Pragmatic WAF Management: Giving Web Apps a Fighting Chance

Version 2.0
Released: August 14, 2014

Securosis, L.L.C. 515 E. Carefree Highway Suite #766 Phoenix, AZ 85085 **T** 602-412-3051 info@securosis.-com www.securosis.com

Author's Note

The content in this report was developed independently of any sponsors. It is based on material originally posted on the Securosis blog, but has been enhanced, reviewed, and professionally edited.

Special thanks to Chris Pepper for editing and content support.

Licensed by: Alert Logic

Alert Logic, the leading provider of Security-as-a-Service solutions for the cloud, provides solutions to secure the application and infrastructure stack. By integrating advanced security tools with 24×7 Security Operations Center expertise customers can defend against security threats and address compliance mandates. By leveraging an "as-a-Service" delivery model, Alert Logic solutions include day-to-day management of security infrastructure, security experts translating complex data into actionable insight, and flexible deployment options to address customer security needs in any computing environment. Built from the ground up to address the unique challenges of public and private cloud environments, Alert Logic partners with over half of the largest cloud and hosting service providers to provide Security-as-a-Service solutions for business application deployments for over 1,800 enterprises. Alert Logic is based in Houston, Texas, and was founded in 2002. For more information, please visit www.alertlogic.com.

Contributors

The following individuals contributed significantly to this report through comments on the Securosis blog and follow-on review and conversations (in no particular order):

Ahmed Masud
Jan Poczobutt
Andrew Ward
Ivan Ristic
Robert Hansen
Chris Grove
Marc Shinbrood

Copyright

Table of Contents

Introduction

Outside our research on ROI and ALE, nothing has prompted as much impassioned debate as Web Application Firewalls (WAFs). Every time someone on the Securosis team writes about Web App Firewalls we create a firestorm. The catcalls come from all sides: "WAFs suck," "WAFs are useless," and "WAFs are just a compliance checkbox product." Usually this feedback comes from penetration testers who navigate around the WAF during their evaluations, and who find their situations complicated by the presence of a WAF. The people responsible for daily WAF operations tell a different story. Both employees and third party service providers who actively manage WAFs acknowledge the difficulty of keeping WAF rules up to date, and the challenges of working closely with application developers. But at the same time, these folks understand how WAFs positively impact their overall application security approach, and are looking for more value from their investment by optimizing their WAFs to reduce application compromises and risks to their systems. We constantly engage with dozens of companies dedicated to leveraging WAFs to protect applications. The people who buy and maintain WAF installations may be less vocal than penetration testers, but they are in a better position to evaluate WAF's value to their company.

Every time we started digging in, we found a fractured market solving a limited set of customer use cases, and our conversations with security practitioners brought up strong arguments both for and against the technology.

Web Application Firewalls remain at the top of our research agenda because they are an important facet of application security. Many customers simply cannot respond to all of the vulnerabilities in their application stacks, so WAF is the only economically viable option. But every time we started digging in we found a fractured market solving a limited set of customer use cases, and our conversations with security practitioners brought up strong arguments both for and against the technology. WAFs have been available for years and are widely deployed, but their capability to detect threats varies widely, along with customer satisfaction. The resemblance to the SIEM market a few years ago is uncanny.

Rather than our typical "Understanding and Selecting" research papers, which are designed to educate customers on what a technology does and the use cases addressed, this paper focuses on how to effectively *use* a WAF.

You already know what WAFs are supposed to do, so we'll cover what you need to do in order to make WAFs effective for countering web-borne threats, and how a WAF helps mitigate application vulnerabilities. We'll dig into the reasons for the wide disparity in opinions on the usefulness of these platforms. This debate

really frames WAF management issues — sometimes disappointment with WAF results from the quality of one specific vendor's platform, but far more often the problems are due to mismanagement of the product.

Defining WAFs

Before we go any farther, let's make sure everyone is on the same page for what we are describing. We define Web Application Firewalls as follows:

A Web Application Firewall (WAF) monitors requests to, and responses from, web based applications or services. Rather than general network or system activity, a WAF focuses on application-specific communications and protocols — such as HTTP, XML, and SOAP. WAFs look for threats to applications — such as injection attacks and malicious inputs, tampering with protocol or session data, business logic attacks, and scraping information from the site. All WAFs can be configured purely to monitor activity, but most are used to block malicious requests before they reach the application; sometimes they are even used to return altered results to the requestor.

> WAF is essentially a peer of the application, augmenting its behavior and providing security when and where the application cannot.

Why Customers Buy a WAF

WAFs are selling at a brisk pace and the market continues to grow. Why? The first reason is it's a central pillar to application security, used commonly by firms with a large legacy code base. WAF protects code that is too difficult — or too fragile — to secure. The second driver can be summed up in three words: *Get. Compliant. Fast.* The Payment Card Industry's Data Security Standard (PCI-DSS) prescribes WAF as an appropriate protection for applications that process credit card data. The standard offers a couple of options: build security into your application, or protect it with a WAF. The validation requirements for WAF deployments are far less rigorous than for secure code development, so most companies opt for WAFs. Plug it in and get your stamp. WAF has simply been the fastest and most cost-effective way to satisfy the PCI-DSS standard.

> WAFs are seen as a simpler, faster, and cheaper way to bolt security on rather than patching all the old stuff.

The reasons WAFs existed in the first place, and still one of the most common reason for customers to purchase them, is that Intrusion Detection Systems (IDS) and general-purpose network firewalls are poorly suited to protecting the application layer. They remain largely ineffective in that use case. In order to detect application misuse and fraud a device must understand the dialogue between the application and the end user. WAFs were designed for this need, and they 'understand' application protocols so they can identify when an application is under attack.

Our research shows a consistent trend: more and more firms are trying to to get more value out of their WAF investments. The fundamental change is motivated by companies which need to reign in the costs of securing legacy applications under continuing budget pressure. These large enterprises have hundreds or thousands of applications, built before anyone considered 'hacking' a threat. You know, those legacy applications that don't have any business being on the Internet, but are now "business critical" and exposed to every attacker on the Intertubes. The cost to retroactively address exposures within the applications themselves is often greater than the application's value, and the time to fix them is measured in years — or even decades. Deep code-level fixes are not an option — so once again WAFs are seen as a simpler, faster, and cheaper way to bolt security on rather than patching all the old stuff.

This is why firms which originally deployed WAFs to "Get compliant fast!" are now trying to "Secure legacy apps for less!"

The Trouble with WAFs

WAFs present a quandary for enterprises. On one hand, compliance mandates have made WAFs the path of least resistance for application security. Plenty of folks have devoted a ton of effort to making WAFs work with mixed results, and they are now looking for more value, above and beyond the compliance checkbox.

On the other hand there is broad dissatisfaction with the technology, even among folks who use WAFs extensively. Before we get into an operational process for getting the most out of your WAF investment, it's important to understand why security folks often view WAFs with a jaundiced eye. Understanding opposing viewpoints between security, app developers, operations, and business managers can help to pinpoint the issues with WAFs, and provide a direction for maximizing the WAF's value. These issues must be addressed before the technology can reach the pervasive adoption level of other security technologies such as firewalls and IPS. The main arguments we have found against WAF are:

It's important to understand why security folks often view WAFs with a jaundiced eye. Understanding opposing viewpoints between security, app developers, operations, and business managers can help to pinpoint the issues with WAFs.

- **Pen-tester Abuse:** Pen testers don't like WAFs. There is no reason to beat around the bush. First, the technology makes a pen tester's job more difficult because a WAF blocks (or should block) the kind of tactics they use to attack clients via their applications. That forces them to find their way *around* the WAF, which complicates their jobs. In most cases, they evade the WAF and bust the application, so the WAF must suck, right? It's not that simple. The reality is that many WAFs are configured to neither block nor conceal the information pen testers look for and leverage for evasion. Information about the site, details about the application, configuration data, and even details on *the WAF itself* leak out, making the WAF nothing more than a speed bump for a good pen tester. Far too many WAF deployments are more about getting that compliance checkbox rather than stopping hackers or pen testers, so the pen testers vilify the technology — rather than pointing at the implementation. Do you see the conflict of interest here?

- **WAFs Break Apps:** The security policies — essentially the rules that determine what a WAF blocks and what passes through to applications — can and do block legitimate traffic at times. Web application developers are used to *turning code* — often pushing changes and new functionality to web applications several times per week, if not more often. Unless someone updates the 'whitelist' of approved application requests for every application change, the WAF will break the app — blocking legitimate requests, pissing off customers and operational folks alike. Developers get blamed, they point at security, and nobody is happy.

- **Compliance, Not Security:** A favorite catchphrase among security professionals is, "You can be compliant and still not be secure." At least the ones who know what they're talking about. Regulatory and industry compliance initiatives are designed to "raise a very low bar" on security controls, but compliance mandates inevitably leave loopholes — particularly in light of how often they can realistically be updated. Loopholes attackers can (and do) exploit. Even worse, the goal of many security programs becomes a quick fix to pass compliance audits — not to actually protect critical corporate data. The perception of WAF as a quick fix for achieving PCI-DSS compliance — often at the expense of security — leaves many security personnel with a bad taste. WAF is not a 'set-and-forget' technology, but for compliance it is often used that way — providing mediocre protection. Until WAF proves its usefulness in blocking real threats or slowing down attackers, many remain unconvinced of WAF's overall value.

- **Skills Gaps:** Application security is a non-trivial endeavor. Understanding spoofing, fraud, non-repudiation, denial of service attacks, and application misuse are skills rarely all found in any one individual. But they are all needed to be an effective WAF administrator. We once heard of a WAF admin who ran the WAF in learning mode **while a pen test was underway** — so the WAF learned the bad behavior as 'normal' and configured policies to allow that traffic! Epic fail. Far too many folks get dumped into the deep waters of trying to make a WAF work without a fundamental understanding of the application stack, business process, or security controls. The end result is mediocre protection — perhaps not accounting for current security threats, not adapting to changes in the environment, or not reflecting the current state of the application. All too often, the platform lacks adequate granularity to explicitly detect all variants of a particular threat, or essential details are not coded into policies, leaving further openings to be exploited. But is this an indictment of the technology, or how it is utilized?

We once heard of a WAF admin who ran the WAF in learning mode **while a pen test was underway** — so the WAF learned the bad behavior and configured policies to allow that traffic!

- **Perception and Reality:** Like all security products, WAFs have undergone steady evolution, and are now a mature product category. But the perception of WAFs still suffers because early WAFs were themselves subject to many of the attacks they supposedly defended against (WAF management is through a web application, after all). Early generations of WAFs also had high false positive rates and ham-fisted threat detection at best. Some WAFs bogged down under the weight of additional policies, and no one ever

wanted to remove old policies for fear of allowing compromise. We know there were serious growing pains with WAFs, but most of the current products are mature, full-featured, and reliable — despite the persistent perception otherwise.

- **Integration:** Another disappointing element, and the most applicable to development, has been the lack of programmatic interfaces to manage WAF policies. Sure, you can *output* alerts to `syslog`, but otherwise integration of command and control with other management tools is non-existent. Remote policy management through an API? Nope. You use the default user interface or you don't use the product. It would be beneficial for WAFs to include programmatic interfaces to link into application development cycles. In this way as the application changes were pushed out, you could both ask the WAF to relearn the altered pages, and instruct the WAF to monitor, alert or block *in advance* of the code changes. We hope APIs will show up on roadmaps — they might even allow the development and QA teams to participate in the management of application security.

- **User Interfaces:** The WAFs we have seen in action provide a full feature set, with plenty of deployment variations to satisfy the needs of discriminating customers. They also offer miserable user experiences. The amount of manual effort needed to run your average WAF is significant, and for a product category around for 8-10 years, disappointing. We understand that someone who manages a WAF must possess good knowledge of how attacks work, HTML and XML protocols, and how applications are deployed, but that does not mean users want to bathe in esoteric details or work through long lists of checkboxes to set up each policy. Signature templates, anyone? Bueller? Many automated functions require lots of manual setup before they are put to good use — and even some of the learning tools don't work until a human has reviewed, tweaked many dials, and clicked lots of boxes. Current user interfaces work for building a handful of policies for one or two web sites, but most don't scale well to large deployments. Worse, UI complexity invites mistakes — compounding issues. They're not all bad, but WAF user interfaces generally do not make anyone's job easier.

> The most serious problems with WAF are not about technology, but with management of the platform.

Our research shows that WAF failures are *far* more often a result of operational failure than of fundamental product failure. Make no mistake — **WAFs are not a silver bullet** — but a correctly deployed WAF makes it *much* harder to attack an app or to completely avoid detection. The effectiveness of WAFs is directly related to the quality of people and processes used to keep them current. *The most serious problems with WAF are not about technology, but with management.*

So we need a pragmatic process to manage Web Application Firewalls to overcome the management and perception issues that plague this effective application security technology.

The WAF Management Process

There are many reasons WAFs frustrate security and application developers. But thanks to the 'gift' of PCI, many organizations have a WAF in-house, and now want to use it more effectively. Which is a good thing, by the way.

There are no silver bullets. Not profiling apps. Not integration with vulnerability reporting and intelligence services. Not anything.

We have mapped out a clear and pragmatic three-phase approach to WAF management. First the caveats. There are no silver bullets. Not profiling apps. Not integration with vulnerability reporting and intelligence services. Not anything. **Effectively managing your WAF requires an significant ongoing commitment.** In every aspect of the process you will see the need to revisit everything, over and over again. We live in a dynamic world — which means a static ruleset won't cut it. The sooner you accept that, the sooner you can achieve a singularity with your WAF. We will stop preaching now.

Manage Policies

At a high level you need to think of the WAF policy/rule base as a living, breathing entity. Applications evolve and change — typically on a daily basis — so WAF rules also need to evolve and change in lockstep. But before you can evolve your rule base you need to build it. We have identified 3 steps for doing that:

1. **Baseline Application Traffic:** The first step in deploying a WAF is usually to let it observe application traffic during a training period so it can develop a reference baseline of 'normal' application behavior for all the applications on your network. This initial discovery process and associated baseline provides the basis for the initial ruleset, basically a *whitelist* of acceptable actions for each application.

2. **Understand the Application:** The baseline represents the first draft of your rules. Then you apply a large dose of common sense to see which rules don't make sense and what's missing. You can do this by building threat models for dangerous edge cases and other situations to ensure nothing is missed.

3. **Protect Against Attacks:** Finally you will want to address typical attack patterns by deploying *blacklists* to watch for common attack patterns. This is similar to how an Intrusion Prevention System works at the network layer. This will block common but dangerous attacks such as SQLi and XSS.

Now you have your initial rule set, but it's not time for Tetris (or Candy Craft or Plants vs. Zombies) yet. This is only the beginning. We will go into detail on the issues and tradeoffs of policy management later in this paper — for now we just want to capture the high-level approach. You need to revisit the ruleset constantly— both

to deal with new attacks (based on what you get from your vendor's research team and public vulnerability reporting organizations such as CERT), and to handle application changes. Which brings us to the next step.

Application Lifecycle Integration

This next step in the WAF management process involves collaboration between the proverbial irresistible force and immovable object — developers and security — to jointly protect applications. Communication between groups is the starting point — later you will want to provide filtered, prioritized, and digestible information to DevOps, but start simple. Further complicating matters are evolving development processes, new development tools, and application deployment practices — all of which WAFs need to integrate with. Obviously you will work with developers to identify and eliminate security defects as early in the process as possible. But the security team needs to be realistic — adversely impacting a developer's work can have a dramatic negative impact on the quality and amount of code that gets shipped and future collaboration. And nobody wants that.

This next step in the WAF management process involves collaboration between the proverbial irresistible force and immovable object to protect applications.

We have identified a set of critical success factors for integrating with the DLC (development lifecycle):

1. **Executive Sponsorship:** If a developer can say 'no' to the security team, at some point under duress they will. Either security is important or it isn't. To move past a WAF useful only for compliance, security folks need the CIO or CEO to agree that the velocity of feature evolution must give way to addressing critical security flaws. Once management has made that commitment developers can justify improving security as part of their job.

2. **Establish Expectations:** Agree on what defines a critical issue, and how critical issues will be addressed among the pile of competing critical requirements. Set guidelines in advance so there are no arguments when issues arise.

3. **Security/Developer Integration Points:** There must be logical (and documented) steps within the application lifecycle for security checks, assessments, etc. This includes design and development integration points, as well as pre-launch validation with operations. Without formal security involvement inevitably these steps will be skipped — especially at crunch time. Go back to point 1 above.

4. **Automation:** Even with executive sponsorship and clearly documented integration points, sufficient automation of security functions (scanning, testing, regression, etc.) is required to scale the operational aspects of application security. Are you seeing a pattern here? The person managing the WAF needs to filter out the noise before it gets to the development team, and provide high-priority actionable information.

5. **Feedback Loops:** Finally, you need to be able to adapt both the application and the process based on feedback at all points in the collaboration. Nothing turns off either side faster than feeling unappreciated. Sure, that's straight out of a new-age success manual, but it's still true.

If you find yourself lacking in any of those aspects, your likelihood of keeping your WAF current and effective is remote.

Securing the WAF

Finally you actually need to protect the WAF itself. We are talking about both setting up the device properly and configuring it to thwart evasion techniques and avoid information leakage. Keep these security considerations in mind:

1. **Securing the Device:** Unplug the device and bad things happen, right? So controlling physical access is a must — as with any computing device. Increasingly you will also need to protect the WAF against a variety of denial of service (DoS) tactics that can render it useless.

2. **Provisioning/Managing Entitlements:** Once the device is secure you need to ensure that only authorized users can mess around with the ruleset and manage the device. You could look at technology like Privileged User Monitoring, or just aggressively manage entitlements to the devices, but you need to ensure a savvy attacker cannot slip in an 'any-to-any' rule.

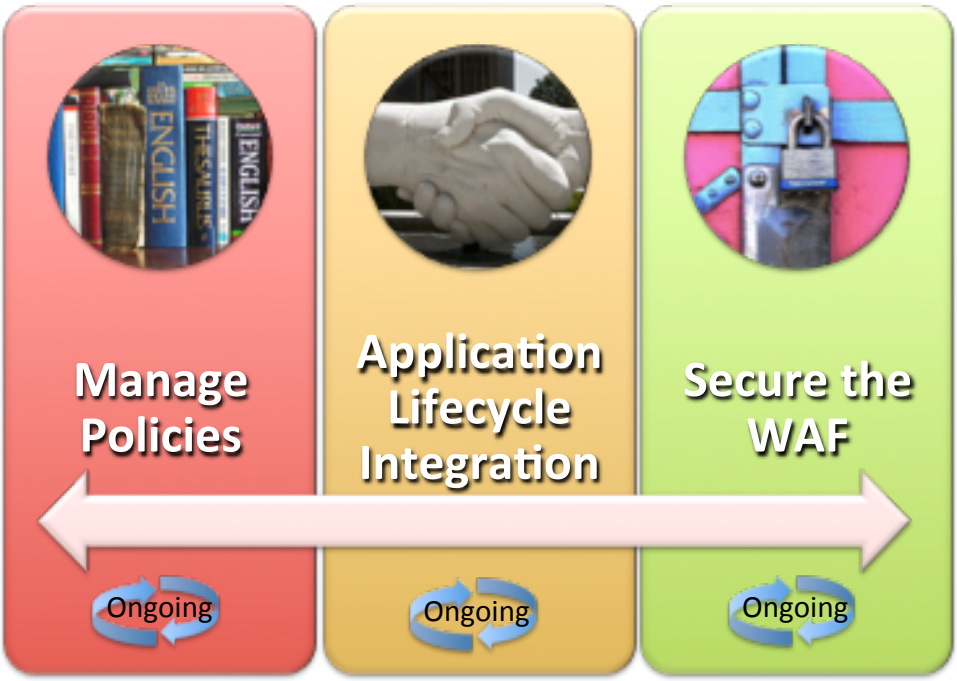
3. **Preventing Evasion:** You need to pay attention to network architecture to ensure an attacker can't just bypass the WAF to get a direct pipe into the application. WAF evasion is an emerging area of interest for security researchers, so you need to stay abreast of new tactics to ensure your WAF has an opportunity to do its job.

4. **Rule Obscurity:** Yes, security folks grimace when they hear the term 'obscurity', but they know sophisticated attackers poke and prod applications before trying to break in. They will use all sorts of tactics to learn as much about your WAF as they can, so they can figure out how to evade it.

This isn't an exhaustive list but it's a start with a few of the main considerations for securing a WAF. But we haven't touched on the importance of WAF deployment architectures to all of the above. For example a managed WAF service should take DoS attacks out of play. Services do not actually *eliminate* the risk — but handling it becomes the service provider's problem. Though understand there may be some financial penalties for traffic overages or attacks endangering other tenants in the shared datacenter. Performance and provisioning also depend on whether the device is on-premise or hosted by a service provider.

Another key to securing the WAF is ongoing testing to ensure that nothing changes unexpectedly. Just as the applications you protect are dynamic, so is the product used to protect it. This process must include an

ongoing commitment to penetration testing both the application and the WAF.



As you can see from the illustration, each step in the process repeats on an ongoing basis. How better to build a process to protect applications than with nested loops and recursion? Okay, we're kidding, but the end result is the same — every aspect of managing WAFs is an ongoing process. This is the antithesis of *set it and forget it* technology. That is the real point of this research. To maximize value from your WAF you need to go in with everyone's eyes open to the effort required to get and keep the WAF running productively.

Policy Management

To get value out of your WAF investment — blocking threats, keeping unwanted requests and malware from hitting applications, and virtually patching known vulnerabilities in the application stack — the WAF must be tuned regularly.

To flesh out the process of keeping a WAF up to date let's dig into policy management — specifically how to tune policies to defend your site. But first it's worth discussing the different types of polices at your disposal. Policies fall into two categories, *blacklists* of stuff you **don't** want — attacks you know about — and *whitelists* of activities that are permissible for specific applications. These negative and positive security models complement each other to fully protect applications.

Negative Security

Negative security models should be familiar — they are the concept behind Intrusion Prevention Systems. The model works by detecting patterns of *known* malicious behavior. Things like site scraping, injection attacks, XML attacks, suspected botnets, Tor nodes, and even blog spam, are universal application attacks that affect all sites. Most of these negative policies come "out of the box" from vendors, who research and develop *signatures* for their customers. Each signature explicitly describes an attack, and they are typically used to identify attacks such as SQL injection and buffer overflows. The downside of this method is its fragility — any variation of the attack will no longer match the signature, and will thus bypass the WAF. So signatures are only suitable when you can reliably and deterministically describe an attack, and don't expect the signature to immediately be bypassed by simple evasion.

> 'Negative Security' policies block known attacks.

WAF devices also provide a myriad of other detection options, such as heuristics, reputation scoring, recognition of evasion techniques, and several proprietary methods used to qualitatively detect attacks. Each method has its own strengths and weaknesses, and use cases for which each class of rule is more or less well suited. They can be combined with each other to provide a risk score for incoming requests, in order to block requests that look too suspicious. But the devil is in the details, there are thousands of attack variations, and figuring out how to apply policies to detect and stop attacks is quite difficult.

Finally fraud detection, business logic attack detection, and data leakage policies need to be adapted to the specific use models of your web applications to be effective. The attacks are designed to find flaws in the way application developers code, targeting gaps in the ways they enforce process and transaction state. Examples include issuing order and cancellation requests in rapid succession to confuse the web server or

database into revealing or altering shopping cart information, replaying attacks, and changing the order of events. You generally need to develop your own fraud detection policies, as the types of fraud tend to be specific to your business. They are constructed from the same analytic techniques, but rather than focusing on the structure and use of HTTP and XML grammars, a fraud detection policy examines user behavior as it relates to the type of transaction being performed. These policies require an understanding of how your web application works, as well as appropriate detection techniques.

Positive Security

The other side of this coin is the positive security model: 'whitelisting'. Catalog legitimate application traffic, ensure you do not include any attacks in your 'clean' baseline, and set up policies to block anything not on the list of valid behaviors.

The good news is that this approach is very effective at catching malicious requests you have never seen before (0-day attacks) without having to explicitly code signatures for everything. This is also an excellent way to pare down the universe of all threats into a smaller, more manageable subset of specific threats to account for with a blacklist — basically ways authorized actions such as `GET` and `POST` can be gamed.

> 'Positive Security' only allows known web requests, and blocks all others.

The bad news is that applications are dynamic and change regularly, so unless you update your whitelist with each application update the WAF will effectively disable new application features. Regardless, you will use both approaches in tandem — with only one approach the job is harder and security suffers.

People Manage Policies

There is another requirement which must be addressed before adjusting polices: assigning someone to manage them. In-house construction of new WAF signatures, especially at small and medium businesses, is uncommon. Most organizations depend on the WAF vendor to do the research and update policies accordingly. It's a bit like anti-virus: companies could theoretically write their own AV signatures, but they don't. They don't monitor CERT advisories or other source for issues to protect applications against. They rarely have the in-house expertise to write these dynamic attack-based policies even if they wanted to. But if you want your WAF to perform better than AV, you need to adjust your policies to your environment.

> Having someone skilled enough to write and manage WAF policies is a prerequisite for success.

So you need someone who can understand the rule 'grammars' and how web protocols work. That person must also understand what type of information should not leave the company, what constitutes bad behavior, and the risks your web applications pose to the business. *Having someone skilled enough to write and manage WAF policies is a prerequisite for success*. It could be an employee or a third party, or you might even pay the vendor to assist, but you *need* a skilled resource to

manage WAF policies on an ongoing basis. There really is no shortcut — either you have someone knowledgable and dedicated to this task, you depend on the canned policies that come with the WAF, or you let a managed WAF service provider deal with it. The critical success factor for managing policies is to find at least one person who can manage the WAF, get them training if necessary, and give them time to keep the policies up to date.

What does this person need to do? Let's break it down:

Baseline Application Traffic

The first step is to learn how the applications are deployed and what they are doing. Sure, some of you just install the WAF, update the attack signatures, and let it rip. Very little effort but even less security. And we hear plenty of stories about frustrated users who botched their first deployment, but that's water under the bridge. Taking a fresh look involves letting the WAF observe your application traffic for a while. All WAF platforms can be deployed in *monitor only* mode to collect traffic and alert on policy violations but most also provide a 'learning' mode to assist in classifying application usage, users, structure and services. These baselines are your roadmap to what applications actually do — providing information you need regarding *what* you need to secure and clues to the best way to write security polices. This initial discovery process is very important for ensuring your initial ruleset actually covers what's really out there, not just what you *think* is out there. There is really no other effective way to understand what 'normal' application behavior is for all the applications on your network, or to get a handle on the different applications and services you need to protect.

Understand the Applications

Now that you have cataloged the application traffic, you want to understand the transactions being processed and get an idea of the risks and threats you face in the context of that business application. Mapping the baselines into a set of rules requires understanding application activities and transactions in order to select the appropriate type of rule. Half the battle is understanding the application, and the other half is understanding how attackers will attempt to undermine its functions. Understanding of specific application transactions is not something you will have coming into this process, unless you already possess deep knowledge of the application, so you will need to work closely with DevOps to understand how each application works and how events are processed. Part of your job will be to work with the people who possess the information you need in order to understand each application well enough to document key functions subject to attack.

Once you understand the application functions, and have a pretty good idea of what attacks to expect, you need to determine how you will counter them. For example, if your application has a known defect that cannot be addressed in a timely fashion through code changes, you have several options:

- You can remove the offending function from the whitelist of allowed activity, thereby removing the threat. The downside is that this also removes an app function or service, effectively breaking at least part of the application.

- You can write a specific signature that defines the attack so you can detect attempts to exploit the vulnerability. This approach will stop known attacks but requires you to account for all possible variations and evasion techniques.
- You can use one or more heuristics to look for clues that indicate abnormal application use or an attack. Heuristics include malformed requests, odd customer geolocations, customer IP reputation, use of known weak or insecure application areas, requests for sensitive data, and any of a number of other attack indicators.

Many customers remain apprehensive about this part of the process. They have a handle on application functionality so they can create whitelists for their applications. But they lack knowledge of how attackers work, so they are not clear on what threats to look for or how to write specific policies. Fear not — there are several ways to fill this knowledge gap.

The first thing to understand is that the WAF vendor will be most helpful in understanding hackers. Review the standard policies that come with the WAF. This will give you a very good idea of the types of attacks in current use — spoofing, repudiation, tampering, etc. Embedded policies reflect the long organizational memories of WAF vendors for web app attacks, and how they choose to block them. They should help you understand how attackers work and provide clues for writing your own policies. Second, there are publicly available lists of attacks — from OWASP, CERT and other organizations — which you can familiarize yourself with. Some even include example attacks, WAF evasion examples, and rules for detecting them. Third, you can hire penetration testers to probe your applications and identify weaknesses. This type of examination is excellent because it approaches exploitation and WAF evasion the way an attacker would. Finally, if you still have concerns, you can always buy professional services to get over the hump.

Another daunting part of this exercise is the scope of the problem. It's one thing if you're only talking about a handful of applications — but it's entirely different when you consider 200 applications, or 1,000. The rules you implement for each application differ, so this process is iterative. When facing an overwhelming number of applications, it's best to take a divide-and-conquer approach: implement basic security across all applications and select a handful of applications to get advanced rules. Iterate your way through the rest of the applications and services, starting with the apps that pose the greatest risk to your organization. Work closely with development teams to cut this problem down to size.

If your WAF has a *learning mode*, use it. Baselining each application and building a behavior profile will help you plow through policies much faster.

If your WAF has a *learning mode* use it. Baselining each application and building a behavior profile will help you plow through policies much faster. You will still need to apply skull sweat to see which rules don't make sense and what's missing. You can do this by building threat models for dangerous edge cases and other situations to ensure nothing is missed. But learning tools do speed up the process.

Protect against Attacks

Now that you have decided how to block threats to your application, it's time to implement the rules and apply them to the WAFs. The WAF administrators need to both write rules and develop test cases to verify the effectiveness of deployed rules. You will want to deploy both negative and positive rule sets, one at a time, then verify that the rules have propagated to each WAF and are protecting applications. Don't *assume* that rules are active or even working — verify with the test cases you developed. Testing is your friend. We previously mentioned the company that taught their WAF to accept all attacks as legitimate traffic — running a pen test while the WAF was in learning mode. The key failure was not messing up the WAF rules — it was not testing the resulting ruleset or discovering the problem until a third-party pen tester probed their system months later. The moral is: Verify that your rules are in place and effective.

> Don't *assume* that rules are active — verify with the test cases you developed. *Testing is your friend.*

Remember this is an ongoing process — especially in light of the dynamic attack space your blacklist addresses, false-positive alerts requiring rule set tuning, and the application changes driving your whitelist. Your WAF management process needs to continually learn and catalog user and application behaviors, and collect metrics as part of the process. Which metrics are meaningful, and which activities you need to monitor most closely, differ between customers and vendors and solutions. The only consistency is that you cannot measure success without progress and performance metrics.

A final note: writing negative security policies requires a balancing act. You need to ensure that the pattern reflects a specific threat, but its definition must not be so specific that it misses trivial variants. A signature is useless if it is immediately rendered obsolete by evolution of the attack. Finding that balance is the hardest aspect of configuring your policies. You need to detect threats without generating too many false positives or negatives. Of course that is easier said than done, so you will continually revisit policies to provide effective protection for your applications.

Application Lifecycle Integration

Let's be honest — developers don't like security folks, and vice-versa. That's an overgeneralization, but it's largely true. Worse, developers don't like security tools that barrage them with huge amounts of stuff they're supposed to fix — especially when the barrage includes many noisy inconsequential issues and/or totally bogus results. The security guy wielding a WAF is an outsider, and his reports are full of indigestible data, so they are likely to get stored in the circular file. It's not that developers don't believe there are issues — they know there is tons of stuff that ought to be fixed, mostly because they took shortcuts to deliver code on deadline. And they know the backlog of *functional* stuff they would like to fix — over and above the threats reported by the WAF, dynamic app scans, and pen testers — is simply too large to deal with. Web-borne threat? Take a number.

Security folks wonder why developers can't build secure code, and developers feel security folks have no appreciation of their process or the pressure to ship working code. We said *working code* — not necessarily **secure** code — which is a big part of the problem. You can't forget about Operations either — you know, the folks responsible for making sure the systems run smoothly and reliably. Yet another system to manage on their network may impact performance, failover, ease of management and user experience. None of which are selling points to an Operations staffer.

The purpose of a WAF is to protect web facing applications from attacks. We can debate build-security-in versus bolt-security-on *ad infinitum*, but ultimately the answer is *both*. Better still, the data a WAF collects on application usage patterns, errors and more subtle issues will help the development team build better code. We have discussed how to build and maintain WAF policies to protect applications, but you also need to adapt your development process to incorporate knowledge of typical attack tactics into code development practices to address application vulnerabilities over time. This involves a two-way discussion between WAF administrators and developers. Developers do their part helping security folks understand applications, what input values should look like, and what changes are expected in upcoming releases. This ensures that WAF rules remain in sync with applications.

Granted, not every WAF user will want to integrate their application development and WAF management processes. But separation both limits the effectiveness of the WAF and puts the application at risk. At a minimum, developers (or the DevOps group) should have ongoing communications with WAF managers to avoid having the WAF complicate application deployment and keep the WAF from interfering with normal application functions. This collaboration is critical, so let's dig into how it should work.

Web applications change constantly, especially given increasingly *'agile'* development teams — some pushing web application changes multiple times per week. Many web application development teams don't even attempt to follow formal release cycles — effectively running an "eternal beta cycle." The development team's focus and incentives are on introducing new features as quickly as possible to increase customer engagement. But this doesn't help secure applications, and seriously complicates efforts to keep WAF policies current and effective.

The greater the rate of application change, the harder it is to maintain WAF policies.

The greater the rate of application change, the harder it is to maintain WAF policies. This simple relationship seriously complicates a major WAF selling point: its ability to implement positive security policies based on acceptable application behavior. Whitelist policies enumerate acceptable commands and their associated parameters. A WAF learns about the web applications it protects by monitoring user activity and/or crawling application pages, determining which need protection, which serve static and/or dynamic content, the data types and value ranges for page variables, and other aspects of user sessions. If the application undergoes constant change the WAF will always be behind, introducing a risky gap between *learning* and *protecting*. New application behavior isn't reflected in WAF policies, which means legitimate requests will blocked and illegal requests will be ignored. That makes the WAF much less useful.

To mitigate these issues we have identified a set of critical success factors for integrating with the SDLC (software development lifecycle). When we outlined this process previously, we mentioned the friction between developers and security teams and how this adversely effects their working relationship. Our goal is to help set you on the right path and prevent the various groups from feuding like the Hatfields and McCoys — trust us, it happens all too often. Here's what we recommend:

1. **Executive Sponsorship:** If, as an organization, you can't get developers and operations in the same room with security, WAF administrators are stuck on a deserted island. It's up to them to figure out what each application is supposed to do and how it should be protected, and they cannot keep up without sufficient insight or visibility. Remember developers are paid to ship new features, not to provide security. Security folks need someone up the food chain — the CISO, the CIO, or even the CEO — to agree that the velocity of feature evolution must give some ground to operational security. Once management has

made that commitment, developers can justify improving security as part of their job. It is also possible — and in some organizational cultures advisable — to include security in application specifications. This helps guarantee code does not ship until it meets minimum security requirements — either in the app or in the WAF.

> Security folks need someone up the food chain — the CISO, the CIO, or even the CEO — to agree that the velocity of feature evolution must give some ground to address operational security.

2. **Establish Expectations:** All parties need to learn what's required and expected to get their jobs done with minimum fuss and maximum security. We suggest you arrange a sit-down with all stakeholders (Operations, Development, and Security) to establish some guidelines on what *really* needs to happen, and what would be nice to have. Most developers want to know about broken links and critical bugs, but they get surly when you send them thousands of changes via email, and downright pissed when all requests relate to the same non-critical issue. It's essential to get agreement on what constitutes a critical issue and how critical issues will be addressed among the pile of competing critical requirements. Set guidelines in advance so there are no arguments when issues arise. Similarly, security people hate it when a new application enters production on a site they didn't know existed, or significant changes to the network or application infrastructure break the WAF configuration. Ideally each party takes some work off the other's plate, so point out how these discussions as mutually beneficial. A true win-win — or at least a reduction in aggravation and wasted time.

3. **Security/Developer Integration Points:** Integration points define how the parties share data and solve problems together. Establish rules of engagement for how DevOps works with the WAF team, when they meet, and what automated tools will be used to facilitate communication. You might choose to invite security to development scrums, or a member of the development team could attend security meetings. You need to agree on a communication medium that's easy to use, establish a method for getting urgent requests addressed, and define a mechanism for escalation when they are not. Logical and documented notification processes need to be integrated in the application *development* lifecycle to ensure the security team understands application changes so they can adjust polices in advance. There must be a logical and documented process for application *deployment* that signals when to update WAF policies and propagate revisions across WAF systems. In those rare cases where a pre-production test environment uses a WAF to help test deployment, take this opportunity to have the WAF *learn* about application changes. Don't forget the beginning and end of the process either — including design, development, and pre-launch validation. Without formal security involvement, inevitably these steps get skipped — especially at crunch time. Go back to point 1 above and start over again if that happens.

4. **Automation:** This is where you implement and automate the results from steps 2 and 3. Once the initial work is done these processes need not be painful, and don't even need to be face to face. After the initial meetings most of the cooperative effort can be via less formal mechanisms — email, bug tracking

software, trouble ticket systems, or even source code management. Even with executive sponsorship and clearly documented integration points, without sufficient automation of security functions (scanning, testing, regression, etc.), things will be difficult to operationalize. Look to make things easy where possible. The WAF team needs to filter out noise before it hits the development team, and keep the focus on actionable information. In return development teams need to provide enough information so the WAF manager knows what's going on and can get the job done. And automation means getting the WAF to do the security work on your behalf — and getting your WAF deployed inline to help stop attacks once all sides have gained confidence in the technology.

5. **Feedback Loops:** Once the process is established, look for ways to improve it over time. We have seen, after the first few development cycles have passed, the value of discussing ways to make the process easier and more efficient for all parties. Don't be bashful about sharing what you learned. For example, if the WAFs blocked attacks that could be easily mitigated with a simple change to the application, it's in everyone's best interest to share that data. Conversely, dealing with false positives is easier if the WAF manager understands how the code is supposed to function. Our research also highlighted the value of this collaboration as the application development team matures their code, which in turn helps the WAF perform better. Nothing turns off either side faster than feeling unappreciated, and this is the time to show your value and how the process can make things easier.

Our focus here is on the relationship between app developers and WAF administrators. But many people who end up managing WAFs have some development background because it's difficult to write WAF policies without understanding the intricacies of web communication protocols and how web services work. And some of the folks we spoke with had no trouble integrating into the DevOps team, partially because their development background provided credibility.

Securing the WAF

Like any application, a WAF itself provides additional attack surface for adversaries. Their goal isn't necessarily to compromise the WAF itself (though that's sometimes a bonus) — the short-term need is evasion. If attackers can figure out how to get *around* your WAF, they can go directly after vulnerable applications. If they can confuse the WAF, crash it, or find areas it not configured to protect, they establish unfiltered dialog with the web application. Your WAF needs to be configured and secured, just like any other device sitting *out there* and accessible to attackers. So let's start by discussing device security, including deployment and provisioning entitlements. Then we can get into some evasion tactics you are likely to see, before wrapping up with a discussion of the importance of testing your WAFs on an ongoing basis.

Their goal isn't necessarily to compromise the WAF itself (though that's sometimes a bonus) — the short-term need is evasion. If attackers can figure out how to get *around* your WAF, they can go directly after vulnerable applications.

Deployment Options

Managing your WAF pragmatically starts when you plug in the device, which requires you to first figure out how you are going to deploy it. You can deploy WAFs either inline or out-of-band. Inline entails installing the WAF in front of a web app to block attacks directly. Alternatively, as with Network Access Control (NAC) devices, some vendors provide an out-of-band option to assess application traffic via a network tap or spanning port, and then use indirect methods (TCP resets, network device integration, etc.) to shut down attack sessions. Obviously there are both advantages and disadvantages to having a WAF inline, and we certainly don't judge folks who opt for out-of-band deployment rather than risking impact to applications. But as with NAC evasion, out-of-band enforcement can be evaded via tactics like command injection, SQL injection, and stored cross-site scripting (XSS) attacks that don't require responses from the application. These tactics can allow these attacks through to the application in an out-of-band WAF implementation, to put applications at risk. But balancing risks, such as reduced application protection against possible application disruption, is why you get the big bucks, right?

You will also need to consider high availability (HA) deployment architectures. If a WAF device fails and takes your applications down with it, that's a bad day all around. So make sure you can deploy multiple boxes with consistent policies and utilize some kind of non-disruptive failover option (active/active, active/passive, load balancer front-end, etc.).

Of course some folks opt for a managed WAF service, so the device doesn't even sit in their data center. This offloads responsibility for scaling up the implementation, providing high availability, and managing devices (patching, etc.) to the service provider. Additionally, the service provider can offer some obfuscation of your IP addresses, complicating attacker reconnaissance and making WAF evasion harder. Depending on how the service is packaged, the provider may also provide resources to manage policies. Of course they cannot offload *accountability* for protecting applications, and a service provider cannot be expected to interface directly with your developers. You should also understand your WAF provider's background. Are they a security company? Does the WAF provide full application security features, or is it a glorified content distribution network (CDN)? Obviously a service provider isn't likely to offer the full granular capabilities and policy options of a device in your data center, so you need to balance the security capabilities of a managed WAF service against what you can do yourself.

Other Security Considerations

Obviously you need to keep attackers away from your physical devices, so ensuring their physical security is the first step, and hopefully already largely covered by existing security measures. After that you need to ensure all credentials stored on the device are protected, including the SSL private keys used for SSL interception.

You also need good security hygiene on the device, which means detailed logging of any changes to the device configuration and/or policies.

You also need good security hygiene on the device, which means detailed logging of any changes to the device configuration and/or policies. Hopefully logs will be aggregated to an external system (a log management server) to prevent tampering, and alerts should be sent if logging is turned off or otherwise blocked. That also means keeping the underlying operating system (for software-based WAFs) and the WAF itself patched and up to date. No different than what you should for every other security device in your environment.

Again, a managed WAF service gets you out of having to update devices and/or WAF software, but make sure you can get access to the appropriate WAF activity logs. Make sure you have sufficient access for forensic investigation, if and when you need to go there.

Finally, keep in mind that Denial of Service (DoS) attacks continue to be problematic, targeting applications with layer 7 attacks, in addition to simpler volume-based attacks. Make sure you have sufficient bandwidth to deal with any kind of DoS attack, a sufficiently hearty WAF implementation to deal with the flood, and a DDoS-focused service provider on call to handle additional traffic if necessary. Protecting against DoS attacks is a discipline unto itself, and we plan a series on it in the near future.

Provisioning and Managing Entitlements

Once you have secured the device make sure the policies and device configurations are protected. Take steps to control provisioning and management of entitlements. Given the sensitivity of the WAF, it makes sense to get back to the 3 A's. Yeah, man, old school.

- **Authorization:** Who is allowed to access the WAF? Can they set up policies? Change configurations?
- **Authentication:** Once you know who can legitimately get into the device, how will you ensure it's really them? Passwords? 2-factor authentication? Digital certificates? Retinal scans? Okay, that last one was a joke, but the question isn't.
- **Audit:** You want an audit event every time a WAF policy, configuration, entitlement, or anything else is changed.

Note that a managed WAF service will complicate your ability to manage entitlements. The service provider will have the ability to change policies and may even be responsible for managing them. Ensure you adequately vet folks who will have access to your policies, with an audit trail. We know we are beating the audit horse, but it's particularly important in this context.

An alternative method for managing access to WAF devices is Privileged User Management (PUM). In this scenario administrators log into some sort of proxy, which manages credentials and provides access *only* to the WAFs each administrator has authorization for. That's just one of the benefits of PUM, so check out our research paper to learn more about these products.

Preventing Evasion

Attacking an application is a means to an end. The attacker is focused on getting the data, and (for now) the path of least resistance is through an application. The WAF is positioned to prevent that attack, so the attacker has a choice. Go *through* the WAF or *around* it. In many cases it's easier to evade a WAF than compromise it. This is another similarity between WAF and Intrusion Prevention Systems (IPS). Both use negative security policies to detect and block attacks. Obviously the WAF is optimized for different attacks against different targets, but they are conceptually very similar.

The WAF is positioned to prevent that attack, so the attacker has a choice. Go *through* the WAF or *around* it. In many cases it's easier to evade a WAF than compromise it.

Secure device evasion isn't a new — attackers (including penetration testers) have been building tools to evade IPS devices for years, and many of them work just as well to evade a WAF. Rather than futzing with lower-level network protocols to evade an IPS, an attacker attempting to circumvent a WAF can instead mess with web requests — HTTP parameter manipulation, adding SQL comments in parameters, changing case, encoding URLs and URIs, gaming SSL interception, inserting random words, packet fragmentation, etc. Any of these tactics might mask an attack pattern the WAF is watching for.

The other main tactic used in WAF evasion is identification of semantic gaps. That basically means understanding that an application and/or database will interpret commands and scripts differently than a WAF, and that provides opportunity for an attacker to game the system. Understanding the details of semantic gaps is a bit deeper than we can go in this research paper.

Reconnaissance

Attackers and security folks play an ongoing cat and mouse game. The attacker starts by poking around, trying to find weaknesses in an application. The good news is that you know how the reconnaissance process works, which enables you to make it harder.

The attacker's first step is to profile your application. They will try to figure out specific IP addresses, web services, operating systems, session timeouts, cookie settings, application response codes, etc. Anything that can help identify specific technologies may suggest areas of weakness or provide clues on how to attack a system. They will also try to identify the particular WAF in place — bad guys share research on which tactics evade work against which WAF devices. So they send a bunch of traffic at the application, then scan the responses they receive for clues to the WAF in use. It's important to review what fingerprints your WAF leaves behind and erase those clues where possible.

They also refine their profiling efforts to get a feel for which rules are active. A skilled attacker will get a pretty good feel for both your application and your defenses at the end of their reconnaissance efforts. And they will have a pretty good idea of how to beat most systems at the end of the reconnaissance phase — even before they have started trying to get past your defenses. Fortunately this is a resource-heavy approach, but if their mission involves getting your data, they will put the time in. You can also learn a bit more detail about reconnaissance for WAF evasion (and an approach to protection) at PaulDotCom.

Hiding the IP addresses of your applications using a CDN or caching service — especially one that employs a coarse WAF filter — can be very helpful by making it harder for attackers to know what they are up against.

Making Evasion Harder

Fortunately there are tactics to make this much harder for attackers, starting with obfuscation. Security by obscurity doesn't generally work very well, but in this case hiding the IP addresses of your applications using a CDN or caching service — especially one that employs a coarse WAF filter — can be very helpful by making it harder for attackers to know what they are up against. This is a feature of managed WAF services as well.

You can also rely on having the WAF just drop malformed requests, rather than sending response codes. At times silence is your best defense, and not giving attackers any clues as to what you are doing can be helpful. A more aggressive disinformation tactic involves "active defense" or "intrusion deception," which involves sending confusing or just wrong response codes to confuse attackers. Of course you need to be sure you are successfully identifying reconnaissance or attack traffic

before you employ these techniques, because it would be unacceptable to mis-identify and deceive or counterattack legitimate users.

The point is not to enumerate and detail all possible evasion tactics — they are a moving target. But make sure you factor WAF evasion into your deployment and policy management processes.

To address the semantic gaps you need to really *understand* each application and do a lot of testing. You need to understand how the application servers and databases interpret commands, and have enough knowledge of the WAF to know whether and how its responses differ. The process we defined in application lifecycle integration is instrumental to successfully avoiding sophisticated evasion techniques. This is also another reason why so many customers have had issues making WAFs truly useful — WAF admins need *both* application and WAF skills to block this kind of evasion.

And don't forget the importance of ongoing discovery of web applications. Strangely enough, not all WAF devices automatically detect traffic coming into a new application and automatically provide generic protections, instead you need to *explicitly* identify each new application to enjoy any application-layer protection. You need to sit on top of the WAF as it learns new URLs, make sure rules are in place, and be sure learned behaviors are quickly turned into policies to ensure those pages are protected. Your adversaries are doing discovery on your web properties *every day,* so you need to do the same. Waiting three months for learning won't cut it, so look for ways to accelerate the process. That may involve a manual review, using your vulnerability scanner to detect new applications, evaluating traffic through the WAF or access firewall, or maybe using a network discovery tool (such as `nmap`). Regardless of technique you need to identify new applications and pages, and apply rules *quickly*.

We advocate an ongoing testing process to ensure your policies are sound, your configuration hasn't drifted, and ultimately that you are blocking what needs to be blocked.

Hacking Yourself and Testing

Despite your best efforts, you still may miss something which leaves your applications exploitable. We advocate an ongoing testing process to ensure your policies are sound, your configuration hasn't drifted, and ultimately that you are blocking what needs to be blocked. Of course you can use scanners, evasion libraries (more kudos to Ivan Ristic for publishing an Evasion Techniques Catalogue), and other tools (such as Havij and SQLmap) to automate the testing process, but that's only half the story. To fully provide this kind of assurance, you need actual humans banging away at your applications.

Additionally, a number of compliance mandates require periodic penetration tests. One of the big sticking points is to what degree a pen tester can evade your defenses do what their job. Talk to enough pen testers, and their frustration at being glorified eunuchs (not being able to *really* attack targets) is evident. We share their frustration knowing that the bad guys don't operate under the same restrictions — they do not hold back when trying to evade the WAF, crack the application and exfiltrate your data. We suggest you check

out some of the work done under the Penetration Testing Execution Standard (PTES) to get a feel for how to structure your engagement to maximize value without crushing your systems.

None of this is perfect science. And nowhere have we said that WAF management is easy — quite the contrary. But the good news is that most attackers, after running into a well-configured WAF, will go elsewhere for lower-hanging fruit. The majority of attackers (not targeting a specific organization) don't have the time or inclination to hammer away at your site indefinitely. Raising the bar and closing the gaps against basic attacks and evasion techniques will save you many headaches. Staying on top of policies and continually probing your own systems for weaknesses gives you a much greater chance at success.

Summary

We have followed the Web Application Firewall (WAF) market for many years, so we understand WAFs' strengths and weaknesses. On the positive side, WAFs can protect applications from themselves — addressing insecure coding practices and dynamic attacks — and providing a simple, faster, and cheaper way to get started securing web applications. When you also factor in the variety of compliance mandates requiring use of WAFs, it's not so much a question of whether a WAF will be implemented, but more how to maximize its value.

Getting the most out of a WAF requires addressing the significant liabilities of WAFs. After almost a decade of use, technological limitations are rarely the most serious problems with underperforming WAF implementations. Most WAF problems can be traced back to problems with management — in terms of setting policies, integrating with the development process, and faulty deployments which put systems at risk. In this paper we have mapped out a three-step process for managing your WAF for maximum impact. But managing a WAF is an ongoing process which requires constant vigilance. WAF is not a set-and-forget technology.

When it's time to define WAF policies, the most important question is positive or negative? The answer is both. You should use learning mode to monitor application traffic and profile applications, in order to build the initial set of positive policies — which reflect and restrict what the application is allowed to do. Negative policies block attacks, and are typically managed by the WAF vendor (similar to IPS signatures).

The next key aspect of managing a WAF is to ensure a strong process links security and developers, to ensure WAF policies keep pace with application changes. If new application features are rolled out but the WAF isn't updated, the application is likely to break and the attack surface is likely to grow rather than being reduced. This requires not only working directly with developers, but also having sufficient executive support to hold up application deployment in order to address security issues if necessary.

Finally, understand that a WAF is a security device, and so it can and will be attacked. Secure deployment is essential, and testing it against both direct attacks and evasion attempts will help make sure the WAF does its job – all day and every day.

If you have any questions on this topic, or want to discuss your situation specifically, feel free to send us a note at info@securosis.com or ask via the Securosis Nexus (http://nexus.securosis.com/).

About the Analysts

Adrian Lane, Analyst and CTO

Adrian Lane is a Senior Security Strategist with 25 years of industry experience. He brings over a decade of C-level executive expertise to the Securosis team. Mr. Lane specializes in database architecture, data security and secure code development. With extensive experience as a member of the vendor community (including positions at Ingres and Oracle), in addition to time as an IT customer in the CIO role, Adrian brings a business-oriented perspective to security implementations. Prior to joining Securosis, Adrian was CTO at database security firm IPLocks, Vice President of Engineering at Touchpoint, and CTO of the secure payment and digital rights management firm Transactor/Brodia. Adrian also blogs for Dark Reading and is a regular contributor to Information Security Magazine. Mr. Lane is a Computer Science graduate of the University of California at Berkeley with post-graduate work in operating systems at Stanford University.

Mike Rothman, Analyst/President

Mike's bold perspectives and irreverent style are invaluable as companies determine effective strategies to grapple with the dynamic security threatscape. Mike specializes in the sexy aspects of security — such as protecting networks and endpoints, security management, and compliance. Mike is one of the most sought-after speakers and commentators in the security business, and brings a deep background in information security. After 20 years in and around security, he's one of the guys who "knows where the bodies are buried" in the space.

Starting his career as a programmer and networking consultant, Mike joined META Group in 1993 and spearheaded META's initial foray into information security research. Mike left META in 1998 to found SHYM Technology, a pioneer in the PKI software market, and then held executive roles at CipherTrust and TruSecure. After getting fed up with vendor life, Mike started Security Incite in 2006 to provide a voice of reason in an over-hyped yet underwhelming security industry. After taking a short detour as Senior VP, Strategy at eIQnetworks to chase shiny objects in security and compliance management, Mike joined Securosis with a rejuvenated cynicism about the state of security and what it takes to survive as a security professional.

Mike published The Pragmatic CSO in 2007 to introduce technically oriented security professionals to the nuances of what is required to be a senior security professional. He also possesses a very expensive engineering degree in Operations Research and Industrial Engineering from Cornell University. His folks are overjoyed that he uses literally zero percent of his education on a daily basis. He can be reached at mrothman (at) securosis (dot) com.

About Securosis

Securosis, LLC is an independent research and analysis firm dedicated to thought leadership, objectivity, and transparency. Our analysts have all held executive level positions and are dedicated to providing high-value, pragmatic advisory services.

Our services include:

- **The Securosis Nexus**: The Securosis Nexus is an online environment to help you get your job done better and faster. It provides pragmatic research on security topics that tells you exactly what you need to know, backed with industry-leading expert advice to answer your questions. The Nexus was designed to be fast and easy to use, and to get you the information you need as quickly as possible. Access it at <https://nexus.securosis.com/>.
- **Primary research publishing**: We currently release the vast majority of our research for free through our blog, and archive it in our Research Library. Most of these research documents can be sponsored for distribution on an annual basis. All published materials and presentations meet our strict objectivity requirements and conform to our Totally Transparent Research policy.
- **Research products and strategic advisory services for end users**: Securosis will be introducing a line of research products and inquiry-based subscription services designed to assist end user organizations in accelerating project and program success. Additional advisory projects are also available, including product selection assistance, technology and architecture strategy, education, security management evaluations, and risk assessment.
- **Retainer services for vendors**: Although we will accept briefings from anyone, some vendors opt for a tighter, ongoing relationship. We offer a number of flexible retainer packages. Services available as part of a retainer package include market and product analysis and strategy, technology guidance, product evaluation, and merger and acquisition assessment. Even with paid clients, we maintain our strict objectivity and confidentiality requirements. More information on our retainer services (PDF) is available.
- **External speaking and editorial**: Securosis analysts frequently speak at industry events, give online presentations, and write and/or speak for a variety of publications and media.
- **Other expert services**: Securosis analysts are available for other services as well, including Strategic Advisory Days, Strategy Consulting engagements, and Investor Services. These tend to be customized to meet a client's particular requirements.

Our clients range from stealth startups to some of the best known technology vendors and end users. Clients include large financial institutions, institutional investors, mid-sized enterprises, and major security vendors.

Additionally, Securosis partners with security testing labs to provide unique product evaluations that combine in-depth technical analysis with high-level product, architecture, and market analysis. For more information about Securosis, visit our website: <http://securosis.com/>.