

**SSL Everything:
Protect all of your website,
not just a few parts**

Chances are, if you're utilizing SSL, you're not utilizing it to its full potential. SSL is a powerful technology that can help organizations protect their data as well as their users. While the technology behind SSL is solid, the most common best practices for its implementation do not take full advantage of the benefits that SSL brings. And this may be inadequate to provide proper security to the modern web application environment.

SSL 101

To explain, let's first quickly review what SSL technology gives an organization in terms of benefits. These benefits are *trust* and *privacy*.

Trust is provided through the use of a CA (Certificate Authority), such as VeriSign or Thawte. By having your SSL certificate signed by a CA, no one else can use your domain name with SSL without showing some sort of error.

Privacy is provided through encryption, which scrambles traffic so that it is incomprehensible to a potential eaves dropper. With regular HTTP, information such as usernames, passwords, credit card information are all sent over the Internet in plain-text. Someone could potentially "sniff" this traffic, which is the electronic equivalent of eaves dropping on a conversation. What's more, this process can be automated so that it can be done unattended. A program automatically sniffs the network, and pulls out parts that are programmed

to be interesting, such as passwords and credit card numbers. So encryption is one of the most powerful tools utilized in e-commerce.

SSL in practice

The ability to provide trust and privacy allows an organization to protect sensitive information. Currently, SSL is used primarily to protect just a few types of sensitive information. These include:

- Passwords (during login)
- Credit card transactions - which is a PCI (Payment Card Industry) compliance requirement

Most of what SSL is used for can be classified in one of those two ways. But is that the only type of sensitive data that could benefit from SSL? Consider the following:

- Memos on corporate strategy (web mail)
- Confidential sales information (CRM)
- Trade secrets (Web portal)
- Session identifiers (in the HTTP cookie)

With the exception of VPN technology that may or may not be in place, most of this information isn't typically protected in most enterprises. It's all done with regular non-SSL HTTP.

And then think for a moment, what isn't data could you send over the Internet that isn't sensitive? That is, what data would you not care about

if it fell into the hands of competitors or the public.

It could even be that most of your data is non-sensitive, but some of it will invariably be, so how are you to know what is and what isn't? So why not protect it all?

Session Protection

There's another type of data that your users might want protected, although most don't know about it. This type of data is called a "session cookie". When the web protocol was developed, it was initially "stateless", in that each request was independent of each other. As the web moved from static web pages to interactive web applications, there needed to be a way to create a relationship between the user and the server, where the web application would customize responses specifically to that user. This was done through assigning the user a session ID, typically in the form of an HTTP cookie.

When a user logs into a web application, they're assigned a session ID, typically a long random string of letters and numbers. The user then sends this session ID to the server every time it makes a request. The server is then able to dynamically create content (such as a custom portal page, private email account, etc.) for that user.

A problem occurs if an attacker were to get hold of this session ID value. They could then hijack a user's session, pretending to be that user without any login challenge from the server.

If only the initial login and perhaps credit card transactions are done with SSL, that means this session ID value is sent repeatedly over the network "in the clear" (without encryption). This makes it relatively easy to eavesdrop this particular data, because there's no privacy.

So what's the catch?

So, given the benefits of SSL and the nature of much of the data being transferred, why isn't SSL used as the default? To answer this question, we need to look at the history of SSL and how usage developed.

When it was first created, the primary drawback to SSL was that it required more horsepower in terms of the server's resources. Encrypting and decrypting is a CPU-intensive operation. What's more, general purpose CPUs (such as x86 processors) are not optimized for this task. This is still true today, even for modern CPUs.

As you browse a website that is operating with SSL, you wouldn't notice this increased CPU utilization required to decrypt a web page coming from a server. The few connections you have open will not make a significant impact on even modest modern web user systems.

But think of it from the server's perspective: It's serving up dozens, hundreds, or perhaps thousands of simultaneous connections. If the server is trying to encrypt everything as well, the CPU is going to be overwhelmed.

Servers have to work a lot harder when serving up pages protected with SSL. Imagine that you've got racks filled to the brim with servers to serve up your web applications. But instead of using their CPUs to deliver your applications, they spend a considerable portion of their available resources doing the encryption and decryption. It's like bringing in a crack team of brain surgeons, and inundating them with mundane paperwork so they can perform only half the surgeries. It's just a waste of valuable resources.

So the traditional way of thinking in regard to SSL is that of a scarcity mentality. Because SSL resources are considered so finite, only the most critical portions of a user interaction are then encrypted. After using SSL for taking a credit card number or authenticating a password, you're redirected right back to the non-SSL portion of a site.

SSL and Application delivery - 101

To address these limitations in SSL delivery, systems designers developed a new technology – the SSL ASIC (Application Specific Integrated Circuit). ASICs are like typical processors, but instead of being designed to perform a wide range of tasks (playing games, running a word processor, browsing the Internet), ASIC processors are extremely limited in their functionality. In the case of SSL ASICs, they're limited to only performing SSL operations, but they are phenomenally good at this task. Whereas an x86 CPU might be able to handle 20 new SSL connections in

a second, an SSL ASIC would be able to handle 2,000 new SSL connections.

Ironically, Intel itself was among the first to market an SSL accelerator, as even Intel realized that the x86 CPUs were inefficient at high rates of SSL communications.

Among the first places these specialized SSL processors found usage was in the servers themselves. A PCI card with an SSL ASIC would be installed, and the web serving software would use these processors instead of the general CPU to do the SSL work. The server would be able to provide the benefits of SSL services without the performance penalty. However, the drawback to this method is that you need one SSL card per server, and that cost can add up quickly. If each card costs \$700, and there's 20 servers, that's \$14,000 just for SSL.

Then special network appliances, called SSL accelerators, were developed and brought to market. SSL accelerators are devices that sit in between a user and a server, accepting SSL connections from the user, and sending them to the server unencrypted. Data sent over the public network is protected, and the server sees unencrypted traffic, so no CPU resources are used on SSL.

This model proved to be very successful. So successful in fact, that load balancers were combined with SSL accelerators, so they now operate in the same chassis, and that hybridization is commonly

referred to as an ADC (Application Delivery Controller).

Because the SSL session is terminated at the ADC, the ADC is able to perform the identical Layer 7 functions that are done with non-SSL HTTP, including content switching, cookie persistence, and intrusion detection/prevention (IPS).

That's where the LoadMaster series of ADCs from KEMP Technologies comes in. A KEMP LoadMaster is located in front of a group of servers, and provides SSL acceleration as well as intelligent load balancing and content switching.



Abundance mentality

With this new technology, it is now feasible to apply SSL to *your entire* web infrastructure. The SSL processors handle the SSL, and the servers spend their time doing what they do best. This frees up server resources to handle other tasks.

So why isn't this more common? Part of the reason is that SSL has long been equated with high CPU utilization and that thinking hasn't changed. But it's about time it did.

The power of hardware

The LoadMaster 2500 and LoadMaster 3500 include integrated SSL processors. The Loadmaster 2500 can handle 1,000 TPS

(transactions per second), and the Loadmaster 3500 can handle 2,000 TPS.

LoadMaster	LM1500	LM2500	LM3500
SSL TPS	100 TPS	1,000 TPS	2,000 TPS
TPS Throughput (Assuming 100 kilobyte page size)	80 Mbps (maximum device throughput)	350 Mbps (maximum device throughput)	1 Gbps (maximum device throughput)
Additional cost for SSL	\$0	\$0	\$0

But is this enough? Is this enough SSL horsepower to move not just the passwords and credit card SSL onto the LoadMaster, but the entire site as well? And how does SSL TPS translate to other, more familiar web metrics? To answer these questions, let's take a look at the fairly well understood parameters of page size and bandwidth.

First, let's talk about what TPS means. The "T" in TPS refers to the initial SSL connection. This is by far the most CPU intensive part of the entire process. The initial connection is where most of the work is done, and where having an SSL accelerator is really beneficial.

By taking a given web page size and a little math, you can get an idea of the bandwidth than a given level of TPS might push. As an example, we'll use a web page size of 100 kilobytes (including HTML and images, which is an average page size). With a page size of 100 kilobytes, how much bandwidth will 1000 TPS create? We're assuming HTTP 1.1, so that all 100 kilobytes of objects are requested within a single

connection (and thus a single SSL transaction). 100 kilobytes equals 800 kilobits, or 800,000 bits. Multiply by 1,000, and we get 800,000,000 bits possible within a single second. This translates into 800 Megabits. This is a tremendous amount of traffic. More than what you would need for the LM-2500. However, the 2,000 TPS available in the LM-3500 translates to over 1 Gigabit, which is more than you would need. Different page sizes and user browsing behaviors will affect these numbers of course, but they're a good indication of the type of performance you can expect. The bottom line: The level of TPS available in the LM-2500 and LM-3500 is more than you would likely need, so TPS will not be a limiting factor. That is plenty of SSL power for virtually all small-to-medium sized businesses (SMB). And with the SSL licenses included, the additional cost to "SSL-protect" your *entire* site is zero, as opposed to adding individual SSL cards to each server.

An easy choice

With hardware-based SSL processors and their ability to offload SSL work for entire web infrastructures with no degradation in performance and capacity, and no additional cost (compared with non-SSL), why not SSL everything? You get all of the benefits, and none of the drawbacks.

With the power of SSL processors integrated into the application delivery controller at no additional cost, there's no reason not to SSL your entire website. The cost is zero, potential risks mitigated, and your users, if they notice a difference, will notice one for the better.