TokDoc: A Self-Healing Web Application Firewall

Tammo Krueger[†] Christian Gehl[†] Konrad Rieck[‡] Pavel Laskov[†⋆]

[†]Fraunhofer Institute FIRST
Kekuléstraße 7
12489 Berlin, Germany
(+49) 30 6392 1870
krutam@first.fraunhofer.de
cgehl@first.fraunhofer.de

[‡]Berlin Institute of Technology
Franklinstraße 28/29
10587 Berlin, Germany
(+49) 30 314 78630
rieck@cs.tu-berlin.de

[⋆]University of Tübingen
Sand 13
72076 Tübingen, Germany
(+49) 7071 29 70574
pavel.laskov@uni-tuebingen.de

ABSTRACT

The growing amount of web-based attacks poses a severe threat to the security of web applications. Signature-based detection techniques increasingly fail to cope with the variety and complexity of novel attack instances. As a remedy, we introduce a protocol-aware reverse HTTP proxy TokDoc (the token doctor), which intercepts requests and decides on a per-token basis whether a token requires automatic "healing". In particular, we propose an intelligent mangling technique, which, based on the decision of previously trained anomaly detectors, replaces suspicious parts in requests by benign data the system has seen in the past. Evaluation of our system in terms of accuracy is performed on two real-world data sets and a large variety of recent attacks. In comparison to state-of-the-art anomaly detectors, TokDoc is not only capable of detecting most attacks, but also significantly outperforms the other methods in terms of false positives. Runtime measurements show that our implementation can be deployed as an inline intrusion prevention system.

General Terms

Security

Categories and Subject Descriptors

C.2.0 [**Computer-Communication Networks**]: General—*Security and protection*; I.5.1 [**Pattern Recognition**]: Models—*Statistical*

Keywords

Intrusion prevention, Anomaly detection, Web application firewall

1. INTRODUCTION

Attacks against web applications pose one of the most serious security threats to modern computer systems. Information portals, social network sites, blogs, content management systems, e-commerce, web email, groupware – these and many other kinds of web applications are accessed by millions of Internet users on a daily basis. Not surprisingly, an explosive growth in the amount of security incidents involving web applications has been observed in recent years [21, 12]. Furthermore, a successful compromise of a web service may be used as a stepping stone for attacking its user community by planting malicious content. Due to a tremendous user exposure, such attacks can be devastating if they are coupled with suitable client-side exploits.

A number of factors contribute to the difficulty of securing web applications. On a non-technical part, the high pressure for rapid time-to-market leads to a dangerous ignorance of security mechanisms, which leaves web applications ridden with security vulnerabilities. The main technical reasons for insecurity in web applications are their inherent openness to arbitrary input, the complexity and heterogeneity of code base as well as the constant extension of their functionality. All these challenges motivate the demand for novel protection mechanisms for web applications.

One of potential mechanisms for improving the security of web applications is a web application firewall (WAF), an appliance for inline monitoring of HTTP communication. Products currently available on the market, such as the Barracuda and ModSecurity WAF, are based on rule-matching. They are capable of filtering entire HTTP requests as well as of selective modification, the so-called *mangling* of certain request fields, according to pre-defined patterns.

Approaches based on pattern matching are, however, prone to two major drawbacks. First, they require extensive application and attack knowledge to maintain a reliable rule base. Furthermore, they are only capable of detecting known attacks and leave a system wide open to the so-called "zero-day attacks" for which no patterns are available. It suffices for a rule base simply to be not up-to-date in order for an attack to slip in.

In this work, we present a method that attains the same level of response flexibility as a WAF – namely, the ability not only to drop but also to heal malicious requests – *without reliance on known patterns.* Our approach is based on anomaly detection carried out at the granularity of HTTP request tokens. In contrast to previous applications of anomaly detection to web attack detection, e.g. [8, 22, 7, 2, 3, 20], our method not only detects, but reacts to such attacks.

We have developed a prototype of a reverse proxy called TokDoc which implements the idea of mangling coupled with anomaly detection. In our prototype, an HTTP request is parsed into token-value pairs and then compared to learned profiles of normal content of specific tokens. Should some

SAC'10 March 22-26, 2010, Sierre, Switzerland.

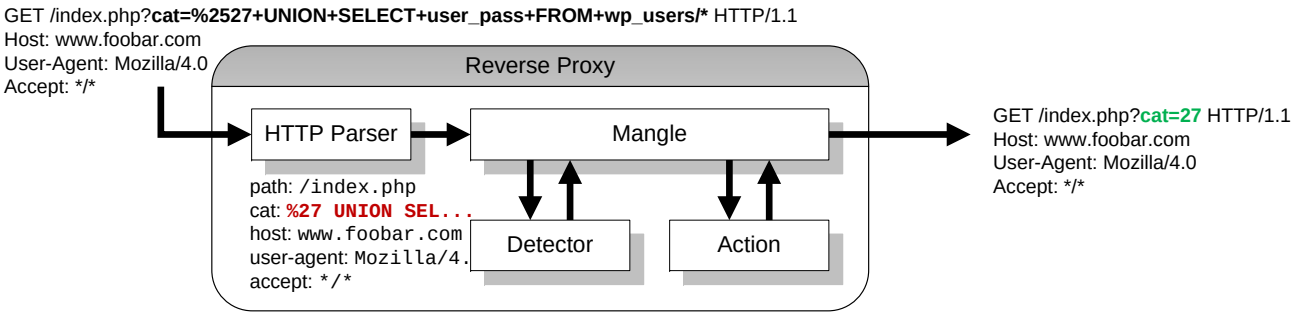

Figure 1: Architecture of TokDoc: The system acts as a reverse proxy, which intercepts each client request. After examination and potential alteration the request is delivered to the production system.

token deviate from a typical profile, it is replaced with an appropriate benign value using token-specific heuristics.
The proposed request healing technique is simple and effective, hence it can be efficiently implemented for deployment in high-speed networks. Its main advantage over simple dropping of requests is that decisions are made in a precise context of specific tokens instead of full requests. This greatly improves detection accuracy, as verified by our experimental comparison with detection at a request level, and makes decisions more fault-tolerant, since the replacement of content with a suitable alternative in certain cases does not harm even if it has been wrongly classified as malicious. In summary, the contributions of this paper are:

- We propose a web application firewall, which decides based on *local, anomaly-based models*, which parts of a request are anomalous and need to be replaced by benign parts observed in the past.
- In addition we employ a *data-driven* setup procedure, which *automatically assigns* data types to extracted tokens both by *structural and statistical features*.
- No "clean", attack-free dataset is needed, since both the learning models as well as the tests are robust against "contaminated" data. No additional attack dataset is needed for the learning and setup procedure.

The paper is organized as follows: In Section 2 we introduce a methodology for a self-healing web application firewall and present our prototype TokDoc. We evaluate its detection performance and runtime using real HTTP traffic in Section 3. Related work is discussed in Section 4 and conclusions are given in Section 5.

2. TOKDOC – THE TOKEN DOCTOR

The automatic healing of malicious web requests entails the following two essential tasks: the identification of malicious content and the construction of the replacement content. Unlike regular intrusion detection in which the problem is to decide whether a request is malicious or not, the identification problem is more complex, since one needs to determine not only the presence of malicious content but also its location which is a prerequisite for replacement.
The identification problem can be addressed by making decisions in a refined context of specific parameters of an HTTP request. Such context-dependent detection has been previously used in [8] for URI parameters of GET requests and in [3] for full HTTP request contents. TokDoc follows roughly the same idea: the requests are parsed into token-value pairs[1], but, instead of combining the scores of token-based models to make decisions at a request level, decisions are made at a token level. Once the content for a token is deemed anomalous, an appropriate replacement is sought.
The diversity of web application traffic content essentially rules out the possibility that a "one-size-fits-all" model of traffic can be learned. Hence we attempt to provide the right anomaly detection algorithm and the appropriate healing action for particular tokens. As a result, the design of TokDoc comprises the following three orthogonal components:

1. *Token Types.* Based on the analysis of real HTTP traffic, we postulate four token types that describe characteristic distributions of token contents.
2. *Anomaly Detectors.* Since malicious content can manifest itself in various features, e.g., unusual length or previously unseen attributes, different anomaly detection algorithms can be used to capture attacks. TokDoc employs a set of anomaly detectors which are automatically coupled with particular tokens in the setup procedure presented in Section 2.4.
3. *Healing Actions.* The particular transformation of a token content is denoted as a "healing action". We propose a set of four healing actions that, similar to detectors, depend on the particular token type and are also configurable during the setup procedure.

From the operational point of view, TokDoc can be characterized as a reverse proxy similar in function as a traffic normalizer [6]. It intercepts HTTP client requests and, after parsing according to the HTTP protocol specification and potentially modifying the request content, relays traffic to actual production systems. In contrast to the system proposed by Valeur et al. [22], TokDoc does not require the target production servers to run at different security levels. The security-related decisions are encapsulated in the proxy itself, which significantly simplifies the design of the network environment to be protected.
The architecture of TokDoc is shown in Figure 1. In the example decision presented in this figure, all but the token cat of the URI parameter are deemed normal and remain unaltered, whereas the anomalous value of cat is flagged for healing. In this example, the value is automatically replaced by previously seen benign data with highest similarity; i.e., the string corresponding to an SQL injection attack is replaced with the benign string 27 due to the occurrence of this string in the attack.

[1]Invalid requests are discarded at this point.

In the following sections we provide a detailed description of the three orthogonal components of TokDoc followed by the presentation of a setup procedure that ties these three components together.

2.1 Token Types

Every request received by TokDoc is parsed into syntactical parts according to the HTTP specification and stored as token-value pairs. We treat the URI path, all parameters of GET and POST requests and all header fields as separate tokens followed by respective values. For example, the request in Figure 1 is parsed into five tokens corresponding to parameters and headers as denoted inside the reverse proxy. Note, that all request parameters with different names will be presented by different tokens.

The distribution of token content is highly diverse. Some tokens, e.g., host, contain only a small number of possible values, sometimes even a constant value. Other tokens such as query and range parameters, contain a wide variety of values, which may even be generated automatically. In order to systematically handle such diversity we classify tokens into four *token types* according to typical properties of information transmitted in HTTP requests:

- *Constants.* In the simplest case the values of a token take the same value, for example the header field host when monitoring a particular web host.
- *Enumerations.* A second type of tokens carries data that takes on only a small set of values dependent either on the HTTP protocol itself or on the web application. An example of such token is the accept-language header.
- *Machine input.* The third type of tokens comprises machine-generated data, such as session numbers, identifiers and cookies.
- *Human input.* The most complex token type is induced by human input, such as free-text fields, query strings, comments and names. The entered data does not exhibit any semantical structure except for being generated by a natural language.

The characteristic features of different token types have to be taken into account in the choice of anomaly detection algorithms and healing actions.

2.2 Anomaly Detectors

Anomaly detection methods have been widely studied for protection of web services [8, 22, 7, 2, 3, 20]. However, all previous approaches flag anomalies for full HTTP requests and hence cannot be directly applied for triggering fine-grained actions on individual tokens. In TokDoc we deploy per-token anomaly detection algorithms as proposed by Kruegel and Vigna [8], however, decision-making remains at the token level.

The choice of an anomaly detection method depends on the token type. For constant and enumeration tokens, a straightforward occurrence check referred to as the LIST detector is a natural choice: if a given value has not been seen in the training data it is deemed anomalous. For the remaining two token types we deploy three different detectors as described below. The decision what detector should be applied to a specific token is automatically made during the setup process presented in Section 2.4.

2.2.1 N-gram Centroid Anomaly Detector (NCAD).

N-gram models have been widely used in security applications [5, 15, 24]. In TokDoc, we deploy the embedding technique proposed by Rieck and Laskov [15], which provides an efficient way for n-gram analysis.

Given the set of all possible n-grams over byte sequences $S = \{0, \ldots, 255\}^n$, we define the embedding function ϕ for a token value x as follows

$$\phi(x) = (\phi_s(x))_{s \in S} \in \mathbb{R}^{|S|} \quad \text{with} \quad \phi_s(x) = b_s(x)$$

where $b_s(x)$ returns 1 if the n-gram s is contained in x and 0 otherwise. The resulting vector $\phi(x)$ is normalized to one to eliminate length-dependence. The vector space induced by the embedding of n-grams grows exponentially with n, however, its sparseness is linear in the length of sequences. This allows one to efficiently construct and compare embedding vectors $\phi(x)$ for byte sequences as detailed in [16].

With the embedding function at hand, the Euclidean distance between embedding vectors can be defined as follows:

$$d(x, z) = \|\phi(x) - \phi(z)\|_2 = \sqrt{\sum_{s \in S} |\phi_s(x) - \phi_s(z)|^2}.$$

Using this distance, the detection can be performed by computing the distance from a previously learned model μ of normal data:

$$\text{score}_{\text{NCAD}}(x) = \begin{cases} \text{normal}, & \text{if } d(\mu, x) \leq t_a \\ \text{anomaly}, & \text{otherwise.} \end{cases}$$

The vector μ is constructed from the training data $X = \{x_1, \ldots, x_n\}$ as an arithmetic mean of the respective embedding vectors $\mu = \frac{1}{n} \sum_{i=1}^{n} \phi(x_i)$. The threshold t_a is determined on an independent validation set as described in Section 2.4.

2.2.2 Markov Chain Anomaly Detector (MCAD)

Markov chains have previously been used in several security applications [8, 22, 4, 20]. We use the 256 possible byte values as states of a Markov chain with 256 possible state transitions each. State transition probabilities can be learned by recording transition frequencies between bytes b_i and b_j in the training data (including an extra start state). The overall size of the transition table is $256^2 + 256$, which is not prohibitively large. Having learned the transition probabilities, we can estimate the probability of a token value x of length n based on the learned Markov chain C:

$$P(x \mid C) = P(X_1 = x[1]) \prod_{i=1}^{n} P(X_{i+1} = x[i+1] \mid X_i = x[i])$$

where $x[i]$ corresponds to the i-th byte in the token value x. We do not use length normalization, for instance by applying the geometric mean, because we want a detector which takes both content and length into account. Equipped with the token-specific Markov chain C and a threshold p_a the MCAD for a new value x is defined as follows:

$$\text{score}_{\text{MCAD}}(x) = \begin{cases} \text{normal}, & \text{if } P(x \mid C) \geq p_a \\ \text{anomaly}, & \text{otherwise.} \end{cases}$$

2.2.3 Length Anomaly Detector (LAD).

Often it is the length of a token value that is characteristic for an attack. For example, the majority of buffer overflow attacks exhibits long token values. This property is

addressed by LAD. This detector is a *fallback position* for tokens, where an insufficient amount of data renders the learning task involved in training the NCAD and MCAD impossible. Therefore we have to find a solution that can cope with a scarce data situation. Modern robust statistics provides us with powerful tools, which are specialized to deal with noisy data and even small sample sizes. Especially for small sample sizes estimates of the mean and standard deviation as used in the Chebyshev's inequality for instance in [8] can be extremely outlier dependent and a test statistic based on these biased estimates can be too loose.
Hence, instead of using a test based on Chebyshev's inequality, we decide to employ a robust statistic as described in [25]. Given a predefined significance level α_{LAD} we estimate the $1-\alpha_{\text{LAD}}$ quantile of the length distribution of the train and validation data L, namely $\hat{L}_{1-\alpha_{\text{LAD}}}$. Now we construct a confidence interval for $L_{1-\alpha_{\text{LAD}}}$ by first calculating the bootstrap estimate of the standard error of $\hat{L}_{1-\alpha_{\text{LAD}}}$, namely $\hat{\sigma}$, and determining the parameter c, so that the interval

$$(\hat{L}_{1-\alpha_{\text{LAD}}} - c\hat{\sigma}, \hat{L}_{1-\alpha_{\text{LAD}}} + c\hat{\sigma})$$

has probability coverage of $1-\alpha_{\text{LAD}}$. Finally we choose the upper bound of the confidence interval as a threshold for the LAD detector to allow for future variability. This results in the following decision rule:

$$\text{score}_{\text{LAD}}(x) = \begin{cases} \text{normal}, & \text{if } \text{len}(x) \leq \hat{L}_{1-\alpha_{\text{LAD}}} + c\hat{\sigma} \\ \text{anomaly}, & \text{otherwise.} \end{cases}$$

2.3 Healing Actions

The fine-grained detection at the token level allows us to devise similarly fine-grained healing actions. Hence our automatic response mechanisms can be less intrusive and more accurate than actions taken at the request level. In particular, TokDoc is equipped with the following healing actions:

- *Dropping of tokens* (`Drop`). The most conservative response to the spotting of an anomalous token value is to remove the token from a request. Notice that this is still a much more benign action than dropping the request itself. We use this action for each token, which has a LAD detector.
- *Preventive encoding* (`Encode`). An alternative but still conservative strategy is to encode the anomalous value using HTML entities. This approach makes common web attacks based on cross-site scripting and SQL injection fail, as control and punctuation characters are escaped. This action provides almost no damage to benign requests, as many web applications can resolve additional encoding of content.
- *Replacement with most frequent value* (`Freq`). For constant and enumeration token types, a natural healing action is to replace the value with the most frequent normal value of the token. This is the natural action assigned to a token having the LIST detector.
- *Replacement with nearest value* (`Near`). The most involved healing action is to replace an anomalous value with its nearest-neighbor from the training set. Such replacement is possible due to the embedding of values in a metric space introduced in Section 2.2. This is the default action for both the MCAD and NCAD. Note that as a side-effect, this action can also correct typos in user-input fields.

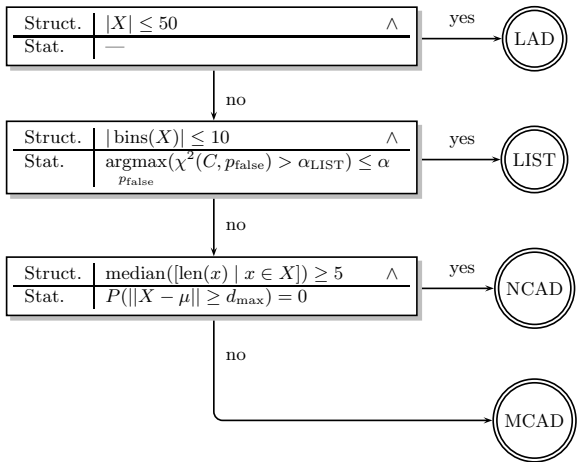

Figure 2: Automatic testing procedure for the setup of TokDoc. After a service-specific split of the training data the testing procedure decides for each token, which detector should be used. By exploiting both structural and statistical features this automatic process is totally data-driven. X denotes the training data for a specific token under test.

Clearly, the four healing actions above are tightly coupled with the particular data types of the considered tokens. The precise assignment of healing actions to token types is presented in Section 2.4, which also allows the administrator to tighten the proposed default actions for special tokens in need of extra protection like password files and cookies.

2.4 Setup of TokDoc

Since the main components of TokDoc are based on learning methods, its setup is dependent on the availability of an *initial corpus of normal data* for training and validation. Initially this sufficiently large pool of client requests should be separated according to services (e.g., by virtual hosts and/or different web services) to allow for service-specific learning of models. This data is parsed and used to generate token-specific data pools used in the following phases. The amount of data should be chosen according to traffic volume so that the widest possible range of normal behavior is covered.
The testing framework depicted in Figure 2 determines for each token an *automatic* and *data-driven detector assignment* by exploiting both structural and statistical features. Using robust, outlier-resistant statistics, this procedure ensures meaningful decisions even for "dirty", attack-tainted datasets. The collected data is split into two equally sized parts: The training pool is used to *learn a model for each token*, for which a threshold is estimated using the validation dataset. After the semi-automatic assignment of actions and outlier adjustment of thresholds for each token, the TokDoc system is ready for deployment. While the model learning has already been discussed in Section 2.2 we now describe in detail the other parts of the setup process.
The data-driven detector assignment is depicted in Figure 2. Each step consists of a structural and a statistical test which is carried out for each token in the original dataset. Starting with a size test, the procedure assigns the simple LAD detector, if the training data of the currently tested token contains 50 or less samples. The rationale here is that all other detectors need a reasonable amount of data for

the estimation of their models. If more than 50 samples are available, the procedure checks, whether the current token is an enumeration. If we observe less than 10 unique values in a token, the procedure tests for statistical evidence by exploiting the well known χ^2-test. First we define the list $C = [d \in train \mid d \in validate]$, which describes, whether each sample of the validation dataset has been observed in the training dataset. Then we can define the function $\chi^2(C, p_{\text{false}})$, which returns the p-value of the χ^2-test, whether C could be generated by a binomial variable, which generates "false" with probability p_{false} and "true" with probability $1 - p_{\text{false}}$. Now we can determine the maximal p_{false}, that barely supports the acceptance of the hypothesis, that C is generated by $p_{\text{false}}$with a given significance level α_{LIST}:

$$p_{\text{worst-case}} = \underset{p_{\text{false}}}{\text{argmax}}(\chi^2(C, p_{\text{false}}) > \alpha_{\text{LIST}})$$

The value of $p_{\text{worst-case}}$ gives an impression of the possible non-matching occurrences for this tokens, that might occur in the future or similarly can be interpreted as the upper bound of the confidence interval of the empirical observed p_{false}. Thus we can use this value for thresholding the expected false-positives per token for the LIST data type.

When deciding between NCAD and MCAD, the test procedure first looks at a structural feature, namely the median length of the token. Since the NCAD detector is based on 2-grams, the detector needs at least two characters for calculating a meaningful mean and distances. If the token passes the structural test, the test procedure focuses on a statistical property: Observe that, given the centroid μ, the largest distance from it is bounded by $d_{\max} = \sqrt{\|\mu\|^2 + 1}$ since the data is normalized to a length of one. By using a kernel density estimator on the validation data (see for instance [19] for details) we can measure and bound the probability, that the maximal distance is ever attained, formally $P(||X - \mu|| \geq d_{\max}) = 0$.

Both the NCAD and the MCAD need a threshold for operation. Since the models are focused on a specific token, we can choose a relatively relaxed thresholding policy. We propose to use the maximal distance for NCAD and minimal probability for MCAD, after a semi-automatic outlier adjustment: All values of the validation data set are ordered by the according output of the detector (descending distances to the mean for NCAD and ascending probabilities for MCAD) and the administrator decides, whether the extremal value is a real, user-generated sample or a malicious token value. During this procedure the administrator additionally can check the quality of the assigned detector and see, whether the chosen model fits the actual data.

In addition he can address privacy and security issues by refining the assigned actions. The administrator can manually adjust, whether a token should be healed or dropped completely. For instance privacy related data such as cookies or passwords *must not be replaced by its nearest counterpart* (`Near` action) but instead *dropped completely* to prevent potential abuse like session or password hijacking.

If the system produces false positives after deployment, these can be tracked down to the token, which caused the false alarm. Thus, the administrator can focus on a specific token and can reconfigure the system according to the incident. In case the website is restructured or new services are deployed, the data model may have to be adjusted accordingly, potentially leading to a retraining of some token models.

3. EVALUATION

Evaluation of an intrusion prevention system is a multifaceted task. Since the effectiveness of response actions inherently depend on the accuracy of malicious content identification, we first evaluate the accuracy of TokDoc detectors and compare its overall performance to other state-of-the-art methods. To check for real-time readiness the runtime of TokDoc is assessed and compared to other proxies.

3.1 Detection Performance

For the evaluation of detection performance we have collected network traces at two different Internet domains. The first data set (FIRST08) comprises 60 days of traffic with 1,452,122 HTTP requests recorded at the web server of a research institute in 2008. The server provides static content as well as dynamic pages using the content management system OpenWorx. The second data set (BLOG09) covers 33 days of traffic with 1,181,941 requests which have been obtained from a domain running various weblogs in 2009. All blogs run on the popular publishing platform WordPress. For the evaluation, both data sets are split into three equally sized parts for training, validation and testing. Due to the different web applications, the amount of monitored tokens as well as the assignment of anomaly detectors differs between the data sets. The TokDoc configuration used for both data sets is presented in Table 1.

In addition to regular network traffic, we have collected network attacks based on 35 exploits obtained from the Metasploit framework as well as from common security archives, such as milw0rm, Packet Storm or Bugtraq. Each attack has been executed in a virtual environment and thoroughly adapted to the characteristics of the two data sets. A detailed listing of the considered attacks and exploits is given in the Appendix. As a result of variations during recording, e.g., usage of different shellcode encoders or SQL statements, the attack pool contains 89 attack instances for FIRST08 and 97 attacks for BLOG09.

3.1.1 Ensemble of Learners and Request Semantics

First of all we want to check, whether all the different detector models are really necessary. For this, we construct special TokDoc instances, which have just the LAD detector instead of both the MCAD and NCAD (referred to as TD_{LAD}), or just the NCAD (i.e. all MCADs are replaced by NCADs, referred to as TD_{NCAD}) or MCAD (i.e. all NCADs are replaced by MCADs, referred to as TD_{MCAD}). We evaluate each of these TokDoc instances on both the FIRST08 and BLOG09 dataset. Each rejected request is manually checked and labeled as false or true positive. In case of doubt a request is replayed against the original server as follows: First we use the unmodified request and save the reply of the server. This is compared to the reply of the server when we send the request modified by TokDoc. If there is a difference, we count this request as a false positive. In the complete replaying process we could not observe any severe or drastic replies from the servers indicating malformed or even malicious requests. This proves, that the inherent request semantic is not harmed by the actions of TokDoc. The results are summarized in Table 2. The first thing to notice is the overall low false-positive rate, which is a direct result of the additional parsing and local decision making of TokDoc. A closer look reveals, that both the TD_{LAD} and TD_{NCAD} suffer from high false-negative rates, while the TD_{MCAD} per-

Category	Detectors FIRST08					Detectors BLOG09				
	LIST	LAD	MCAD	NCAD	Σ	LIST	LAD	MCAD	NCAD	Σ
Header	14	14	5	10	43	22	77	15	17	131
Parameter	9	3	4	—	16	14	166	28	7	215
Path	—	—	1	—	1	—	—	1	—	1
Σ	23	17	10	10	60	36	243	44	24	347

Table 1: TokDoc configurations used in the experiments. The column category summarizes the tokens into tokens originating from headers, parameters from queries and the path token as introduced in Section 2.1.

Dataset	Detector	FP	TP	FN
FIRST08	TokDoc	0.00002	0	0.00000
	TD_{LAD}	0.00000	0	0.02247
	TD_{MCAD}	0.00001	0	0.00000
	TD_{NCAD}	0.00002	0	0.22472
BLOG09	TokDoc	0.00003	212	0.04124
	TD_{LAD}	0.00001	68	0.15464
	TD_{MCAD}	0.00009	186	0.04124
	TD_{NCAD}	0.00003	0	0.22680

Table 2: Detection performance of several instances of TokDoc. FP = false-positive rate. TP = attacks found in normal traffic. FN = false-negative rate.

forms equally good on the FIRST08 dataset but falls behind TokDoc on the more involved BLOG09 data. The plain TokDoc with its diversity of models is the only method, which performs nearly identical on both datasets and is also capable of detecting the most true positives in the tainted BLOG09 data. This trend is further confirmed by Table 5 in the Appendix: All presented detectors are necessary to disarm the used attacks. Note, that the malicious parts of the attacks are spread throughout different tokens rendering the TokDoc approach even more valuable. Additionally, the *healing actions* employed in TokDoc *save roughly 0.0001 of the data* from being discarded as false positives on both the BLOG09 and the FIRST08 dataset. In summary the results show, that just the *full variety of models* embodied in TokDoc leads to an overall *good performance* while keeping the general *request semantic intact.*

3.1.2 Comparison to other Detectors

As a baseline for detection performance, we consider two state-of-the-art anomaly detection techniques using the raw HTTP request payload as input: The *Markov Chain* detector uses a Markov chain as described in Section 2.2.2 over the full content of the requests for anomaly detection. It is learned on the same training data as TokDoc and similarly calibrated using the validation partition. As second baseline, we have implemented a variant of *Anagram* [24]. The detector stores n-grams of benign HTTP requests in a Bloom filter and uses the ratio of unknown n-grams in incoming requests as anomaly score. The detector is calibrated on the validation data, and n is fixed to 2.

The results of our evaluation are summarized in Table 3. For both the FIRST08 and BLOG09 dataset we report the FP_{TD}, which equals the false-positive rate of a detector calibrated to the true-positive rate of TokDoc, and FN_{TD}, which is the rate of missed regular attacks, where each detector is calibrated to the false-positive rate of TokDoc. Focusing on the FIRST08 dataset we see, that both TokDoc and Anagram yield an acceptable false-positive rate, however Anagram is much more porous: nearly 17% of the attacks are

Dataset	Detector	FP_{TD}	FN_{TD}
FIRST08	TokDoc	0.00002	0.00000
	Markov Chain	0.02005	0.80899
	Anagram	0.00004	0.16854
BLOG09	TokDoc	0.00003	0.04124
	Markov Chain	0.16698	0.18557
	Anagram	1.00000	0.39175

Table 3: Detection performance of TokDoc and payload-based anomaly detectors. FP_{TD} = false-positive rate of detector when calibrated to the true-positive rate of TokDoc. FN_{TD} = rate of missed regular attacks when detector is calibrated to the false-positive rate of TokDoc.

not detected. On the contrary TokDoc is capable of detecting all attacks while attaining even a lower false-positive rate than Anagram. The Markov chain is simply overburdened with the FIRST08 and even more with the BLOG09 dataset, where its false-positive rate rises to unacceptable 17%. Surprisingly Anagram breaks down on the BLOG09 dataset: when calibrated to the true-positive of TokDoc, Anagram flags *all* legitimate requests as anomalous. This is due to the fact that 23% of the attacks have an anomaly score of 0, which is the smallest possible score attainable, therefore tagging all incoming requests as anomalous. But even if we calibrate Anagram to the 23% false-negative rate (roughly 8 times higher than TokDoc in this setup), it yields still a false-positive rate of 0.00038, which is a magnitude higher than TokDoc. These numbers clearly demonstrate the outstanding performance of TokDoc both in terms of false positives and negatives even for hard datasets like the BLOG09 data.

3.2 Runtime Performance

To deliver inline intrusion prevention, a system itself has to be reasonably fast, since every client request has to pass the reverse proxy without an intolerable delay. In this part, we subject the TokDoc prototype to a stress test to see whether it can be used in a real-time scenario.

Our prototype is implemented in Python using the twisted framework. This framework provides a mature interface to a number of network protocols. By re-using its proxy module and integrating an optimized n-gram C library into Python, we were able to produce a full-fledged prototype of the TokDoc system. We replay the complete testing slice of both the FIRST08 and BLOG09 (approximately 500k requests each) to get a stable estimate of the processing time. As a baseline, we measure the processing time with Squid as a proxy. Secondly, we consider the ModSecurity web application firewall with a minimal setup of rules to assess, how the additional parsing affects the processing time of a request. Furthermore we use a very simple forwarding proxy application implemented in the twisted framework to see, how

Dataset	Proxy			
	Squid	ModSec.	twisted	TokDoc
FIRST08	1.387	1.536	2.552	2.768
BLOG09	1.500	1.694	2.430	2.902

Table 4: Median runtime in miliseconds of different proxies for both FIRST08 and BLOG09 data.

much the twisted framework itself impose on the processing runtime. Finally, we test TokDoc in the same environment. The median runtime of each setup is presented in Table 4. First we can observe, that the two datasets exhibit different baselines: Generally the FIRST08 dataset seems to have a simpler structure compared to the BLOG09 data. Furthermore the highly optimized Squid and ModSecurity are roughly 1 ms per request faster compared to their Python equivalents. When looking at the inter-application differences, we can observe an increase of 0.1 ms and 0.2 ms from Squid to ModSecurity and 0.2 ms and 0.5 ms from the twisted proxy to TokDoc respectively. This implies, that the additional anomaly detection methods employed in TokDoc just add up to roughly 0.1 ms to 0.3 ms per request. These experiments clearly demonstrate that, while there is still room for improvement in terms of runtime, the anomaly detection methods used in TokDoc are suitable for running in an inline system and that TokDoc even in the current, unoptimized state already can be used as an intrusion prevention system.

4. RELATED WORK

The automatic protection of web applications is gaining an increasing attention among security researchers. Conventional IDS such as Snort [18] and Bro [13], which rely on specific attack signatures or predefined attack characteristics, cannot provide adequate and timely protection against dynamically changing web attacks. Anomaly detection techniques based on payload analysis, for example [15, 24, 10, 14], provide the only possibility for detecting previously unknown attacks. These approaches enable protection of different network services and attain sufficient throughput rates, yet the lack of protocol context in their analysis restricts their use in intrusion prevention to simple dropping or redirection of packets.

First protocol-aware methods for detection of attacks in web traffic using anomaly detection have been proposed by Kruegel and Vigna [8] and extended in ensuing work [9, 22, 17]. The main idea of these methods is the combination of multiple anomaly detectors, such as length checks, byte distributions and Hidden Markov Models, applied to individual URI parameters. Similarly, finite state automata [7] and multiple Markov chains [20] have been recently proposed for detection of anomalous HTTP requests. Our approach differs from all these methods in that it detects anomalous content in *individual tokens* instead of combining token-level anomaly estimates to judge the anomaly of complete requests. Such fine-grain detection enables us to devise novel token healing actions which are much less disruptive than request dropping.

Another line of research combines network anomaly detection with host monitoring. Anagnostakis et al. [1] proposed a system in which anomalous requests are executed in a specially instrumented "shadow honeypot" system. The feedback, whether the request actually harms a system, can then be used to update an anomaly detector, similarly to the work in [11]. In line with this idea, Vigna et al. [23] combine SQL attack detection and a reverse proxy to forward requests to web servers, which manage different levels of sensitive information, depending on the anomaly value of the web request. An SQL query anomaly detector on the host decides whether or not the models for the web request anomaly detector should be updated, if the request results in a malicious database query. TokDoc does not require any additional host instrumentation and serves as a transparent proxy, which greatly simplifies its practical deployment.

5. CONCLUSION AND FUTURE WORK

We have introduced a protocol-aware reverse proxy TokDoc which is capable of deciding – at a token level – which parts of a request are deemed normal and which anomalous. Several intelligent mangling strategies for anomalous tokens, apart from just dropping them, have been described. Experiments on real-world data sets demonstrate the usefulness of the approach and runtime measurements show its readiness for inline intrusion prevention.

While the prototype showed good performance, especially in terms of false negatives, we are aware of several extensions that can improve and extend the system. Practical considerations include the integration of TokDoc into Squid or the ModSecurity platform, which would be a valuable step towards runtime improvement. The coupling of the system with a shadow system as proposed in [1] and incorporation of a feedback loop [11] in combination with learning techniques is another promising extension. Finally, integration of session-awareness and "long term memory" into TokDoc would be an interesting extension, which could be used for flagging user sessions in which a lot of anomalous tokens have been seen as dangerous. This could be used to reliably close down suspicious connections and even track down attacks distributed over several requests.

Overall, the TokDoc system has proven to be a promising, full-fledged web application firewall in the present state, which is capable of effectively preventing and "healing" a wide range of recent web-based attacks. Its runtime performance makes it readily applicable for protection of modern web applications.

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

6. REFERENCES

[1] K. G. Anagnostakis, S. Sidiroglou, P. Akritidis, K. Xinidis, E. Markatos, and A. D. Keromytis. Detecting targeted attacks using shadow honeypots. In *Proc. of USENIX Security Symposium*, pages 129–144, 2005.

[2] M. Cova, D. Balzarotti, V. Felmetsger, and G. Vigna. Swaddler: An Approach for the Anomaly-based Detection of State Violations in Web Applications. In *Recent Adances in Intrusion Detection (RAID)*, pages 63–86, September 2007.

[3] P. Düssel, C. Gehl, P. Laskov, and K. Rieck. Incorporation of application layer protocol syntax into anomaly detection. In *Proc. of International Conference on Information Systems Security (ICISS)*, pages 188–202, 2008.

[4] J. M. Estévez-Tapiador, P. García-Teodoro, and J. E. Díaz-Verdejo. Measuring normality in http traffic for anomaly-based intrusion detection. *Computer Networks*, 45(2):175–193, 2004.

[5] S. Forrest, S. Hofmeyr, A. Somayaji, and T. Longstaff. A sense of self for unix processes. In *Proc. of IEEE Symposium on Security and Privacy*, pages 120–128, Oakland, CA, USA, 1996.

[6] M. Handley, V. Paxson, and C. Kreibich. Network intrusion detection: Evasion, traffic normalization and end-to-end protocol semantics. In *Proc. of USENIX Security Symposium*, 2001.

[7] K. L. Ingham, A. Somayaji, J. Burge, and S. Forrest. Learning DFA representations of HTTP for protecting web applications. *Computer Networks*, 51(5):1239–1255, 2007.

[8] C. Kruegel and G. Vigna. Anomaly detection of web-based attacks. In *Proc. of 10th ACM Conf. on Computer and Communications Security*, pages 251–261, 2003.

[9] C. Kruegel, G. Vigna, and W. Robertson. A multi-model approach to the detection of web-based attacks. *Computer Networks*, 48(5), 2005.

[10] T. Krueger, C. Gehl, K. Rieck, and P. Laskov. An architecture for inline anomaly detection. In *Proc. of European Conference on Computer Network Defense (EC2ND)*, pages 11–18, 2008.

[11] M. E. Locasto, K. Wang, A. D. Keromytis, and S. J. Stolfo. Flips: Hybrid adaptive intrusion prevention. In *Recent Adances in Intrusion Detection (RAID)*, pages 82–101, 2005.

[12] Microsoft. Microsoft security intelligence report: January to June 2008. Microsoft Corporation, 2008.

[13] V. Paxson. Bro: A system for detecting network intruders in real-time. *Computer Networks*, 31(23–24):2435–2466, Dec. 1999.

[14] R. Perdisci, D. Ariu, P. Fogla, G. Giacinto, and W. Lee. McPAD: A multiple classifier system for accurate payload-based anomaly detection. *Computer Networks*, In Press, Corrected Proof:–, 2008.

[15] K. Rieck and P. Laskov. Detecting unknown network attacks using language models. In *Detection of Intrusions and Malware, and Vulnerability Assessment, Proc. of 3rd DIMVA Conference*, LNCS, pages 74–90, July 2006.

[16] K. Rieck and P. Laskov. Linear-time computation of similarity measures for sequential data. *Journal of Machine Learning Research*, 9(Jan):23–48, 2008.

[17] W. Robertson, G. Vigna, C. Kruegel, and R. A. Kemmerer. Using generalization and characterization techniques in the anomaly-based detection of web attacks. In *Proc. of Network and Distributed System Security Symposium (NDSS)*, 2006.

[18] M. Roesch. Snort: Lightweight intrusion detection for networks. In *Proc. of USENIX Large Installation System Administration Conference LISA*, pages 229–238, 1999.

[19] B. W. Silverman. *Density Estimation for Statistics and Data Analysis*. Chapman & Hall/CRC, 1986.

[20] Y. Song, A. D. Keromytis, and S. J. Stolfo. Spectrogram: A mixture-of-markov-chains model for anomaly detection in web traffic. In *Proc. of Network and Distributed System Security Symposium (NDSS)*, 2009.

[21] Symantec. Symantec global internet security report: Trends for July-December 07. Volume XIII, Symantec Corporation, Apr. 2008.

[22] F. Valeur, G. Vigna, C. Kruegel, and E. Kirda. An anomaly-driven reverse proxy for web applications. In *Proc. of the 2006 ACM symposium on Applied computing*, pages 361–368, 2006.

[23] G. Vigna, F. Valeur, D. Balzarotti, W. Robertson, C. Kruegel, and E. Kirda. Reducing errors in the anomaly-based detection of web-based attacks through the combined analysis of web requests and SQL queries. *J. Comput. Secur.*, 17(3):305–329, 2009.

[24] K. Wang, J. Parekh, and S. Stolfo. Anagram: A content anomaly detector resistant to mimicry attack. In *Recent Adances in Intrusion Detection (RAID)*, pages 226–248, 2006.

[25] R. R. Wilcox. *Introduction to Robust Estimation and Hypothesis Testing*. Academic Press, 1997.

APPENDIX

CVE / milworm	Token	Detector
Buffer overflow attacks		
1999-0874	Path	MCAD
2001-0241	Path	MCAD
2001-0500	∅ (Protocol viol.)	Parser
2002-0392	∅ (Protocol viol.)	Parser
2003-0471	Param. User	Normalizer
2003-1192	∅ (Protocol viol.)	Parser
2004-1561	∅ (Protocol viol.)	Parser
2004-1134	∅ (Protocol viol.)	Parser
2005-4734	Param. url	Normalizer
2006-1148	∅ (Protocol viol.)	Parser
2006-0992	Accept-Language	MCAD
2006-5216	Path	MCAD
2006-5478$^{\text{first}}$	Host	LIST
2006-5478$^{\text{blog}}$	Host	NCAD
Code injection attacks		
2005-0116	Param. configdir	Normalizer
2005-0511	Param. template	Normalizer
2005-1921	Body	MCAD
2005-2847	Param. f	Normalizer
2006-1551	Body	MCAD
2007-0774	Path	MCAD
php_inject	Param. z	Normalizer
WordPress attacks		
2004-1584	Body	MCAD
2005-1810	Param. cat	MCAD
2005-2612	Cookie	MCAD
2007-1599	Param. redirect	MCAD
7738$^{\text{milwOrm}}$	Param. thread	Normalizer
2008-1982	Param. ss_id	Normalizer
6842$^{\text{milwOrm}}$	Param. id	LAD
2008-5752	Param. book_id	Normalizer
2009-0968	Param. id	LAD
2009-1030	Host	NCAD
Miscellaneous attacks		
httptunnel	∅ (Protocol viol.)	Parser
2004-1373	∅ (Protocol viol.)	Parser
2007-1286	∅ (Protocol viol.)	Parser
xss/sql_injection	Param. s	MCAD

Table 5: Table of HTTP exploits. Each attack is executed in different variants. We have listed the tokens, in which the attack is located and its detector. In case a token has never been seen before, TokDoc normalizes the request by dropping this token.