Notes for Web Application Firewalls (WAF)

WAFs

Web Application Firewalls (WAF) are a newer technology that fills in some of the gaps that traditional firewalls are vulnerable to. Most simply, a WAF is able to disseminate internet traffic using domains, origin of packet, size of packet, frequency, IP ranges, and time of day, and block these HTTP packets using rule-based, anomaly-based, or signature-based detection policies to mediate external traffic to a protected server. WAFs are powerful because they are configured for the apps they protect specifically, and thus “speak the language” of that web app, allowing it to be able to identify unusual or disallowed request.

Traditional firewalls can not determine which URLs, directories, or parameters are acceptable, and which are suspicious, because their rulesets mostly apply to ports and services, and an HTTPS request containing a cross-site-script (XSS) or a SQL injection will look valid to a traditional firewall. Luckily, a WAF will be able to pick this traffic up and block it if necessary.

WAFs at OSi

Many of our clients have web application servers, most notably, and most familiarly to me, is LeadingAge. They have an entire network of web-based applications for LA members, partners, and LA staff from other states. They log on and fill out information on Drupal and Salesforce and other systems hosted on their domains. Recently, they just sent out their annual “Membership Survey”, which appears, at a cursory glance, to be just a massive opportunity for XSS and SQL injection, assuming they Simon or Jamall do not already have a dedicated WAF in place. Many of our other clients have web applications as well, but applications that revolve around user-entered data is especially vulnerable to nefarious threat actors exploiting those fields.

CloudFlare’s WAF will cover Drupal (in addition to other web app environments like PHP and WHMCS), which is used extensively for all non-staff internet access for LeadingAge members, partners, and clients.

Imperva’s SecureSphere focuses on Microsoft Azure and AWS hosted web apps. It encourages multiple web application gateways for each web app. It has commercial pricing (not open source) and it is ideal for larger organizations. Also touts end user auditing. Uses “dynamic profiling” to analyze normal web app interaction behavior, and crowd-sourced threat definitions.

Options

The three most popular WAFs are listed below.

* SecureSphrere, by Imperva
* CloudFlare WAF
* ModSecurity, by Trustwave

Imperva is typically for enterprise environments and is accordingly very expensive. ModSecurity applies only to Apache servers, and is open source. Cloudflare is application/cloud-based and allows dynamic profiling and threat intelligence feeds. According to their WAF information page, their rulesets are mostly a combination of OWASP rules and CloudFlare special rules, with the latter making up over 85% of the total.

The CloudFlare WAF is a “community WAF”, and while they offer different rules and focuses for different customers according to their needs, they treat the WAF as one entity that defends all clients, up to 5.5 million requests every second. While this figure is certainly dramatic, it does seem to be a popular option for web application servers.

Both Imperva SecureSphere, and CloudFlare’s WAF tout a <1ms latency for web visitors, and CloudFlare claims a 30s worldwide rule propagation time. Unlike the DNS name-server changes, this rule cannot be tested by bash scripts since the WAF is for enterprise environments only. Both services also meets PCI DSS requirements, which allows users to process credit cards on the WAF-protected site. Alternatively, without a WAF, one would need to conduct semiannual application vulnerability security reviews for all in-scope web applications.

TBoth WAFs protect against OWASP’s top 10 vulnerabilities, which are:

* Injection
* Broken Authentication and Session Management
* Sensitive Data Exposure
* XML External Entities (XXE)
* Broken Access Control
* Security Misconfiguration
* Cross-Site Scripting (XSS)
* Insecure Deserialization
* Using Components with Known Vulnerabilities
* Insufficient Logging & Monitoring