

SSL with Mon Guide

Last Update: 2001-08-07

Thomas Morin
(c) Webmotion Inc. 2000-2001

Table of Contents

<u>1. Introduction</u>	1
<u>1.1. Context</u>	1
<u>1.2. Why SSL ?</u>	1
<u>2. Server side ("Slave")</u>	2
<u>2.1. Principle</u>	2
<u>2.2. Software used</u>	2
<u>2.3. Configuration</u>	2
<u>3. Client Side ("master")</u>	3
<u>3.1. Patching Mon::Client</u>	3
<u>3.2. Dependencies</u>	3
<u>3.3. Examples</u>	3
<u>4. Client authentication with SSL</u>	5
<u>4.1. Introduction</u>	5
<u>4.2. On the mon master side</u>	5
<u>4.3. On the mon slave side</u>	5

1. Introduction

The current version of this documentation can be found at http://people.webmotion.com/~thomas/monssl/monitoring_ssl_with_mon.html

1.1. Context

The context is a situation where we need to monitor services on a distant site by using a master/slave scheme (where a mon master server connects to a mon slave server to retrieve the status of services tested locally on the slave, like free FS space, memory or CPU load) and want to encrypt and authenticate connections to the distant server (to avoid both eavesdropping, potential exploits of failure of the mon server, or whatever thing you fear).

In this context the mon master server is a mon client to the mon slave server. Typically the mon server will have a service using something like the `remote.monitor` script.

1.2. Why SSL ?

The same encryption/authentication could probably be done in a similar way with ssh and RSA keys authentication, but I wanted a more general/transparent solution : with this solution, you can quickly set up a secure unique frontend to many distant mon servers, just by using the `use_ssl` option of [Mon::Client](#). For scalability and ease of configuration, I also think it's best not to have to build one ssh tunnel per host you want to poll.

And last point, I wanted to learn more about SSL. ;)

2. Server side ("Slave")

2.1. Principle

In a first time, because it was easier at first to just implement the client side of the ssl protocol, I chose to use an ssl wrapper to wrap the access to the mon TCP port on the mon slave. It could also be possible to implement SSL directly in the mon server, which may be done one in the future, but here I wanted a quickly working solution, and a one which wouldn't require to patch every mon installation on each slave.

The wrapper accepts SSL connections on a `monssl` TCP port (eg 25830) and then locally forwards the content of the socket to the local mon server on the standard mon TCP port (2583) – which is, as you can guess, not accessible from the outside, you can use the `serverbind 127.0.0.1` configuration directive to tell mon to only bind the loopback interface.

The SSL wrapper can perform SSL client authentication by checking that their client SSL certificate is in it's "access list", and/or that it is signed by a trusted party.

2.2. Software used

I found STunnel (www.stunnel.org), and it very well suits the need of our application, it uses the *OpenSSL* library and seems actively maintained.

Note: the security of the setup describes here depends partly on the security of that software. I'd be very happy to have any feedback on that. Stunnel version prior to version 3.9 have a format string remote hole.

2.3. Configuration.

Just run `stunnel` this way : `stunnel -d 25830 -r 2583`

<i>option</i>	<i>description</i>
<code>-d 25830</code>	listen on TCP port 25830 for SSL connections
<code>-r 2583</code>	forward on local port 2583

3. Client Side ("master")

3.1. Patching [Mon::Client](#)

Since we want the most transparent solution, the thing to do was to implement the SSL client-side part in the perl *Mon* client module.

So you'll need to patch your Client.pm file with the `ssl-mon-client-patch` file :

In the Mon directory of the Mon::Client source tree, with a GNU patch utility :

```
patch Client.pm < ssl-mon-client-patch
```

This patch adds new options to the [Mon::Client](#) object :

<i>name</i>	<i>description</i>	<i>default value</i>	<i>note</i>
ssl	use ssl ?	0 (of course)	
ssl_pkey	path to the private key	/etc/mon/ssl/key.pem	<i>only required for SSL client authentication</i>
ssl_cert	path to the client certificates	/etc/mon/ssl/cert.pem	<i>only required for SSL client authentication</i>

3.2. Dependencies.

This patch requires the [Net::SSLeay](#) perl module (available from CPAN and others), which is a perl binding of the *OpenSSL* library. For smooth operation, you'll want a recent version of [Net::SSLeay](#) (> 1.05) .

3.3. Examples

Once all those things set up, to use SSL with mon in a script like the `remote.monitor` you just have to had the `ssl` option to the instantiation of the Client object (and use the right port for SSL):

```
$c = Mon::Client->new(  
    "ssl"    => 1  
    "port"   => 25830  
);
```

You can use the `remote.monitor-ssl-patch` (against version 1.2 of `remote.monitor`) to add the `ssl` feature to `remote.monitor`, it just add a `-s` flag which tells `remote.monitor` to use SSL, and give it the right port for SSL :

```
$ ./remote.monitor -p 25830 -s host  
foobar.webmotion.com
```

```
Details for foobar.webmotion.com failure :  
Watch servers service syslogd failed
```


4. Client authentication with SSL

4.1. Introduction

When doing public key based cryptography it is very important that peer authentication is made at some point, to avoid man in the middle attacks (see <http://www.monkey.org/~dugsong/dsniff/> if you doubt it). With SSL but no peer authentication, your setup will be more secure than without, in the sense that breaking into it won't be that easy, but it won't be cryptographically secure.

We could have had the mon master check the certificate of the slave it is connecting to, but this would require heavier patching of [Mon::Client](#), and my goal was to avoid this.

What we can easily do is use SSL certificates to allow only the master (or masters) to access the slaves, by telling `stunnel` – which runs on the slave – to check that the certificate the clients use are trusted ones.

With client authentication, you also prevents anyone but the trusted ones to access your mon slave servers.

4.2. On the mon master side

You'll have to have an SSL certificate for the server, it can be a self signed certificate, an official and expensive certificate that you bought from a "trusted" party, or a certificate signed by a CA certificate that you created for yourself.

Then you'll have to place the cert and its key (preferably unencrypted!) in the default places (`/etc/mon/ssl/{cert|key}.pem`) or elsewhere, but then you'll have to use the `ssl_pkey` and `ssl_cert` parameters at the instantiation of the [Mon::Client](#) object.

4.3. On the mon slave side

You'll have to start `stunnel` with the right parameters. The options to play with are `-v`, `-a` and `-A` .

For instance, I have a working setup with certificates signed by a CA. On the master side, the SSL certificate of the master server is present at the default place, and I have a `remote.monitor` patched to use SSL which will use this certificate.

On the slave, `stunnel` is started this way :

```
stunnel -v 3 -A /path/to/rootCa.pem -d 2538 -r localhost:2883
```

With this `-v 3` option it will check the validity of the client certificate against the root CA certificate given. Then it will check that the certificate presented is a trusted one.

For more details on certificates use with `stunnel`, see this excellent FAQ : <http://www.stunnel.org/faq/certs.html#ToC1>

The modssl FAQ also has very practical answers to certificates generation questions:
http://www.modssl.org/docs/2.7/ssl_faq.html#ToC24