Varnish Security Firewall

2012-07-14 - Kacper Wysocki & Edward Fjellskål

High Voltage Protection For Your Webapps

Abstract

Varnish is the swiss army knife of the HTTP transport, and its flexible configuration language has long been used to thwart application attacks and DoS with custom rulesets. Varnish is fast, lightweight, aids in enabling applications to handle tremendous loads, and there are numerous security barriers in the architecture. The new Varnish Security Firewall framework enables us to rapidly secure web applications, and how it allows for easy and fast rule writing to enhance the security and quickly react to attacks. This approach lends itself well to securing cloud applications.

Varnish security barriers

The architecture in Varnish follows the principle of "zero trust", which means that the management process does not trust its children, which are run with the lowest possible privileges so they can securely handle untrusted user data. All communication channels with the child processes are well defined and defended by the manager.

Flexibile manipulation of HTTP requests

Since the Varnish Configuration Language allows programming the request flow, you can quickly and easily implement security policies in Varnish. VCL is compiled into machine code and yeilds high performance, allowing the handling of massive amounts of HTTP traffic.

Structured rules

Due to the programmatic nature of the configuration language, the security ruleset is structured around if-elseif statements, regular expression patterns and response handlers, as opposed to traditional security rulesets which are highly irregular, succinct and unstructured. This structure makes rules easy to comprehend in the context of the request flow, and to modify, update and expand the rules to cover new situations.

Each security rule gets an unique ID (the SID) to ease rule debugging and identifiability, as is the custom for security software. Concurrent development of rules is done by allocation of specific SID ranges to specific research groups. All rules have a name, description and severity level to aid in incident response.

Ease of writing rules

Writing rules are understandable when there are different search buffers with intuitive naming. Examples are "req.url", which is the request URI or "req.http.user-agent" which will match on the HTTP header field of the User-Agent.

Rule efficiency

When deploying VSF you take advantage of the highly optimized Varnish engine, which has been shown to handle large rulesets in highly loaded real-world production systems with traffic in the order of tens of thousands of requests per second [1]. Evaluating the rules is done super fast.

Realtime mitigation

The security module logs triggered rules to the Varnish Shared Memory Log, which can be monitored in real time, allowing a fast tactical overview of the current situation. The security ruleset can be configured in non-enforcing mode, where triggered rules are logged and passed through, allowing painless deployment in lieu of false positives. You can deploy the ruleset in production without fear of disturbing production systems, and examine realtime request data to determine whether any false positives need to be dealt with before enforcing the security policy.

In effect, you can deploy rules so they are only logging what they would have triggered on. Before you deploy a rule that would acctually block traffic, you can run the application firewall for a period of time to test the impact on performance as well as checking rules for false positives.

Request inspection

GET and POST requests can be decoded and inspected, on a header-by-header basis. This allows us to normalize traffic before it reaches the application, and to deny application access to clients based on access control lists as well as behavioral analysis.

Attack response

Rule handlers make it possible to handle a request triggering rules arbitrarily, and based on rule severity a request can be denied, dropped, fingerprinted, passed to an attack response script or even transparently proxied to a honeypot for analysis. Clients that continue to trigger rules can be put on a ban list in the style of fail2ban or OSSEC.

With the advent of Varnish Modules (VMODs) like vmod-shield and vmod-postgetcookieparse, and vmod-ratelimit, we can analyze POST requests and offer mitigation for DoS, XSS, SQL injection, and code injection attacks. We could also pass POSTed file attachments to a malware backend for processing before allowing it to pass to the application.

Cloaking

Privacy is an increasing concern in the age of deep packet inspectors and increasing internet-related litigation. Web service providers often wish to protect their service infrastructure by cloaking the server product and vendor information from clients. Some services also wish to wash their logs of any person-identifiable information such as User Agents and IP address. The VSF ruleset contains a cloaking module which allows us to cloak the client from the server, thereby aiding in user anonymity, and the server from the client, effectively obscuring the attack plane.

Cloud security

The advent of computing clouds like AWS and the need to commoditize infrastructure has driven the need to expand the security perimiter beyond corporate firewalls. Deploying VSF in a cloud environment allows us to enforce ACLs and security rules on a per-application basis rather than the traditional, and dated, source/destination host:port paradigm.

Previous work

Scarpellini's Varnish FireWall (VFW) [3], an OWASP [4] project and Scarpellini's master thesis [11], explored the efficacy of doing request and content inspection in Varnish 2.1 with inline-C modules. VFW was deployed on several hundred production systems and showed good protection and performance compared to the unprotected, Varnish-cached application. VFW also includes a web GUI for management.

Security.vcl [5] was the first attempt at a security framework for the Varnish web accelerator and together with VFW create the precursor to the Varnish Security Firewall, which the authors of these two projects have agreed to merge.

Tools such as p0f [6], WhatWeb [7], PADS [8] PRADS [9] and htrosbif [10] have been deployed standalone as well as in web- and email services for system and application fingerprinting, to various different aims. Such aims include attack surface enumeration, asset detection and service discovery.

Varnish has been employed for distributed denial of service mitigation in the past, as detailed in a Varnish Whitepaper [2].

Future work

Security rules for new attacks will be an ongoing effort, and some sort of community oriented around rule writing will support this effort. Normalization of requests is an open topic and needs further study to be properly efficient. The web GUI of VFW needs to be adapted for use in VSF. Better and more complete POST handling is on the wishlist. We would very much like to see passive fingerprinting and honeypot redirection implemented as part of VSF to better study new web application attacks.

Summary

VSF is a new libre software frontend security application which delivers a promising and unique amount of flexibility and performance. The approach has been proven in a number of production environments demanding high performance and stability while serving tremendous amounts of simultaneous requests. The Varnish engine is lightweight enough that it offers ease of deployment even in constrained, low-resource environments on last years commodity hardware, while still taking full advantage of tomorrow's powerhorse hardware. Take advantage of a already deployed Varnish infrastructure or speed up your sites in the same take as raising the security!

Footnotes

1 https://www.varnish-software.com/blog
2 Varnish Security Whitepaper, available upon request at https://www.varnish-software.com/about-us/whitepapers
3 Varnish Firewall: https://www.owasp.org/index.php/OWASP_VFW_Project
4 OWASP: https://www.owasp.org
5 Security.vcl: http://github.com/comotion/security.vcl
6 p0f: http://lcamtuf.coredump.cx/p0f3/
7 WhatWeb: http://www.morningstarsecurity.com/research/whatweb
8 PADS: http://passive.sourceforge.net/
9 PRADS: http://github.com/gamelinux/prads
10 htrosbif: http://useofwords.blogspot.no/2009/11/introducing-htrosbif.html
11 Scarpellini's Masters Thesis is available upon request.