

# DAP 3.0

Rev	Date	Author	Changes
0.1	27 Aug 2008	Vasan	Initial Draft

## 1. Introduction

### 1.1 Purpose

There is a debate on the present implementation and future direction that DAP 2.0 (aka Learning Mode) should take. This document attempts to lay down the concepts, fundamentals and facts that will help this debate.

### 1.2 Terminology

**Request Element:** There are, in a very broad sense, the following element types: Parameters, URLs, Cookies and Headers in a request. When we say “an element” or “a request element”, we refer to an instance of any of these element types. For example, a cookie is a request element.

**Application:** An application is in something that gives one set of features, and typically one unit of protection. It need not necessarily be an IP:Port. Using virtual hosting, one IP:port can support multiple applications based on the hostname (For example, svn.barracuda.com and twiki.barracuda.com could be hosted on the same IP:Port). Using URL suffixes is another commonly used method (for example, svn.barracuda.com/viewvc and svn.barracuda.com/twiki are two different applications).

**Configuration Preference:** We distinguish categorize configuration parameters as preferences and rules. Preferences are like conscious choices about how the administrator would like the Web Firewall to behave, whereas rules depict the application behaviour.

### 1.3 Background

Although we keep talking about Learning Mode as a feature, it should be recognized that it is a means to an end from the product perspective rather than a feature by itself. Learning Mode is *not* a panacea or a yellow brick road. It is an instrument to help the administrator automate the configuration, and to help him/her understand their own application.

Of course, the so-called feature has been hyped up by our competitor, Imperva solely for the purpose of comparison against the NetContinuum product, and they have done a pretty good job of beating us down in this respect. It must also be said that the Sales team has not been given enough ammunition to counter such an attack: the complete lack of Learning Mode.

In 5.6, we had DAP (short for Dynamic Application Profiling) which had some problems, notably the difficulty and tediousness of customizing what DAP learns, as also the lack of visibility into what DAP learned.

In 6.1, we attempted DAP 2.0 replacing the original DAP, as a pre-cursor to a full fledged Learning Mode. This was really a knee-jerk reaction to Imperva's blitz against our lack of anything equivalent.

### 1.4 DAP in 5.x

DAP was a completely dynamic learning-cum-protection mechanism. It focussed on looking at responses from the server and gathering information from it, using that information to create rules *in memory* that forbade any deviations.

This applied only to URLs and Parameters (Cookies were covered under Cookie Security,

but did not fall under the ambit of either learning or DAP). When DAP was enabled, the NCs could learn about existence of URLs via hyperlinks, about length and a few other attributes of parameters by parsing forms in the response.

Earlier (pre 5.2) implementations did not differentiate between parameters and URLs, one did not have a choice if they wanted to restrict DAP to Parameter protection alone. In 5.2, DAP was split into URL DAP and Forms DAP (A URL with a query string is treated as a form, although it is part of a URL. This confused many people, but it is actually quite convenient this way).

The main advantage of DAP was that it required little or no configuration and management. It just worked.

The drawbacks, as seen by the field were:

1. Too many false positives due to Javascript enabled applications modifying things that were part of the original form/URL in the response.
2. No visibility into what DAP was doing
3. Exceptions could be made via Policy Wizard, but it was too tedious.

---

## 1.5 Learning Mode in 6.x

---

Learning mode in 6.x inverted two aspects of the model of DAP: profiling the application now focussed on requests rather than responses, and at the same time, learning resulted in persisted rules in the configuration rather than on-the-fly or in-memory rules.

The implementation was pretty much a copy of the Imperva way of doing things.

Now that the new learning brought much better control to the administrator, it also meant that the responsibility of the administrators increased. Now, they need to watch what the system is learning and tend to it continuously. Heuristics and auto-locking were postponed to the next version, hoping we will get some feedback on how the new scheme of things were being received in the field.

The Learning Mode resulted in generation of Profiles, which mapped out and described the application in terms of URLs and Parameters. In addition, Profiles were seen as the complete description of the application, and therefore anything not in the profile was not allowed.

We do not have much feedback from the field, but there are enough reports to indicate that the implementation does not even come close to matching Imperva in terms of effectiveness or in terms of ease of use.

## 2. The Web Firewall: an Analysis

I already said that Learning Mode is a means to an end, the end being effective protection with least amount of configuration. Note that the end is not always “complete” protection, it is many a time just effective protection.

In this chapter, we try and define how to reach the objective of effective protection. We go over the following concepts:

- ◆ Effectiveness of protection
- ◆ Application Profiles
- ◆ Security Models
- ◆ Variability based Categorization of attacks

---

### 2.1 Web Firewall Basics

---

#### 2.1.1 Non-web firewall features

There are many things that the WSF does, apart from the pure Web Firewall function:

- ◆ Traffic Management: Like load balancing, caching etc.
- ◆ Encryption: This is a security feature, but technically not a Web Firewall function
- ◆ Authentication: This also helps in security, but again, not a Web Firewall function
- ◆ Web Address Translation: Hides the internal structure of the network, but does not do much of direct protection against attacks.
- ◆ Cloaking and Data Theft Protection: guards the responses from exposing too much information.

In order to bring focus, in this document, the Web Firewall feature set we talk about is restricted to those things that allow or deny *requests* which are suspected or real attacks. We exclude everything else, in particular, things that the WSF does to the response like Cloaking, which are traditionally a Web Firewall function from a market or product perspective.

### 2.1.2 Effectiveness

The effectiveness of a Web Firewall is measured by two aspects:

1. The higher the number of genuine attempted attacks prevented (denying what must be denied), the more effective.
2. The lower the number of false positives (denying what must be allowed) the higher is the effectiveness.

### 2.1.3 Application Profile

No two applications are the same, and therefore, what is an attack for one application is not necessarily an attack for another. The same percolates downwards to parts of an application. An IP:port can host many domains, each one having static and dynamic content, many URLs, each one accepting many parameters. At every level what is an attack for one is not an attack for the other.

For example one application can handle multiple domains, whereas for the other, this is not true. It is valid for some URLs to accept POSTs, but not others. It is valid for some parameters to contain non-ascii characters, but not for others. And all this is *very* common for web applications, further, due to changes in technology, this variation keeps varying over time quite rapidly.

This nature of Web Applications makes protection from attacks very dependent on the structure of the application, and therefore, the knowledge of the application's anatomy is essential for effective protection. The higher the knowledge, the better the protection can be.

We call the structure or anatomy (which defines various properties of different request elements) the Application Profile. A base level of protection can be offered by Web Firewalls, but very often, the Web Firewall must be supplemented with an application profile to increase effectiveness of protection. The effectiveness is directly proportional to the detail and accuracy of the the Application Profile.

The application profile may be complete (describing all elements) or incomplete (describing only exceptions from a generic Negative security model rule), or dynamic (on-the-fly profiling based on responses).

In general, we will assume that creating and maintaining an accurate profile reduces ease of use for larger applications.

---

## 2.2 Security Models

There are two well accepted security models when it comes to Attack Prevention, popularly known as the Positive and Negative security models.

### 2.2.1 Positive Security Model

This is a paranoid approach, where the stance taken is "Deny unless we know that it is valid". This requires that the the entire application be accurately profiled to arrive at the complete set

of valid requests. Anything else is denied. Even requests that are not attacks, but things that the application normally does not accept (it will probably result in an error) are denied.

This approach is termed as the "best" approach, since it also protects against so called zero-day attacks (attacks that we don't know about yet).

The trade off is that the administrator needs to deal with a possibly very large application profile and maintain it, unless the device can do a very good job of making that it easy to do that. Learning using live traffic is one way to generate profiles, but the effectiveness depends on how well live traffic covers all valid requests, both in terms of validity of every element as well as coverage of all possible elements.

In addition, application changes often require re-configuring and/or periods of re-profiling, and introducing unavailability during these periods.

### **2.2.2 Negative Security Model**

In this model, only known problems are blocked. It is a "looking for bad stuff" approach, based on an "allow unless an attack" stance. For example, injecting an additional parameter may not be valid for a particular post, but if it does not cause a problem on the server, it will not be rejected. In comparison, in the positive security model, the request would have been rejected despite the fact that it is not an attack.

Although less secure, this model causes far less false positives, requires much less maintenance of profiles, and therefore and increases the overall availability and manageability of a Web Firewall.

This model is not to be scoffed at as the inferior one. Although not perfect in its protection, it is quite convenient for many a website administrator to adopt this model. If it were not, there would not be an IPS/IDS market still alive and getting new customers.

Also, this model is not completely painless in terms of profile maintenance. There are going to be exceptions to rules (for example, one particular URL requires that it accepts a huge post), and the exceptions need to be created and maintained. In comparison to the Positive Security model, the number of profile elements that need to be maintained will be far less, since we only save the profiles of exceptions to generic rules.

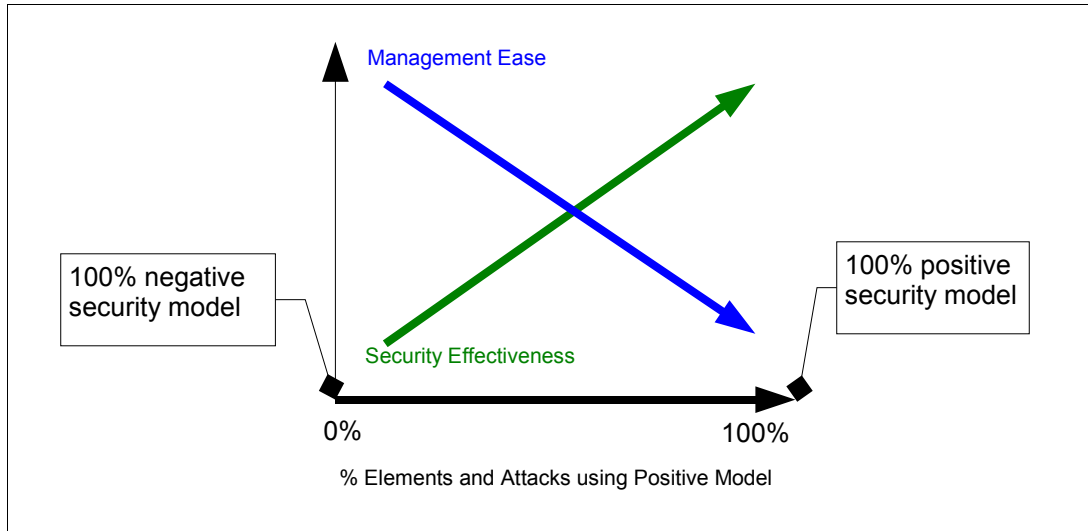
One of the drawbacks of the negative security model that is often talked about is the necessity to maintain and keep updating the vulnerability list, but this is actually Barracuda's forte via Energize Updates.

### **2.2.3 The Choice**

The model choice is almost always a preference setting for an administrator, depending on how they value their security vis-a-vis availability and effort required to manage the positive security model. I will not venture into a hazzarding any guesses about percentages of web site administrators who will prefer one or the other, but let's just say that there are all kinds.

Also, it is not a Yes/No black-and-white decision. There is in fact a wide spectrum of choices based on attack types, because each attack type can be protected using one model or the other, independent of the choice for the other attack types.

The following figure depicts pictorially the available choices for an administrator and what it means to them:



#### 2.2.4 Balancing Act

NetContinuum's Web Firewall has included the ability to choose either negative or positive models, that too at the granularity of every element.

The negative model was effected by Attack Patterns and stringent limits, with a default deny-all stance. This typically resulted in a large number of false positives and the administrator would then create exceptions, and create URL and/or parameter ACLs that applied the positive model only for those URLs or parameters.

The resulting configuration was now very effective in terms of security and at the same time did not sacrifice management ease. The trick was to restrict the Positive Model to only those elements that truly deserved it, and use the negative model for the large portion of elements for which it was good enough.

The only problem was: arriving at this configuration required multiple iterations of looking at logs and creating exceptions. DAP 1.0 did not do much to solve this problem, as the learning was limited to response learning, did not do much to automate the profile generation. Neither did the Policy wizard, due to lack of batch processing, and the need for human intervention.

### 2.3 Attack Categories based on Variability

There are many ways to categorize attacks and group them based on different attributes. For this discussion, I find that it is useful to categorize based on how the validity of a request varies with respect to an application or element.

If the variation is high, that is, there is a higher chance that what is considered valid for one application or element is not valid for another application or element, then the higher is the requirement to supplement the basic Web Firewall with an accurate profile.

The categories are, in the order of least to most variability:

1. Protocol Violations
2. Vulnerability Injections
3. Tampering
4. Exceeding Limits
5. Element Injection
6. Invalid flow

In the following sub-sections, we discuss each attack category and the following attributes of each:

- ◆ Variability, and therefore the applicability of Positive or Negative security model

- ◆ Whether a profile is required, if so what kind (complete/exceptions). Higher the variability, the more difficult it is to create and maintain an accurate profile.
- ◆ The risk of letting through an attack (aka false negative). The higher the risk, the greater is the preference towards a positive security model.
- ◆ Whether a configuration preference makes sense, if both positive and negative security models are applicable, considering the above attributes.

### 2.3.1 Protocol Violations

These are the most straightforward, as the protocol is well defined as to what is acceptable and what is not in HTTP. Some examples are malformed requests, multiple content-length headers etc.

There are a few (very rare) cases that we encountered which required us to relax protocol violations and “allow” them despite not being allowed by the HTTP specifications, due to either buggy user agents or servers. But, by and large, determining whether a request violates the protocol is very unambiguous. Consequently, the positive security model best suites protocol violations, and the WSF does not even have a configuration parameter for these.

The preferred model to adopt here is the Positive security model since the variability is low. Also, a profile is not required to prevent Protocol Violations. There is no configuration preference for the administrators.

### 2.3.2 Vulnerability Injections

Injections use well known signatures or patterns. Assuming that the signatures are well tuned and water-tight, there is little reason to believe that there will be any variations among applications. For example, a string sequence such as 'OR 1=1--' may not actually invalid or cause an attack on the server, but it is unlikely (though not impossible) that it is valid input.

Vulnerability injections apply mainly to parameter values, but they rarely do apply to URLs, Headers and Cookies (in that order of prevalence).

To a large extent, injections can be protected by signatures alone. There are very little variations between applications or parameters, but there may be exceptions.

A positive model for preventing such injections is next to impossible since the profile would need to enlist *all* possible inputs that are valid to use the positive model. The appropriate model for Vulnerability is the negative model, with very few exceptions, which is part of the profile.

Restriction of characters applicable in elements is a positive security stance, but its ability to prevent vulnerability injections is minimal. For example, the character sequence “--” is a possible SQL injection, whereas, a single “-” is not. So, applying positive security model in *addition* to signature scanning is a configuration option for the administrator.

### 2.3.3 Tampering

Tampering applies only to elements that are known to be read-only, and element types that have values. URLs don't have values, but Cookies, Parameters and Headers do. For Headers, there is no “expected” value that can be tampered, so that leaves us only Cookies and Parameters.

Typically, all cookies, and read-only or choice parameters have a limited set of values that cannot be changed. Detection of tampering is done by looking at responses returned by the server and determining the values that must not be modified, tracking them on the request to see that they are not modified.

But, in modern applications, it is common practice to modify even read-only/hidden parameters and cookies via Javascript. Thus, there is wide variability with respect to knowing *which* are the elements that require tampering protection.

The only model applicable here is the Positive security model with dynamic profiling of what are valid values, typically session based.

The application profile will tell us which are the elements that are to be exempt from tampering checks, or it can be looked at the other way: the profile may instead tell us which are the elements to be *exempt* from tampering checks. (This is a profile model choice that we need to make, not a configuration preference for the administrator).

(In 5.6, we had DAP, and therefore, for both Parameters and Cookies, the profile model was based on exceptions. In 6.1, Cookies follow the exception model, whereas parameters are dealt with by a list of parameters that should not be tampered).

#### 2.3.4 Exceeding Limits

The limits of what a certain element can accept (typically count and length of the element) widely varies among applications as well as elements. For example, the length of a URL varies between applications, the size of a file upload depends on the server and the length of a parameter value depends on each individual parameter.

There will be no application where one set of limits will apply to all elements. Limits widely vary, not just among applications, but also among individual elements.

We could split the limits into common limits (larger limits) and per-element limits (accurate limit for that particular element). Invalid requests (those that exceed limits) may not always cause attacks. Those that exceed common limits are likely to be attacks, but those that exceed per-element limits are unlikely to be attacks unless they also cross the common limits.

For limits, one can choose either the Positive security model (where every element's limits are specified in the profile), or the negative security model (use only common limits and create exceptions).

#### 2.3.5 Element Injection

In tampering, we saw how elements can be modified when they are not expected to be. Similarly, elements can also be injected, that is, elements that are not supposed to be part of the request can be added. In many cases, injections are harmless, but in some (such as forceful browsing), it may lead to disclosure of information.

Using the negative security model, it is possible to prevent only known elements that are vulnerable (like `asp:f` header, `robots.txt` and backup files). This does not make much sense for cookies, though.

A positive security model is more appropriate, but it requires one of the following:

1. A complete Static profile enumerating all the valid elements
2. Dynamic Profiling of responses, with exceptions for legal Javascript injected elements

Further, the configuration preference can vary based on each element type. For cookies, it more appropriate to adopt dynamic profiling approach, whereas for URLs and Parameters, the administrator may choose to use a complete profile. For headers, injections are rarely a matter of concern, but we could extend the configuration preference to headers too.

The configuration preference for applying a positive security model is more of an addition to an assumed enforcement of the Negative model.

#### 2.3.6 Invalid Flow

These kinds of attacks are arguably the most difficult to determine whether they are valid or not. These depend on application flow, for example, it may be invalid to access a particular document without first logging in (enforcing entry-control flow), or it may be an attempted brute force attack when a login is attempted many times.

A positive security model requires in-depth knowledge of the application flow, and a profile is usually insufficient, it needs very specific handling of known instances. Due to this, we will not consider Invalid Flow type of attacks in this discussion.

## 2.4 Category Attributes Summary

The following table summarizes the categories and their attributes discussed above.

<b>Category</b>	<b>Variability</b>	<b>Model</b>	<b>Profile Type</b>	<b>Configuration Preferences</b>
Protocol Violations	Low	Positive	Not applicable	None
Vulnerability Injections	Low	Negative and Positive	Negative: exceptions Positive: Char Class	Optional positive model
Param Tampering	Medium	Positive	List of elements	None
Cookie Tampering	Medium	Positive	List of elements	None
Limits	High	Negative or Positive	Negative: exceptions Positive: complete	Negative or Positive?
URL Injection	High	Negative or Positive	Dynamic: exceptions Static: complete	Dynamic or Static?
Param Injection	High	Negative or Positive	Dynamic: exceptions Static: complete	Dynamic or Static?
Cookie Injection	High	Negative or Positive	Dynamic: exceptions Static: complete	Dynamic or Static?

## 2.5 Profile Generation

### 2.5.1 Profile types

Whether the administrator chooses the Negative, positive (dynamic or static), model there is profile generation required to complete the deployment. In case of the Negative security model, the profile generation will result in a bunch of exceptions, as also in the Dynamic profile generation in the positive security model.

The Positive security model requires a complete profile in most cases, specifying the attributes of each element.

In all cases, though, there is some effort required to get the right and complete profile, and this requires some level of automation to ease the pain of deployment and management of the Web Firewall. Note that we consider exceptions also as a kind of profile that needs automation, it is not just the positive security model that needs automation.

### 2.5.2 Generation Mechanism

Various mechanisms are possible, including the following:

- ◆ Manual entry
- ◆ Predefined profiles (templates) for known applications
- ◆ By processing logs as a result of live traffic (Policy Wizard)
- ◆ Automatic, transient, based on responses (DAP)
- ◆ Automated persisted learning from live or simulated traffic (Learning Mode)<sup>1</sup>

The WSF (and many other Web Firewalls) do support the ability to key in rules for protection. Templated profiles for known applications is not that popular, but it is one possible differentiator for the Barracuda Web Site Firewall.<sup>2</sup>

<sup>1</sup> One could also argue that determining a template profile automatically based on the traffic we see is another mechanism. Although that is true, it is more of a “cool” feature than something that will add significant value. It is probably much simpler and less error prone to just ask the administrator a few questions on the technology used in the application.

<sup>2</sup> According to Steve Pao, we will not go head-on against Imperva and fight them with their forte, which is Learning Mode.



## 3. Conclusion

---

### 3.1 Common Misconceptions

#### 3.1.1 Learning mode is a feature

This is far from the truth. Learning mode is one possible mechanism for automating profile generation. It is not the only one. It is only a means to an end: the end being ease of deployment by automatic and accurate profile generation. The way the profile is generated itself is based on configuration preferences for the attack types.

In particular, taking the Policy wizard forward and requiring less human intervention is on very viable options. It may also turn out that different mechanisms are appropriate for different attack types.

#### 3.1.2 Learning Mode and Positive Model

Till now, we assumed Learning Mode applies only to the positive security model. But in fact, it can apply to the negative security model too, to create exceptions to generic rules. Learning Mode is an automation mechanism to arrive at the profile, but not necessarily the complete profile which is only required for the Positive Security model.

In all cases, there is some amount of human intervention or heuristics required to separate out the traffic that is valid from the traffic that is invalid. But apart from that, we should focus on getting the configuration right, the profile model right, and then adopting the most appropriate mechanism for automating profile generation.

---

### 3.2 Decisions to make

Now that we have some perspective on what the Web Firewall is all about, we need to make some key decisions on the following:

1. How to structure the configuration, mainly with respect to the profile. Should we make it a list of exceptions from a generic rule, or a list of attributes only (See the discussion on Parameter Tampering)?
2. How to structure the configuration preferences: One button for a global Negative/Positive choice, or one per attack type?
3. What kind of generation should we adopt: response based, request based, dynamic or a combination of these, or a selection for each attack type?

---

### 3.3 Design

I am not proposing any design in this document. We need to put our heads together, talk about various options, priorities, feasibility, requirement and then arrive at a design.

But my recommendation, looking at the effort required and our positioning at the lower end models, is the following:

- ◆ Extend Policy Wizard batch mode to automate, rather than re-introduce the Learning Mode.
- ◆ Separate out the configuration preferences and make them explicit. The Policy Wizard automation will consider these preferences and act accordingly.
- ◆ Adopt a Negative security model for most attack types.
- ◆ Do not bring back DAP.

(TODO: justify the above).

---

Instead, our market segment will prefer good defaults and many great templates.