introduction to kernel-based learning algorithms. *IEEE Neural Networks* 12(2), 181–201 (2001)
- [14] Pang, R., Paxson, V., Sommer, R., Peterson, L.: binpac: a yacc for writing application protocol parsers. In: Proc. of ACM Internet Measurement Conference, pp. 289–300 (2006)
- [15] Paxson, V.: Bro: a system for detecting network intruders in real-time. In: Proc. of USENIX Security Symposium, pp. 31–51 (1998)
- [16] Rieck, K., Laskov, P.: Detecting unknown network attacks using language models. In: Büschkes, R., Laskov, P. (eds.) DIMVA 2006. LNCS, vol. 4064, pp. 74–90. Springer, Heidelberg (2006)
- [17] Rieck, K., Laskov, P.: Language models for detection of unknown attacks in network traffic. *Journal in Computer Virology* 2(4), 243–256 (2007)
- [18] Rieck, K., Laskov, P.: Linear-time computation of similarity measures for sequential data. *Journal of Machine Learning Research* 9, 23–48 (2008)
- [19] Roesch, M.: Snort: Lightweight intrusion detection for networks. In: Proc. of USENIX Large Installation System Administration Conference LISA, pp. 229–238 (1999)
- [20] Shawe-Taylor, J., Cristianini, N.: Kernel methods for pattern analysis. Cambridge University Press, Cambridge (2004)
- [21] Tax, D., Duin, R.: Data domain description by support vectors. In: Verleysen, M. (ed.) Proc. ESANN, Brussels, pp. 251–256. D. Facto Press (1999)
- [22] Wang, K., Parekh, J., Stolfo, S.: Anagram: A content anomaly detector resistant to mimicry attack. In: Zamboni, D., Krügel, C. (eds.) RAID 2006. LNCS, vol. 4219, pp. 226–248. Springer, Heidelberg (2006)
- [23] Wang, K., Stolfo, S.: Anomalous payload-based network intrusion detection. In: Jonsson, E., Valdes, A., Almgren, M. (eds.) RAID 2004. LNCS, vol. 3224, pp. 203–222. Springer, Heidelberg (2004)