

Learning In 7.X

Rev	Date	Author	Changes
0.1	Oct 4, 2008	IVShekar	Version 1: Derived from discussions on DAP 3.0 document

Contents

1.	Introduction.....	2
1.1	Purpose.....	2
1.2	Scope of the Document.....	2
1.3	Overview.....	2
2.	The approach.....	4
2.1	Goodness and Pitfalls of earlier attempts.....	4
2.2	The Direction henceforth.....	5
2.3	Unlearning.....	9
2.4	Enrichments.....	9
2.5	Implementation of the above ideas.....	10
2.6	Summary.....	10
3.	References.....	13

1. Introduction

1.1 Purpose

The purpose of the document is to lay down the plan for a renewed approach of learning which would be implemented over the next few releases based on the feedback and drawbacks from the previous implementations of Learning in 5.x and 6.x releases of NC web-firewall

1.2 Scope of the Document

This document extends the ideas laid down in the document DAP 3.0 by Vasana and outlines the approach we decided to take after the internal discussions on that document. The document assumes the reader is familiar with the WSF configuration model., and the terminologies used both in the context of WSF protection and learning, and except for a minimal explanation of certain key concepts, it does not elaborate on the basics of a WSF (viz., the idea of profiles, default protection, templates, exceptions etc). Please refer DAP 3.0 document for some basics of the WSF model of attack prevention.

The document is expected to grow into a PRD over the next few days/weeks. Please note that this document outlines the direction we wish to take, and does not get into specifics like “honoring learning based on trusted hosts, or return codes” or specifics about heuristic numbers etc..

1.3 Overview

An effective WSF is supposed to block genuine attacks without creating false positives. This would be the definition of “efficacy” that we would be using in the document here.

Any attempt to “learn” and hence automate the protection should keep in view:

- (a) The ways to enrich the automation of configuration to increase effectiveness of the WSF
- (b) The existing architecture of the WSF in terms of enforcing rules and policies.

This is quite important since there has been consensus in the fact that our learning would not be on the footsteps of Imperva. Parsing and learning through responses, pushing out signatures via attackdefs dynamically, the concept of manual and highly granular controls through profiles and extended match concepts are some of our strengths and our approach has to be based on exploiting these strengths.

1.3.1 Brief about positive and negative security models:

Please refer DAP 3.0 for a detailed explanation. Just to recapture in a nutshell., a positive

security model would mean 'deny unless allow' stance which is the best security stance the admin of a WSF can take. Unfortunately, this stance mandates a complete understanding of what is to be allowed and what is not to be and may potentially need a very elaborate configuration which in turn affects manageability.

The negative security stance, on the contrary, is a "allow unless you know its an attack' stance., and while it does not provide security to the extent a positive model does, its the most preferable for the admins who are not very paranoid and are looking for a manageable set of rules with decent protection. But this stance may actually result in "false negatives" or genuine attacks which bypass the defaults, making it to the backend.

1.3.2 Our stance so far on 7.0/7.1

We have been shipping the WSF with a negative security model for all types of attacks as the default configuration. This stance is the most effective for SMB customers who would want availability of the services along with zero day protection from sql, xss and reconnaissance on their services. But since no configuration fits all applications, from the deployments we had so far on the field., the following false positives are commonly seen. They have been categorized for convenience here.

Category: Buffer overflow protection

Parameter length overflow false positives. Relaxing the limit checks or adding the parameter to the exception list is the remedy.

Content length overflow: The default 32K is not enough for few POSTs which may have files to be uploaded. Remedy was to relax content length under default-url-protection.

Category: Forceful browsing

Content-type: We allow few content types by default and any other content type which causes false positive has to be added to the default list.

Methods: GET, POST are the defaults and

Category: Vulnerability injections:

False positives on sql, xss and cmd injection patterns

Category: Tampering

False positives on cookie replay protection and "unreognized cookies"

It is clear that even in the case of SMB markets., to increase the effectiveness of the website firewall, we need to "learn" these false positives and create a conducive configuration to minimize the loss of availability of the application while effectively preventing genuine attacks.

2. The approach

There are primarily two areas associated with the learning or automation of the WSF configuration

- What elements need to be protected
- Ways to learn these elements

2.1 Goodness and Pitfalls of earlier attempts

Although its already captured in DAP 3.0 document, here is a brief of the earlier approaches to increase the effectiveness of NC firewalls

2.1.1 Dap in 5.x versions

The first version of dynamic application profiling concentrated on learning various elements from responses and creating rules in memory, thereby resulting in a strict positive stance. It did a good job of preventing forceful browsing and also session based enforcements of parameter tampering and forceful URL browsing with minimal configuration. However, it fell short in three counts:

- (a) Response based learning caused false positives for client generated or modified urls and parameters, which is not rare in javascript world
- (b) There was no visibility on the in memory rules due to architectural challenges,
- (c) There was little or no learning of the parameter class that a parameter belongs to, and so all “input” parameters would still have to go through the default vulnerability injection checks and a manual intervention in creating exceptions

2.1.2 Dap in 6.x versions

The version in 6.x tried solving each of these problems, it was designed to overcome the above shortcomings by

- (a) learn all elements of an application by both response and request and enforce policies, thus creating exception for client generated/modified elements and
- (b) providing visibility to the admins on what is learnt through the concept of profiles.

Nevertheless, this version introduced some important pitfalls like

- (a) It had no middle stance. It would enforce a positive security model when selected. Anything not matching a profile would be disallowed. This resulted in potentially huge configurations which is sometimes unmanageable. Techniques to optimize the profiles were proposed but not implemented since it is complex to optimize.
- (b) Learning through requests was designed in such a way that the admin has to manually lock the profiles to turn to active mode after the admin “feels” satisfied with learning. This approach does not help in “incremental learning”, which was the forte of 5.x model. Also since no amount of laboratory tests would cover the entire web application,

the efficacy was also questionable since there indeed will be false positives even after considerable “learning” time.

2.1.3 The new approach

The new approach we propose here is aimed solving the pitfalls of 5.x and 6.x models by rethinking the way we present the configuration and also combine the goodness of the 5.x and 6.x models. The primary problems to solve are:

- Build in flexibility to NOT have a firm positive or negative stance at the whole WSF level protection. A positive or negative stance at the attack category level makes more sense depending on the preferences of the admin on his preference of manageability versus paranoia. This solves one important problem of 6.x, that “once you choose a positive security model., the profiles created are heavy weight, extensive and are unmanageable”.
- Explore the heuristics way to automatically transition to the “locked” mode. This was not possible earlier since we did not think of a split positive and negative model. This “split” gives us the flexibility to automatically “lock” and transition to active mode for some attacks. The concept of “active learning” thus will be satisfied to an extent for evaluations like “ICSA”.
- Bring out the visibility of session based learnt rules since we now can deal with the file system unlike in 5.x
- Learning and automation ways (like session based learning) and the concept of “positive and negative’ models can be fit into the “feature matrix” and packaged as high end features to the 860/960 models
- A way to “unlearn’ rules (perhaps a UI button to that extent), and granularity of choice to the admin of what to “unlearn”.

Also, the scalability of the solution with respect to both Barracuda models and the implementation is also kept in view in proposing the ideas.

2.2 The Direction henceforth

We need to answer four important questions for each attack category type as we move towards the new direction.

Q1 - The Model: Positive model (less manageability) or negative model with exceptions (less nodes, but relaxed)

Q2 – Learning Approach: How do we learn (heuristics using log analysis or live profiling, templates, dynamic learning like in 5.x or code base of 6.x)

Q3 - Config Translation: How does it translate in terms of configuration

Q4 - Locking: Is it active from day one or when does it turn active

The reason for false positives is the “variability” factor of the policy, ie., a policy which holds good for one element does not or may not hold good for another element. Higher

the variability, less effective the negative security model is, more the exceptions are and hence less is the manageability. Let us now discuss the various attacks WSF blocks and the approach we would take with respect to increasing the efficacy as defined in the section

2.2.1 Protocol Violations: Protocol violations are basically triggered due to requests which are non-conformant to the HTTP 1.0 and 1.1 protocols and the HTML specifications

Q1 - The Model: The variability is low since clients are mandated to follow HTTP protocol, hence a positive security model that we have today is good enough to protect such attacks. In other words, there's little to "learn" and "create exceptions for".

Q2 - Learning Approach: Parsing engine automatically takes care of the violations. There is scope for "relaxing" however through action policies

Q3 - Config Translation: No configuration except the action policies

Q4 - Locking: Active from day 1

2.2.2 Cookie tampering: Failure to decrypt the external cookies, failure to find an encrypted pair for an incoming cookie, replay of cookies from unauthorized IP and headers and expired cookies all come under the purview of this

Q1 - The Model: Positive stance of disallowing all violating cookies unless an exception is created. This is our default stance anyway.

Q2 - Learning Approach: We need to learn the exceptions. This has not been attempted in 5.x or 6.x. We do get a lot of false positives especially on unrecognized cookies. In 7.1 we introduced the concept of "allow unrecognized cookies for 7 days since the creation of service". This was to reduce false positives on cookies coming without their encrypted equivalents as soon as a WSF is deployed. We can indeed go ahead and learn the cookies expected by the backend and generated by Javascripts (like x_proto_type in our Barracuda GUI) add them to the exempt list. Heuristics (a configuration with defaults like more than 5 instances of mismatched IP from 5 different sources) would help us in learning such cookies and also avoiding false positives due to cookie replay protection.

Q3 - Config Translation: Configuration would remain like the present cookie security config

Q4 - Locking: Cookie replay protection and Unrecognized cookies will not be active till heuristics lock the config.

2.2.3 Exceeding limits: Basically these are buffer overflow attack mitigation rules

Q1 - The Model: Option will be exposed to admin to go for negative model or positive model.

Q2 - Learning Approach: Negative model relies having a default set of limits., which can be either fixed across services or created based on intelligent templates (please refer section). We would learn the exceptions based on one of these methods: (a) Heuristics which will be configurable and either policy wizard or a profile agent like in 6.x would

digest the heuristics to create exceptions (b) session based learning of max limits and creating the profile element if a “max-limit” is seen less than the generic value. We can leverage the 6.x code base here. And the option to create exceptions from learning responses can be a 860 feature.

When positive security model is chosen, we would use the 6.x code based to learn exact elements and their limits as in 6.x.

Q3 - Config Translation: 7.1 introduced the “middle stance”, viz., do not enforce strict profile check and use default protection when a profile is not defined. A negative security model would have “strict profile check=no” and result in creation of learnt exceptions under application profile. A positive security model would be like in 6.x where the flag “strict profile check” is “yes” and profiles are created (but we can only expose the limits section and apply default protection (or positive model again depending on that attack-category’s admin preference) for others parameters like “allowed-methods, attack-types”. The UI should be worked to display the necessary stuff only.

Q4 - Locking: In a Negative security model, the transition to active would happen after heuristics are locked. In a positive model., we may go with the present 6.x way (manual locking)

2.2.4 Vulnerability Injections: These include well known attacks patterns for sql injection, cross site scripting etc.

Q1 - The Model: Option for the admin for negative and positive model. Negative model will be akin to our present default parameter protection and signatures which can be updated through attackdefs. This is our forte and should be default for 360/460 models.

Q2 - Learning Approach: Negative model relies on a set of defaults with “presumably” water tight signatures. Exceptions can be learned again by (a) Heuristics which can be provided as config options with defaults as in the case of limits and can be enforced by either automated policy wizard or the live profile code base in 6.x. We also need to have a mechanism to turn off a “selected pattern ” unlike turning off entire “pattern group” in 7.1. Positive model can be on the lines of 6.x where we learn the parameter class and “input types” and “allowed meta chars’ to make sure we “deny all patterns except those matching the input type”

Q3 - Config Translation: Configuration model will be similar to the description in the “limits” section above for negative and positive security models.

Q4 - Locking: Same as the q4 in limits section

2.2.5 Tampering (Parameter): This pertains to protecting read-only, session choice, global choice parameters

Q1 - The Model: Tampering requires a positive model and no choice for the admin. The biggest risk of positive model here is the client modified/generated parameters and values. This is where the importance of exceptions comes into picture.

Q2 - Learning Approach: Learning would be based on responses and requests, much like in 6.x code base. 6.x has the capability to “learn” hidden, file-upload, read-only, and

choice params on a per-session basis*, and also mark a parameter as “INPUT” if it learns via the requests that the parameter can be “Client modified”.

Q3 - Config Translation: Configuration model would be the same (internally to stm) as the existing profiles., except that the specific parameters learnt as choice, hidden, file-upload would have to be displayed in the UI in an optimized way.

Q4 - Locking: Manual locking might be required if we leverage the 6.x code base.

2.2.6 Forceful Browsing - URLs: Multiple things fall under this category. Prominent are the forceful URL browsing, uploading non permitted filetypes., disallowed methods and content-types, exceeding permitted instances of parameters. This section focuses on URLs

Q1 - The Model: For URL browsing, Option should be given to the admin.

Q2 - Learning Approach: A negative security model would be to allow all URLs and create exceptions via global and local acls. Templates would be a good way to approach this problem, except that we have a limited set of templates today (only for sharepoint, owa apps). Learning can be based on heuristics again.

Positive security model would be to learn the urls from responses and requests, leveraging code for generation of url-profiles from 6.x. Optionally, session based learning of URLs which was one strong area in 5.x should also be integrated into the system for 860 and above models as enhanced security.

Q3 - Config Translation: Negative model would be with acls as outlined above. Positive model would populate “profiles” but the UI would only show up the list (unlike the display of entire profile element in 6.x)

Q4 - Locking: Negative profile locking would be triggered on heuristic matches. [TODO: Think about the domain match]. Positive model needs manual locking.

2.2.7 Forceful browsing – Parameters: This falls under the category “unwanted parameters in a URL”. [TODO: think about mandatory params for a url]

Q1 - The Model Positive model is the only option.

Q2 - Learning Approach: There are two approaches here. We can bring back the “blacklisted” parameter concept and create a * parameter as “black list” and learn exceptions on what is to be allowed, or just go the 6.x way and create parameter profiles much like the above described url-profiles without the baggage of other definitions.

Q3 - Config Translation: Same as existing profiles except that the UI would be intelligent to display only params learned under a url.

Q4 - Locking: Manual locking of profiles.

2.2.8 Forceful browsing – Others (allowed methods and content-types, max-instances)

Q1 - The Model: Choice for the admin

Q2 - Learning Approach: For negative security model, learning again can be based on configured heuristics and via automated policy wizard or live profile code base of 6.x.

Q3 - Config Translation: Configuration for negative model is like the default protection today. Learnt exceptions can be added as outlined in section for limits for both negative and positive models.

Q4 - Locking: Locking happens when heuristics match for negative model and would be manual for positive model.

2.3 Unlearning

Is still a manual option by the deletion of the learnt rules. We may expose the unlearn option on a per attack category basis which could not have been possible earlier with 6.x approach. Thus one can unlearn the learnt session parameter “profiles” while the WSF still operates on the, say, limit exceptions.

2.4 Enrichments

In the past, there have been a few Feature requests and enhancements which haven't been addressed. Though this document is not to address them, this is an attempt to capture them since learning may play a role in such problems:

- **Parameter associations:** Certain applications require a particular parameter to be of a certain attributes based on another parameter. In 6.x we have introduced the concept of extended validations, but it has been of a limited definition and has not been used on the field.
- **Parameter Arithmetic:** There have been instances to limit the range of a parameter's value to some values. Or that of a parameter's value being less than a defined number (like in bank accounts where the value cannot exceed a number) etc.
- [To be extended with other Feature requests from Product marketing]

2.4.1 Learning the application structure

This may be a completely different and independent exercise. One of the reasons why we defined profiles and the approach of displaying each and every profile in 6.x is also to achieve visibility, so that the admins would get to know the components of their application. But the mistake we did in 6.x was that the requirement to enforce policies and the requirement to have visibility were coupled together. This resulted in a potentially huge and unmanageable profile list that may turn off the admins.

With more and more vendors of WAF opting for third party tools like Cenzic for crawling the servers and displaying the structure of the web applications., we also need to go this way. The visibility problem of 6.x would then be achieved without the tight coupling of enforcing the policies versus showing the backend application structure.

2.4.2 Learning the type of the application at the backend and Using intelligent templates:

We do have the concept of templates in one way in 7.1 releases. We have the default security policy and security policies with respect to OWA, Sharepoint and Oracle applications. And the ability to bind and use the appropriate template.

But there are two major deficiencies here:

(a) The policies are not extensive and we don't have them for many other popularly used applications. This needs some research to understand these applications and come up with some default policies.

(b) A security policy is a 'service' level binding, and as the services are defined by the VIP and Port, we may not be able to handle the security policies with respect to multiple kind of applications residing on a virtually hosted machine. To handle this we need to bring back the concept of VAPP which was discarded in 5.x releases of NC. This coupled with url-translations and rule groups will help us define defaults on a url space which defines a virtual host on the backend. Learning/Automation of configuration will also be affected by this paradigm.

2.5 Implementation of the above ideas

The above ideas are an outline of the future course that the product must be taking in order to achieve maximum efficacy with minimal manual intervention. We need to implement the ideas in a phased manner over the next few releases. As mentioned earlier, this document should evolve into a PRD over the course of time once the ideas are thoroughly discussed and a consensus is reached. The approach lends itself to a scalable and phased implementation. For the immediate low hanging fruits for the SMB market, we can implement the negative security model with exceptions for limits and injection attacks in 7.2 release. The 7.3 release can focus on the rest of the learning paradigms. The crawler for auto discovery of the applications can be brought in by the subsequent release timeframes. And the enhancements of the templates can be a parallel effort.

2.6 Summary

The following table summarizes the discussion in the previous sections:

Category	Model	Learn from (Implementation from STM point of view)	Locking	Supported In	Attempted release
Protocol Violations	Positive	None, but action policy to disable	Active from day one	360 and above	Already In 7.1, exposing the non selectable option in 7.2
Cookie Tampering	Positive and exceptions	Learn JS and client cookies from Heuristics	Active from the day heuristics are met (Auto active)	360 and above	Attempt in 7.2
Limits	Negative Positive	Heuristics 6.x and response learning	Auto Active Manual	360 and above 860 and above for resp based	Attempt in 7.2 7.3 and later
Vulnerability Injections	Negative Positive	Heuristics 6.x way	Auto active Manual	360 and above	7.3 and later
Parameter Tampering	Positive	6.x way Combination of 6.x and 5.x for incremental learning	Manual Auto due to 5.x approach	360 and above 860 and above	7.3 and later

Forceful browsing - URLS	Negative	Learn allow exceptions for global and local acs from Heuristics	Auto Active	360 and above	7.3 and Later
	Positive	6.x way 5.x for session based learning for URLS	Manual Auto due to 5.x approach	360 and above 860 and above	
Forceful Browsing - Parameters	Positive	6.x way 5.x for session based learning	Manual Can be auto	360 and above 860 and above	7.3 and Later
Forceful browsing – Others	Negative	Heuristics and defaults	Auto active	360 and above	7.3 and Later
	Positive	6.x way	Manual		
Application structure discovery	NA	Third party tools	NA	860 and above	7.3 and Later
Learning Application Type	NA	VAPP concept and understanding the application technology from the traffic (Eg: CFID cookie indicates its a cold fusion application)	NA	860 and above (these may require the vapp concept)	7.3 and Later

3. References

The following documents have been referred:

- DAP 3.0 : Based on which the new approach has been laid out
- Section 2 of PRD-Learning.sxw (Source: ncsw/doc)
- Alternatives in Alt-Learning.sxw (Source: ncsw/doc)