Learning Web Application Firewall - Benefits and Caveats

Dariusz Pałka[1] and Marek Zachara[2]

[1] Pedagogical University of Cracow, Poland
dpalka@up.krakow.pl
[2] University of Science and Technology (AGH) Cracow, Poland
mzachara@agh.edu.pl

Abstract. The paper discusses selected issues related to the implementation and deployment of the Web Application Firewall that protects the target application by verifying the incoming requests and their parameters through matching them against recorded usage patterns. These patterns in turn are learned from the traffic generated by the users of the application. Since many web applications, including these operated by the government, are prone to exploits, there is a need to introduce new easily implementable methods of protection to prevent unauthorized access to sensitive data. A Learning Web Application Firewall offers a flexible, application-tailored, yet easy to deploy solution. There are certain concerns, however, regarding the classification of data that is used for the learning process which can, in certain cases, impair the firewall ability to classify traffic correctly. These concerns are discussed on the basis of reference implementation prepared by the authors.

Keywords: web, firewall, learning, security.

1 Introduction

According to a recent survey [27], 72% of the interviewed companies had their web sites/applications hacked during the preceding 24 months. Most of the successful attacks happen on the application layer (Layer 7 of OSI model [12]) - as this is still the most vulnerable area. Even though lower levels are not ultimately secure, over the years the network protocols have been hardened to the point where hacking them is extremely difficult and few vulnerabilities emerge. Also, the operating systems, web servers, and encryption engines are relatively secure, as they usually come from only a few vendors and each has a large enough user base to identify vulnerabilities early and get them fixed quickly.

Web applications, however, are the opposite - they are usually unique (even if they use portions of a common code - e.g. some framework) and usually maintained by or for a single client only. This results in both lower average security of the code (as the specialized knowledge is less available for each company in such fragmented market) and fewer chances to spot the vulnerability before it is exploited. This is why web application vulnerabilities still outnumber browser/operating system vulnerabilities by the ratio of 1:10 [17].

Many web applications have vulnerabilities that are extremely easy to exploit. According to the research by Web Application Security Consortium [30], “More than 13% of all reviewed sites can be completely compromised automatically. About 49% of web applications contain vulnerabilities of high risk level (Urgent and Critical) detected during automatic scanning. However, detailed manual and automated assessment by a white box method allows to detect these high risk level vulnerabilities with the probability reaching 80-96%”.

Unfortunately, governmental web sites and applications are no exception. Although there are no specific statistics available for the vulnerability ratios of such sites, they are subject to successful hacking attempts. For example, a recent news item has reported a large number of these sites hacked with access details available for sale on the black market [10]. It is important to note that governmental sites attract not only regular cyber-crime, but also the attention of hostile governments. With incomparably larger resources that can be committed to find/exploit a vulnerability and also cover the actions, this is a significant threat that cannot be ignored. In recent years there have been a number of accusations of such actions circulating the media, yet for obvious reasons no hard proof has been presented to the public.

2 Common Attack Methods

There are numerous methods in which an attack on a web application can be conducted. Yet, there are a few distinct categories that group most of them. Some of them are explained in greater detail below. These are the types of attack that can be (to various extent) prevented with the use of the dynamic parameter evaluation described in this paper. What is important to note, however, is that the list includes all most common vulnerabilities according to SANS Top 25 CWE list [1]. More in-depth explanation, also reffering to other categories, can be found in [9].

Script injections, especially *SQL Injections* are attack vectors that attempt the execution of an arbitrary code on the target machine. In its simplest form, such vulnerability exists if a user-supplied parameter is inserted into a predefined query and sent to the target interpreter. Vulnerability to this kind of attack is usually the result of insufficient validation of input, which neither sanitizes nor rejects harmful parameter values.

Parameter tampering is a general term for modifying the parameters sent to the server with a request, usually assigning them values not expected by the server. The objective of this action is to push the application servicing the request off its usual working pattern, possibly making it behave in an unpredictable way. Script injections are an example of parameter tampering, but the attack may also center on application/server failure (DoS) or identity theft.

Forceful browsing is a method of directly accessing available server’s resources by requesting specific URL’s instead of following links and forms provided by the application. This way a potential attacker can get access to pages/files outside normal execution path or can bypass poorly designed authentication procedures.

Cross-site scripting refers to a specific code injection technique, where the goal of the attacker is not to attack the server itself but other users that are (or will be) connected to it. If the web application allows for user-supplied content and does not validate that content properly before making it available to other users, the attacker can post a harmful code that can be presented by the server to other users - and executed on their PCs. As a result, the attacker may gain access to confidential information - including users' session identification.

For the sake of fairness, it should be noted that there are also various other threats/attack vectors that are not affected by the methods presented in this paper. The major ones that have to be considered are:

Buffer overflows that exploit stack/heap or other limited memory allocation pools by trying to overwrite the program's data with their own code. Even though they are a serious threat in general, they usually do not affect web applications. Although theoretically possible, they would be extremely difficult to conduct because of primarily two reasons: most web applications are written in relatively 'safe' languages that implement boundary controls, and also the web applications are often customized or custom made - forcing the attacker to act 'blindly'. Because of this, it is much easier to conduct this type of attack against other components of the system - the web server, database, or even a WAF - since these are widely available components that the attacker can investigate prior to the attack.

Man in the middle is a technique that involves listening to (or modifications of) the data in transit between a client and the server. The goal is to either get access to privileged information (e.g. user's credentials) or to change it before it reaches the server (e.g. change the account number for wire transfer). So far the primary protection against this threat is an encrypted transmission with verification of digital certificates. Unfortunately, if the user's operating system is compromised, there is hardly any way to protect them against such threat.

Session hijacking, cookies tampering and other methods that target user-side data in order to gain access to user's resources (e.g. by exploiting the Session ID) are only affected if the attack vector utilizes a cross-site action (as described above). Otherwise, they are unaffected by a WAF. Generally, WAF's task is to protect the server, while the user protection granted by possible rejection of cross-site scripting attempts is an additional bonus.

Denial of Service (DoS) and Distributed DoS (DDoS) are attacks that intent to prevent a web service (a web application) from functioning. This can be achieved by either exploiting a system bug or vulnerability (e.g. shutting down the service by a means of sending special command to the server), or by utilizing system resources (e.g. saturating the bandwidth) to the point where the system becomes unresponsive. While a WAF can protect against the former scenario, little can be done to prevent the latter. Also, in most cases, the protection against resources overloading should rather be done on a lower, network level.

3 Attack Prevention Measures

Most companies live in a delusional comfort of superficial protection. According to CSI/FBI 2006 study [4]: “97% of interviewed companies and administrations were using an anti-virus, 98% have a network firewall, 69% have intrusion detection systems. However ... 65% of these organizations have undergone a viral or spyware attack, 32% have experienced unauthorized access to their internal data and even 15% have suffered from network intrusions”

To fend off attacks, two broad categories of methods are used, namely *prevention* and *detection*. Prevention is the 'first line of defense' - its task is to block harmful attempts from reaching a vulnerable spot. Prevention usually works by blocking certain user's activities on the basis of prior knowledge - considering them either harmful or unnecessary for the use of the service. Detection, on the other hand, is used to identify attack attempts that get past the blockade, usually by monitoring the traffic and identifying certain behavioral patterns. These patterns can be in turn compared to a fixed database of known types of malicious behavior (which is in a way similar to the design of most anti-virus programs), or some heuristics which identify unusual circumstances can be implemented (this is the domain of Intrusion Detection Systems [21]).

Unfortunately, the flexibility offered by heuristic solutions comes at a price of false alarms being raised in unusual, but not attack-related circumstances. A system ability to detect an unknown (undefined) attack pattern is usually a trade-off for a number of legitimate users being reported as abusers. With increased sensitivity of such a system, the number of false positive reactions increases, making it unacceptable to allow it to act (e.g. block certain users from accessing the application) on its own. Therefore, such alerts are usually forwarded to a human supervisor for a decision. On the other hand, lowering the sensitivity, which would allow for the elimination of human supervision, results in a possibly higher number of real attacks passing unnoticed.

Focusing on securing a web application, it must be noted that this is a multi-objective task. It includes secure architecture and data handling, secure implementations and securing of external systems and libraries used by an application (e.g. an external database). It is postulated by many professionals, including Microsoft [7], to treat security as a process spanning through the application life cycle rather than a single task, yet in reality it rarely happens. Corporate management prefer to have 'security/penetration tests' checked off somewhere on the way and to move to the next tasks in a project.

This increases the importance of well-defined specific solutions that would address the security issues, while being understandable and manageable by project managers in organizations. As far as web applications are concerned, the Web Application Firewall (WAF) [28] [29] is probably the most universal solution which limits the risk of exploiting the application, yet it is rarely used. This is due to several reasons:

- Difficulty in configuring a WAF - which needs to cover every aspect of interactions between the users and the protected application - the parameters passed from the user, the logic of the page flow, etc.
- Duplication of protection rules that should be implemented in the application itself. This implies acknowledging that the application may not be secure.
- Constant adjustment of WAF rules as the web application grows and evolves to match the new usage scenarios.

The combination of these conditions, especially the need of constantly adjusting a WAF to mirror the changes in the application, theoretically makes the application developers best suited for the task - at least from the organizational/management point of view. However, this leads to the situation when they are loaded with extra work based solely on an assumption that their core work (the application) is insecure. Naturally, their reaction will be to convince the management that a WAF is not really necessary and it is better to direct the available resources towards the development to achieve better quality and functionality. Personal experience of the authors confirms that WAF adoption is very limited in commercial environments, even for critical systems like Internet banking.

4 Adaptive Web Application Firewall

As it was already mentioned, the complexity and number of rules needed for setting up a WAF is probably the most important obstacle preventing its wider adoption. Therefore, a natural approach would be to try generating a rule set from a sample of legitimate traffic. In other words, to equip a WAF with the ability to learn.

Before discussing the methods of creating WAF rules, it is important to outline how a HTTP request is processed. A typical request looks like this:

```
POST /service/admin/base?p=submit&type=normal HTTP/1.1
User-Agent: Mozilla/5.0
Accept: text/html,application/xhtml+xml,application/xml,*/*
Accept-Language: en-us,en;q=0.5
Accept-Encoding: gzip,deflate
Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7
Keep-Alive: 115
Connection: keep-alive
Referer: http://server.domain/service/admin/login
Cookie: PHPSESSID=12345678901234567890
Content-Type: application/x-www-form-urlencoded
Content-Length: 42

42
email=manitou@some.server&password=try_me!
```

As can be noticed, this request carries four parameters:

- *p* and *type* inside the URL
- *email* and *password* as POST data content.

The request also includes other pieces of important information:

- the target web page: */service/admin/base*[1]
- the previous URL visited: *http://server.domain/service/admin/login*

Storing and analyzing this information for each target page over a period of time can lead to valuable conclusions:

Page flow of the webservice: By comparing the current Referrer URL against the statistics of the Referrer URLs collected for the page, it can be determined whether the current client's behavior matches the usual usage patterns. If the Referrer URL has not been observed before, or has been observed extremely rarely, it can be assumed that the client is not behaving correctly, i.e. can be doing something undesirable. Such request can then be rejected, providing the application with the protection against the *Forceful browsing* exploit scenario.

Parameters range for the page: Having a large enough set of values for each parameter supplied by clients for the page, it can be deducted whether the currently supplied value falls within what can be considered as a 'normal' range. This is a typical classification task [5] with a potential for simplification, as described later. By controlling and rejecting parameter values which do not match normal behavior patterns, the application can be protected against all kinds of *Parameter tampering* attacks, including *SQL injections* and *Cross-site scripting.*

Of course, the effectiveness of these methods rely heavily on the quality of the statistics which in turn rely on the number of data samples collected and the ratio of malicious or erroneous attempts against all events. This issue will be discussed in greater detail in the following section.

5 Concerns and Implementation Issues

Despite obvious benefits of a self-learning WAF, there are still certain issues that prevent market adoption and will need to be addressed. These can be divided into two major categories.

5.1 Data Collection

Apart form purely engineering issues of gathering the HTTP traffic, there are several major issues that have been encountered during the design and implementation of the self-learning WAF. These are related to the amount of data collected, the storage of the data and the application context considered for the parameter analysis.

[1] If an URL parsing module is used (e.g. Apache's mod_rewrite), some parameters may be passed as part of the target URL path.

Data scope. One of the primary decisions that impact a WAF ability to correctly deal with incoming parameter values is the scope of historical (reference) data that is used for evaluation. Too little data and a WAF may reject legitimate values that just did not make it into the reference scope, too much data and a WAF will be a subject to over-training that impairs generalization (similarly to the effect observed in neural networks). The data retention time (DRT), over which the data is analyzed, depends primarily on clients' usage patterns that may e.g. change on a weekday basis. There seems to be no sound scientific model that could help estimate the DRT, leaving it to the experience of the system administrators. It is important to note that the shorter DRT is, the quicker a WAF will adjust itself to a changed application, yet it will also be more vulnerable to an attack that will flood the server with forged requests.

Parameter context. From the user's point of view, a web application (or parts of its functionality) is a stateful machine with its behavior and state depending on user's actions. This implies that in some cases the range or number or valid parameter's values depend on the current state. A typical example could be a money transfer, where a logged user can supply the target account number as a parameter value, but a user who is doing so without being logged will be denied. As with the data scope, the optimal context may depend on specific application needs, yet in most real cases the target page is used as the context, optionally with the referrer URL and/or session ID presence flag.

Data storage. A WAF processes the same information which is sent by a user to the target web application. This means that it also receives sensitive personal data, including authorization data (login names and passwords). However, if a WAF is to have the ability to check the incoming values against the historical knowledge, it must retain the data over a period of time. This leads to a critical issue of storing the values on the server that contains sensitive data. Since a WAF does not know in advance which data might be sensitive, it should apply the same rules to all analyzed parameters. One possible solution is to encrypt all the data stored with a key (preferably supplied from outside a WAF on runtime). The other is to keep all the retained data in RAM only. The former solution puts a tremendous overhead on computational power required as data needs to be encrypted and decrypted constantly. The latter implies loosing all protective knowledge on each server restart - leaving a potential attack window open until enough new data is collected. A practical, yet not elegant solution is to specify parameters that contain sensitive data manually - and only these parameters are e.g. encrypted - or most often an arbitrary value pattern is defined for them by the system administrator.

5.2 Learning Patterns

As far as data collection and analysis are concerned, there are two distinct work regimes that can be discussed. One is a triggered learning (TL) scenario, and the other is a continuous learning (CL) scenario. The TL scenario requires a

WAF administrator to specifically trigger on and off the learning mode. In this mode, parameter values are collected - and are either stored as reference values or possibly generalized into a reference pattern or a value set. After the learning stage is completed, a WAF is switched into 'working' mode where the learned patterns are not adjusted, only the incoming parameter values are compared against the learned knowledge. This scenario has several important benefits:

- There is no need to consider the data retention period size. The data considered for pattern evaluation is all the data gathered during the learning process.
- There is no need to store all historical data. At the end of the learning period, all the data can be transformed into a matching pattern or a value set. This also eliminates the sensitive data storage concerns described earlier.
- A WAF is resistant to attacks targeting its learning process.

However, there are also considerable drawbacks:

- The learning process must be complete; i.e. include all possible usage patterns - and since all these actions must be performed by the staff, it may incur additional costs. However, as most commercial deployment of web applications include acceptance tests - which usually cover all of the application usage scenarios, this work can be utilized to train a WAF.
- A WAF must be manually re-trained after each change introduced to the protected application. Again, in the commercial environment such changes usually trigger some acceptance tests that can be used for this purpose.

To summarize, this scenario is preferable for a commercial deployment, especially of web applications that do not change frequently over time (e.g. Internet banking). It may be difficult to apply to very dynamic web services (e.g. a web portal).

The CL WAF offers somehow less precise protection at the benefit of much less manual input being required. In this case, as described above, the data is collected from user-supplied input over a defined constantly moving time window. The data from the recent window is used to verify incoming parameters. The major issues implied by this scenario are as follows:

- A WAF will only accept parameter values that match recent users' behavior patterns. If these patterns change over longer periods of time, there will be a high number of false positive reports by a WAF
- The firewall may be susceptible to specially engineered attacks that target its learning process (e.g. supplying malicious data slowly over a period of time in order to make a WAF accept it as a valid pattern).

5.3 Statistical Data Significance

The following considerations apply to the CL scenario only, since in the TL scenario all the learning data is considered to be valid and legitimate. Gathered data retained by a CL WAF may, however, include incorrect or even harmful parameter values - since it is collected from unknown, usually anonymous users. For a WAF to function correctly the following assumption must be met: *the amount of incorrect parameter values must be an insignificantly small portion of all the analyzed values.*

The reason behind this requirement is obvious: the significant amount of incorrect data will train a WAF to accept them as valid input. What is not obvious is how to set the level of 'insignificance' for a certain data set. As of now, there has not been specific research targeting the distribution and grouping/classification of web services parameter values, hence there are no clear guidelines. In most cases, this significance level will be set by the system administrator based on his experience, expected traffic and knowledge about the application.

There is, however, a certain characteristic of the parameter values that can be exploited for the benefit of classification. Based on authors' experience, a significantly large number of parameters have values that fall into one of the following groups:

– *Numerical values.* A large portion of users' input is numerical. Moreover, numerical values are used to control the state of the application, to point at certain database records, etc.
– *Selection values.* These are usually tied to users' selection options. As a result, only a very limited number of values is observed.

These two groups are relatively easy to detect using simple heuristic methods. Therefore, adding appropriate algorithms to the parameter evaluation provides significant engineering benefits, although there are still parameters which will not fall into these groups (e.g. addresses), and which are at the same time a popular target for hacking attempts.

5.4 Reference Implementation

A Learning Web Application Firewall has been implemented as a module for the Apache HTTP Server. The Apache HTTP Server was selected because it is free, open source, modular and powerful software. It is also the most widely used web server - as shown by Netcraft [18], Apache is used by over 61% of 312 693 296 sites investigated in the survey. The main components of the proposed Learning WAF are presented in Fig.1. Request Data Validator is an Apache module which registers and analyzes incoming requests from web server clients. It is located in the input filter chain right after the SSL filter which is responsible for decoding incoming SSL requests. The Request Data Validator extracts a request context and values of all request GET and POST parameters from incoming requests. The extracted data are then sent to the Data Collector and Data Validator subsystems.

The Data Collector sends request data to the Data Store, but due to security reasons discussed above, all request parameters must first be encoded. Request data from the Request Data Validator is also checked by Data Validator to detect and prevent web attacks. If a potential attack is detected, it can be logged and

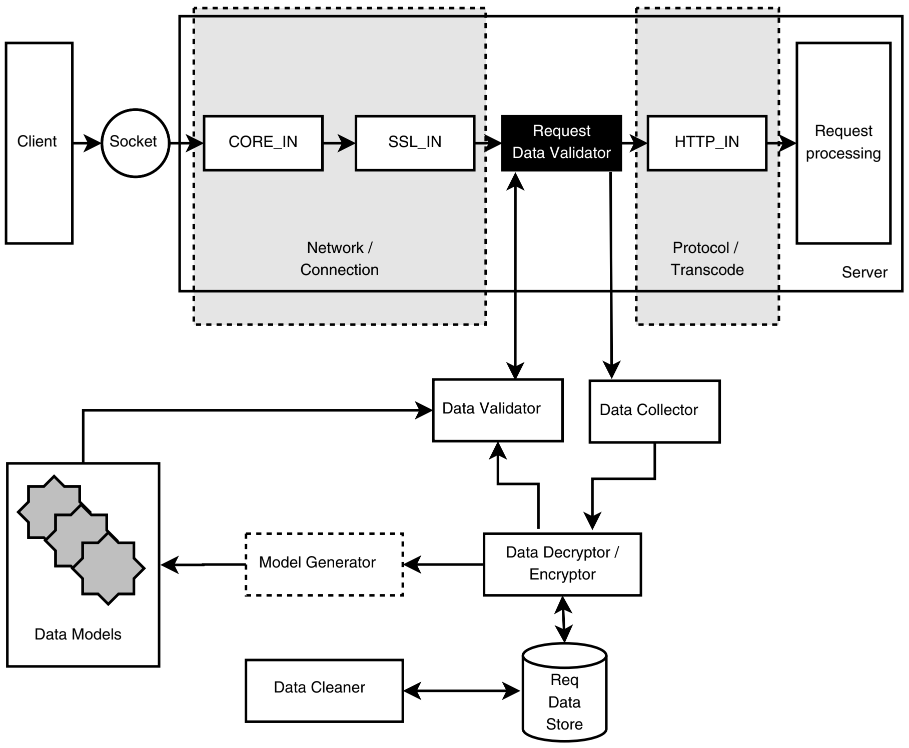

Fig. 1. The Learning Web Application Firewall components

the request may be rejected. Apart from the Request Data Validator module there are also other external processes working with Request Data Store, e.g. Data Cleaner whose task is to remove old parameter values (that are past data retention window). This cleaning process prevents excessive growth of the data store.

The most important of those external processes is Model Generator whose task is to create models for the data collected from requests. If the data in the incoming request do not match the current model, an anomaly score is assigned for this request. The request with an anomaly score that exceeds an assumed threshold is treated as an attack attempt. The Models for the request data are based on the following characteristics:

- the length of parameter values
- the character distribution of parameter values
- belonging to one or more predefined classes (such as numerical value, email address, URL)
- belonging to an enumerated type

The models mentioned above (as well as others) can be found in [14], however unlike [14], the method proposed in this paper can utilize a continuous learning (CL) scenario for creating the models.

In case of some attacks, the models described above are not sufficient to detect anomalies in parameter values; in such cases a model of parameter structure can be useful. To create such model it is assumed that parameter values are words belonging to a language L generated by an unknown grammar G. On the basis of those words (positive examples) grammar inference is conducted and grammar G obtained in this process is treated as a data model. Because of the complexity of the process, it is assumed that in order to describe parameter values, the simplest (in Chomsky's understanding) hierarchy regular grammar [11], [13], [14] is enough. In the worst case however, learning only from positive examples is known to be NP-complete [8]. Because of this, instead of identifying grammar G, other methods are used including: inference of Deterministic Finite Automaton (DFA), Nondeterministic Finite Automaton (NFA) or Hidden Markov Models (HMMs), which might be treated as probabilistic generalization of NFA. There is extensive literature on learning regular languages [2], [3], [6], [19], [22], [25], [26].

Due to a stochastic character of the sample data (incoming requests), it was decided to base the grammar inference on HMMs and use Baum-Welsh algorithm [23]. It should be noted, however, that creating a model based on grammar inference is time-consuming, which means that using it in a CL scenario is limited.

6 Conclusions and Future Work

The article discusses potential benefits of introducing a Learning Web Application Firewall for the purpose of detecting and eliminating malicious users' behavior. Making use of learning processes for the benefit of computer security is recently gaining audience among researchers [14], [11], and, with the growing number and sophistication of the web attack attempts, it appears to be a logical approach to address these problems. The obvious benefit of such a WAF is the ability to fend off a large class of dangerous attack patterns directed at the web application, with a very limited set-up time.

6.1 Comparison against Existing WAFs

Certainly, WAFs are not an absolute novelty. They have been available on the market for several years and there are many vendors (e.g. Cisco, Barracuda, F5) offering their commercial hardware/software products (a larger list of sample vendors is available in [28]). There is also an open source Apache ModSecurity project that can be used to provide that functionality. Yet, all of these solutions are based on fixed rule sets, which need to be configured manually for a custom web application. The vendors often provide templates for typical services or applications, but if a custom made application is to be secured, it usually requires a lot of manual setup and on-going maintenance as described earlier in this paper. On the other hand, commercial solutions, especially those that include vendor's hardware, come with additional features, e.g. load balancing, effective SSL support, and so on.

There are other systems which might be helpful in securing Web Applications, e.g. signature scanning intrusion detecting system (IDS), such as SNORT [24] or Bro [20]. However, such signature based methods require frequent updates of the attack signature databases by security experts and tuning it to reduce false positive alarms. These systems also cannot detect novel (zero day) attacks.

There are also anomaly based systems which treat web servers as generic network services, and in such systems incoming and outgoing data are modelled as a stream of bytes or discrete packets. The anomaly detection can be based on packet headers [15] or on matching patterns in the first few packets of connection [16]. In those systems no rules need to be written and they can detect novel (zero day) attacks.

The Learning WAF presented in this paper resembles the approach proposed in [13] and [14], however, it has been implemented as the Apache module located in the input filter chain (not as a log analyzer as in [13] and [14]), and, as a result, it has access to all parameters transferred through GET and POST. Additionally, it can block requests identified as attack attempts, and not only report detected threats. Another significant difference between these two approaches is the fact that instead of triggered learning used in [13] and [14], our learning WAF can also utilize a continuous learning process.

6.2 Fields of Use

The primary advantage of a Learning WAF is its ability to secure custom applications with none or little extra setup involved. Companies that utilize commonly used products or components may be able to get a pre-defined templates for a WAF that will protect their applications, but organizations that run unique or highly customized web applications need to take care of the protective rule sets on their own.

In such cases, a Learning WAF may be the best option - providing a customized protection without much manual work required. The best environment for such a WAF is either one with high traffic, and/or where complete acceptance tests are part of the deployment. Sample types of organizations that could be considered for this solutions are:

– Portals and other high traffic, frequently changed web sites.
– Banks and institution with high financial exposure to Internet threats (but they also have a very thorough acceptance tests).
– Governmental institutions - last but not least, they almost always use unique/ custom web applications while often not employing security professionals to protect them (with a notable exception of military). With these applications often having access to very sensitive data (e.g. individual tax reports), they are attractive targets for crooks and the enemies of the state alike.

The major concern for the deployment of a Lerning WAF into the production environment is the quality and completeness of the data used to train a WAF before it can reliably be used to control incoming traffic. In case of a TL WAF, the only concern remaining is the completeness of the training data, as all of it comes presumably from a trusted source. A CL WAF, however, poses a much greater challenge, as it requires automated evaluation and classification of unknown data coming from untrusted sources. This does not mean that such a CL WAF would not protect the application, but it must be taken into account that it may not provide a complete protection and, under certain circumstances, might allow malicious attacks to pass through. This type of WAF thus requires further research.

References

1. CWE/SANS Top 25 Most Dangerous Software Errors (2010), `http://cwe.mitre.org/top25/archive/2010/2010_cwe_sans_top25.pdf`
2. Angluin, D., Smith, C.: Inductive Inference: Theory and Methods. ACM Computing Surveys 15(3), 237–269 (1983)
3. Cicchello, O., Kremer, S.C.: Inducing grammars from sparse data sets: a survey of algorithms and results. Journal of Machine Learning and Research 4, 603–632 (2003)
4. CSI/FBI Computer Crime and Security Survey (2006), `http://i.cmpnet.com/gocsi/db_area/pdfs/fbi/FBI2006.pdf`
5. Duda, R.O., Hart, P.E., Stork, D.G.: Pattern Classification. Wiley, New York (2001) ISBN: 978-0-471-05669-0
6. Fernau, H.: Algorithms for Learning Regular Expressions. In: Jain, S., Simon, H.U., Tomita, E. (eds.) ALT 2005. LNCS (LNAI), vol. 3734, pp. 297–311. Springer, Heidelberg (2005)
7. Gallagher, T., Jeffries, B., Landauer, L.: Hunting Security Bugs. Microsoft Press, Redmond (2006) ISBN: 978-0-7356-2187-9
8. Gold, E.: Complexity of automaton identification from given data. Information and Control 37(3), 302–320 (1978)
9. Hope, P., Walther, B.: Web Security Testing Cookbook. O'Reilly Media, Sebastopol (2008) ISBN: 978-0-596-51483-9
10. Imperva Data Security Blog: Major websites (gov,mil,edu) are Hacked and Up for Sale, `http://blog.imperva.com/2011/01/major-websites-govmiledu-are-hacked-and-up-for-sale.html`
11. Ingham, K., et al.: Learning DFA representations of HTTP for protecting web applications. Computer Networks 51, 1239–1255 (2007)
12. ISO/IEC standard 7498-1:1994, `http://standards.iso.org/ittf/PubliclyAvailableStandards/s020269_ISO_IEC_7498-1s_1994E.zip`
13. Kruegel, C., Vigna, G.: Anomaly detection of web-based attacks. In: Proceedings of the 10th ACM Conference on Computer and Communications Security, pp. 251–261. ACM Press, New York (2003)
14. Kruegel, C., Vigna, G., Robertson, W.: A multi-model approach to the detection of web-based attacks. Computer Networks 48(5), 717–738 (2005)
15. Mahoney, M.V., Chan, P.K.: Learning nonstationary models of normal network traffic for detecting novel attacks. In: Proceedings of the Eighth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pp. 376–385. ACM Press, New York (2002)

16. Mahoney, M.V.: Network traffic anomaly detection based on packet bytes. In: Proceedings of the 2003 ACM Symposium on Applied Computing, pp. 346–350. ACM Press, New York (2003)
17. Microsoft Security Intelligence Report, Key Findings, `http://www.microsoft.com/security/sir/keyfindings/default.aspx`
18. Netcraft, Web Server Survey (April 2011), `http://news.netcraft.com/archives/2011/04/06/april-2011-web-server-survey.html`
19. Oliveria, A.L., Silva, J.: Efficient search techniques for the inference of minimum sized finite automata. In: Proceedings of the Fifth String Processing and Information Retrieval Symposium, pp. 81–89. IEEE Computer Press, Los Alamitos (1998)
20. Paxson, V.: Bro: A System for Detecting Network Intruders in Real-Time. In: Proceedings of 7'th USENIX Security Symposium Lawrence Berkeley National Laboratory, San Antonio TX, January 26-29 (1998)
21. Pietro, R., Mancini, L. (eds.): Intrusion Detection Systems. Springer, Heidelberg (2008) ISBN: 978-0-387-77265-3
22. Pitt, L.: Inductive Inference, DFAs, and Computational Complexity. In: Jantke, K.P. (ed.) AII 1989. LNCS, vol. 397, pp. 18–44. Springer, Heidelberg (1989)
23. Rabiner, L.: A Tutorial on Hidden Markov Models and Selected Applications in Speech Recognition. Proc. IEEE 77(2), 257–286 (1989)
24. Roesch, M.: Snort Lightweight Intrusion Detection for Networks. In: Proceedings of 13th Systems Administration Conference, LISA 1999, pp. 229–238 (1999)
25. Sakakibara, Y.: Recent Advances of Grammatical Inference. Theor. Comput. Sci. 185(1), 15–45 (1997)
26. Stolcke, A., Omohundro, S.: Best-first model merging for Hidden Markov Model induction, Technical Report TR-93-003. International Computer Science Institute, Berkeley, Ca (1993)
27. The State of Web Application Security (2011), `http://www.barracudanetworks.com/ns/downloads/White_Papers/Barracuda_Web_App_Firewall_WP_Cenzic_Exec_Summary.pdf`
28. Web Application Firewall, `https://www.owasp.org/index.php/Web_Application_Firewall`
29. Web Application Firewall Evaluation Criteria, `http://projects.webappsec.org/w/page/13246985/Web-Application-Firewall-Evaluation-Criteria`
30. Web Application Security Statistics (2008), `http://projects.webappsec.org/w/page/13246989/Web-Application-Security-Statistics`