VARNISH PLUS

VARNISH SOFTWARE

White Paper

Security in Varnish Cache

Protection Against Web Attacks

Security barriers in Varnish

Security is a very important design driver in Varnish. More likely than not, if you find yourself thinking “Why did Varnish do that?”. The answer has to do with security. The Varnish security model is based on some very crude but easy to understand barriers between the various components.

The really important barrier

The central actor in Varnish is the Manager Process, “MGR”, which is the process the administrator “(ADMIN)” starts to get web-cache service

Having been there myself, I do not subscribe to the “I feel cool and important when I get woken up at 3AM to restart a dead process” school of thought. In fact, I think that is a clear sign of mindless stupidity: If we cannot get a computer to restart a dead process, why do we even have them?.

The task of the Manager Process is therefore not to cache web content, but to make sure there always is a process which does that, the Child “CLD” Process. That is the major barrier in Varnish: All management happens in one process; all actual movement of traffic happens in another one and the Manager Process does not trust the Child Process at all. The Child Process is in a totally unprotected domain: Any computer on the InterNet “(ANON)” can connect to the Child Process and ask for some web-object. If John D. Criminal manages to exploit a security hole in Varnish, it is the Child Process he subverts. If he carries out a DoS attack, it is the Child Process he tries to fell.

Therefore, the Manager starts the Child with as low privileges as practically possible, and we close all file descriptors so it should not have access to them and so on. There are only three channels of communication back to the Manager Process: An Exit Code, a CLI Response or writing stuff into the Shared Memory File “VSM” used for statistics and logging; all of these are well defended by the Manager Process.

The ADMIN/OPER barrier

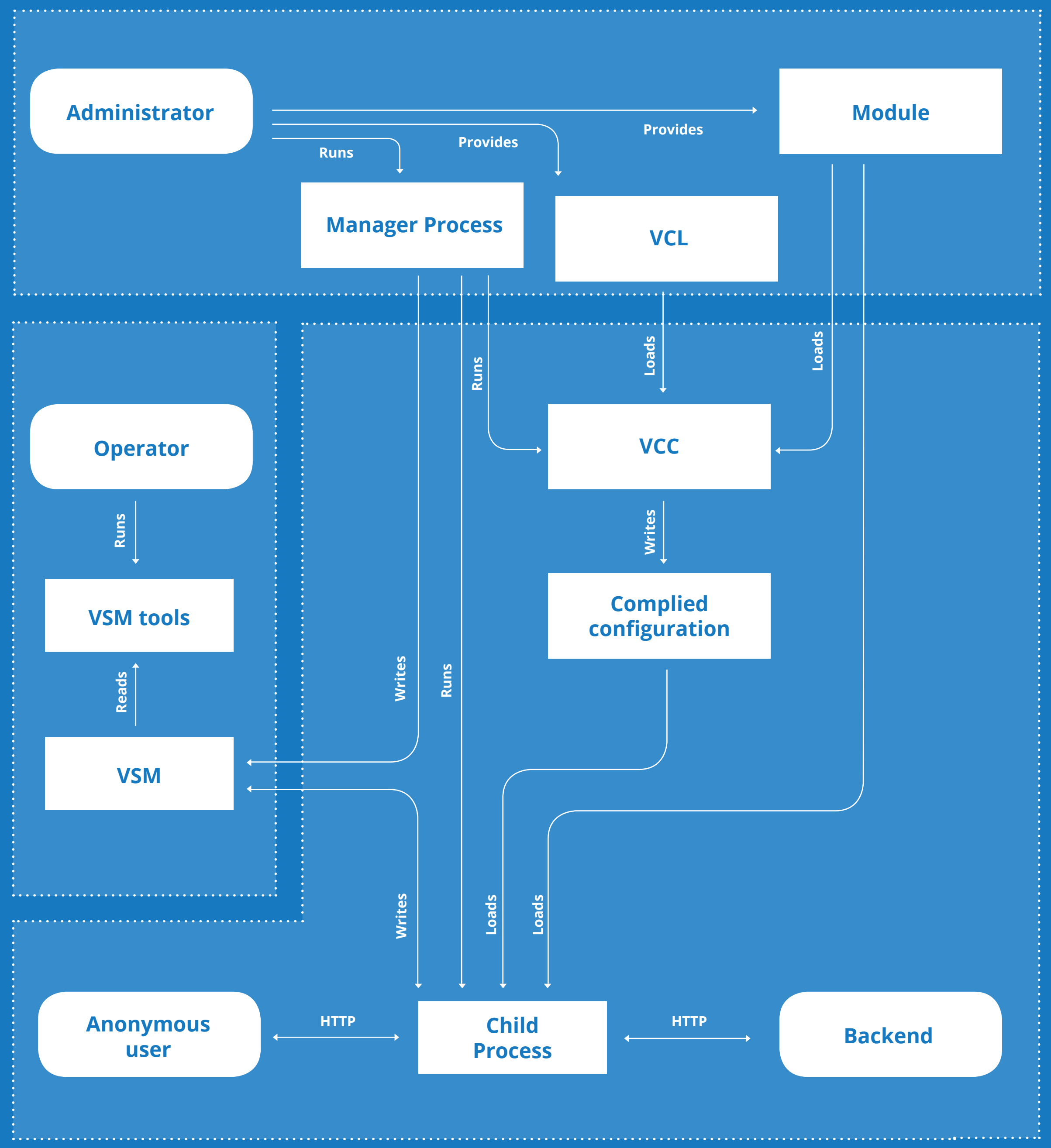
If you look at the top left corner of the diagram (Page 4), you will see that Varnish operates with separate Administrator "(ADMIN)" and Operator "(OPER)" roles. The Administrator does things, changes stuff etc. , while the Operator keeps an eye on things to make sure they are as they should be.

These days, Operators are often scripts and data collection tools, and there is no reason to assume they are bugfree. So, Varnish does not trust the Operator role, that is a pure one-way relationship. (Trick: If the Child process runs under user "nobody", you can allow marginally trusted operations personel access to the "nobody" account (for instance using .ssh/authorized_keys2), and they will be able to kill the Child process, prompting the Manager to start it again with the same parameters and settings. The Administrator has the final say and of course, the administrator can decide under which circumstances that authority will be shared. Needless to say, if the system on which Varnish runs is not properly secured, the Administrator's monopoly of control will be compromised.

All other barriers

There are more barriers that are more sort of "technical" than "political". Generally, Varnish tries to guard against programming flaws much the same way as security compromises. For instance, the VCC compiler runs in a separate Child Process, to make sure that a memory leak or other flaw in the compiler does not accumulate trouble for the Manager Process.

Diagram: Varnish Security Barriers



Alleviate attacks with Varnish

Varnish is something like the swiss army knife of HTTP. Since you can actually program Varnish with VCL, you can quickly and easily implement security policies in Varnish. VCL is compiled into machine code and runs extremely fast, giving you the possibility to process huge amounts of live HTTP traffic. One such use is meant to defend against DDOS attacks.

Using Varnish to fend of DDOS attacks

A denial of service attack can hit any web site without warning. There are different types of attacks, some exploit weaknesses in the TCP layer, some try to overrun the server by sheer volume of requests sent. Varnish can alleviate attacks by using the HTTP protocol. Either by helping you identify and stop the attacking requests or by giving you capacity to actually handle the incoming traffic.

Camiel Dobbelaar, CTO of Sentia BV describes how they used Varnish to defend a client against a distributed denial of service attack. Sentia BV is a Dutch hosting provider serving international customers with high availability requirements. In April 2010 Sentia BVs infrastructure was suddenly hit by a massive increase in traffic. The traffic was directed at a customer of Sentia. The traffic consisted of thousands of requests per second, looking just like normal traffic. Unlike a simple denial of service attack the traffic was originating from all over the world. Since the customer had a global reader base it wasn't possible to shut certain parts of the internet out without harming the normal traffic.

Closer inspection, by using the Varnish tools varnishtop and varnishlog, made it possible to identify the traffic programmatically. The attackers where sending the same User-Agent string with every request. This is one of several strings that a is sent along together with a HTTP request and gives some metadata about the request. Often, a denial of service attack can be identified by a certain header remaining fixed or having just a few variants.

" **With Varnish in place** it was easy for Sentia to trap the offendig requests and having Varnish answer with just an error page. The error page hardly uses any bandwidth and delivering it hardly consumes any resources.

With Varnish in place, it was easy for Sentia to trap the offending requests and having Varnish answer with just an error page. The error page hardly uses any bandwidth and delivering it hardly consumes any resources. VCL code to trap the user-agent 'Evilbot 3000' can be seen here:

```
sub vcl_recv {
  if http.user-agent == 'Evilbot 3000' {
    error(700);
  }
}
sub vcl_error {
   if http.status == 700 {
   set obj.http.Content-Type = "text/html; charset=utf-8";
   synthetic {"Please go away!"};
   return(deliver);
   }
}
```

Camiel did a few optimizations after the attack was under control. In vcl_deliver they stripped of almost every header, so the error page was as minimal thus freeing up bandwidth. They also started generating blacklists from the varnish logs, feeding these into their firewall. From that moment on, when one of the captured computers in the botnet sent one single request, it was blacklisted from their site saving further bandwidth and resources. The supporting tools that come with Varnish are built on standard Unix principles, so output can be piped and processed and the whole operation could be automated.

Handling the volume

Some attacks use real browser sessions to send requests. These are impossible to distinguish from real user traffic. If the attack is small or mid-sized you might actually manage to handle the attack by adjusting your caching policies and making sure your hit rate is approaching 100%. Having a specially crafted VCL ready, which sets the TTL of all objects to a week, might mean that you'll serve stale content - but at least your site will stay up. Take special care to make sure various forms and other POST requests get handled in an appropriate manner.

Varnish can usually deliver content at the speeds of your network so if you have enough bandwidth available you'll just answer all the incoming requests. Hopefully, your attackers will tire and direct their attack somewhere else, when they see that your site is still up.

Secure your servers with security.vcl

Web application firewall (WAF) was a term coined a few years ago describing a firewall that is operating on the application layer. These firewalls act as filters looking for suspicious patterns and blocking them out. In 2008, we took the predefined patterns implemented in by the insightful people of mod_security and rewrote them to VCL. Since VCL is compiled to native code, we are able to run most of these filters with almost no performance impact. These patterns look for attack signatures in URLs or headers. A lot of commonly used SQL injection attacks and access to deployed root kits can be stopped this way, either rendering the attacks futile or preventing the black hats from actually taking advantage of your servers.

Since Varnish is deployed at the edges of your network, implementing changes in your Varnish configuration is much less error-prone then deploying changes to your core web applications. In most cases, a change in Varnish can often be deployed much quicker than a full blown deployment of your web application. Often the change is minimal, blocking a certain set of URLs or adding missing escape characters. Then, after you have deployed the hot fix in Varnish you can work on doing the proper fix in your web application without shutting your site down for hours.

Example security.vcl rule

An example of a rule taken from the trojan detection module in security.vcl looks like this:

```
sub vcl_recv {
  set req.http.X-Sec-Module = "2vcl";
## REQUEST_HEADERS_NAMES,
## Rule: REQUEST_HEADERS_NAMES rx :
## REQUEST_FILENAME,
## Rule: REQUEST_FILENAME rx :
if(req.url ~ "root\.exe"){
   set req.http.X-Sec-Return = "404";
   set req.http.X-Sec-RuleInfo = "Backdoor access";
   set req.http.X-Sec-RuleName = "MALICIOUS_
SOFTWARE/TROJAN";
   set req.http.X-Sec-Severity = "2";
   set req.http.X-Sec-RuleId = "950921";
   call sec_sev1;
 }
}
```

This is one of the simpler rules, which just matches the URL against the string root. exe - which is a known trojan, often left behind as the result of a successful attack on your web site.

These are three aspects of Security in Varnish. If you want to learn more, please go to: **http://www.varnish-software.com/.**

About Varnish Software

Varnish Software is the provider of the open source web acceleration software, Varnish Cache. Varnish Cache is a reverse HTTP proxy that will speed up a website by storing a copy of the page served by the web server in its cache the first time a user visits that page. The next time the user requests the same page, Varnish will serve the copy instead of requesting the page from the web server. This means your web server needs to handle less traffic and your website's performance and scalability go through the roof. In fact, Varnish Cache is often the single most critical piece of software in a web based business.

Varnish Software delivers a comprehensive range of enterprise products and services that will help customer further increase the performance and scalability of their websites. Leading websites all over the world rely on Varnish, including BBC, Vimeo, Nokia, Business Insider, Morningstar LiveJournal, Qt/Digia, The Globe and Mail, and The Hindu.

The Varnish open source project began in 2005 as an idea within VG Multimedia, Norway's largest online newspaper. Varnish Software was then founded in 2010 and now has offices in Oslo, London and New York City

VARNISH PLUS+

VARNISH SOFTWARE

New York	+1 201 857 2832
Los Angeles	+1 310-648-8474
Paris	+33 607 47 36 90
London	+44 20 7060 9955
Stockholm	+46 8 410 909 30
Oslo	+47 21 98 92 60