Running wolfSSL on Cavium Octeon

Running wolfSSL on Cavium Octeon

The powerful combination of wolfSSL and Octeon make for an exceptionally speedy and secure SSL/TLS protocol handling. The goal of this tutorial is to get wolfSSL running on a multi core Cavium Octeon-I, Octeon-II, or OCTEON-III device.

This tutorial has been divided into three parts. The first part covers the installation of wolfSSL on a Linux host, cross-compile it for Octeon and get it running on Octeon-Linux. The second part will invoke the Octeon encryption hardware by patching the wolfSSL source code. Using encryption hardware gives an acceleration of up to 68 times! For this step to complete you must have access to the Octeon crypto-api. The third step introduces Sirius, an Octeon TCP/IP stack. This stack unleashes the real power of the Octeon chip. Together with wolfSSL it will allow multiple cores to process numerous SSL connections simultaneously.

Audience

This paper is intended for software developers and systems administrators who need to implement an SSL/TLS stack on a Cavium Octeon device. It gives a step by step guide on how to modify the wolfSSL code and the Octeon crypto-api to take full advantages of the multi core Octeon. Each step will be concluded with a test to verify all modifications were conducted correctly.

Prerequisites

To complete this tutorial we assume the following

- You are familiar with the Octeon device and its SDK.
- You have both the SDK and some hardware with an Octeon at your disposal. (You can get a free SDK at www.cnusers.org. The only thing lacking is the simulator).
- You have installed SDK 2.3 or higher on a Centos 6.6 Linux host PC and a version of Busybox Linux running on the Octeon and you are able to load applications via a flash card or NFS onto the Busybox Linux. Older SDK versions might work but have not been tested. The same goes for the OS on the Linux host; other version will probably work, but have not been tested.
- If you want to use the hardware acceleration you need a copy of the Octeon crypto-api core. Contact Cavium for this (www.cavium.com). To use encryption hardware acceleration make sure the type of the Octeon chip is AAP or SCP. If your device does not have an encryption hardware accelerator, using wolfSSL on Octeon is still possible, but you will not benefit from the acceleration created by the encryption engine. Using multiple cores instead of a single core will give the extra performance boost.
- If you want to go multi core, use the Sirius TCP/IP stack. Contact Phact BV for a copy of the Sirius library (www.phact.nl or www.sirius-networks.com).
- The echo server example discussed in part 3 of this tutorial is written in C++ so your Octeon SDK has to be capable of compiling and linking C++ files. If not, you can find a tutorial that explains how to modify your SDK so it will digest C++ files as easy as C files in the white paper section on our website www.sirius-networks.com.

Part 1: Getting started

This part of the document covers the basic installation of wolfSSL on your Linux host and getting wolfSSL's benchmark test running on Octeon.

Download the wolfSSL source code from the wolfSSL website (https://www.wolfssl.com/wolfSSL/download/downloadForm.php) and save it to a directory which is convenient for you, for example */root/wolfSSL/.*
Unzip this file by typing

#unzip wolfSSL-x-x-x.zip

where x-x-x is the version you downloaded. Step down into the directory that was created by the unzip utility (wolfssl-x.x.x) and build the system for x86 by typing

#./configure --enable-dh --enable-sha512
#make

Run their test suite

#./testsuite/testsuite.test

The output of the testsuite test should be something like this

MD5 test passed!
SHA test passed!
SHA-256 test passed!
SHA-384 test passed!
SHA-512 test passed!
HMAC-MD5 test passed!
HMAC-SHA test passed!
HMAC-SHA256 test passed!
HMAC-SHA384 test passed!
HMAC-SHA512 test passed!
ARC4 test passed!
DES test passed!
DES3 test passed!
AES test passed!
RANDOM test passed!
RSA test passed!
DH test passed!
SSL version is TLSv1.2
SSL cipher suite is TLS_DHE_RSA_WITH_AES_256_CBC_SHA256
SSL version is TLSv1.2
SSL cipher suite is TLS_DHE_RSA_WITH_AES_256_CBC_SHA256
Client message: hello wolfssl!
Server response: I hear you fa shizzle!
sending server shutdown command: quit!
client sent quit command: shutting down!
ciphers = RC4-SHA:RC4-MD5:DES-CBC3-SHA:AES128-SHA:AES256-SHA:DHE-RSA-AES128-SHA:DHE-RSA-AES256-SHA:AES128-SHA256:AES256-SHA256:DHE-RSA-AES128-SHA256:DHE-RSA-AES256-SHA256
9a104af3d164e37c0dabe3316f7bc35b9be5fd771a09ea1cda478d4057f0b06e input
9a104af3d164e37c0dabe3316f7bc35b9be5fd771a09ea1cda478d4057f0b06e /tmp/output-8VBJFr
All tests passed!

Now we will configure and build for Octeon, but we will not use the Octeon hardware encryption engine yet.

#./configure --host=mips64 CC="mips64-octeon-linux-gnu-gcc -mabi=64" --enable-dh --enable-sha512
#make

Transfer the *testsuite.test* file, the test files *quit* and *input* and the *certs* directory from your Linux host machine to Octeon Busybox Linux. Make sure the date and time on your Octeon Linux device is valid. If necessary, change it

#date -s "2015-03-24 11:47:00"

Run the test suite application from the Octeon device on a single core

#schedtool –a 0x1 –e testsuite.test

The output should be similar to the output created when testing on the Linux host PC.

Now copy the benchmark test application to the Octeon system (it's located in wolfcrypt/benchmark) and run it.

#schedtool –a -0x1 –e benchmark

Be patient, depending on the speed of the processor it may take some time to get the first results. The output of the program should be something like

AES 5 megs took 0.567 seconds, 8.683 MB/s
ARC4 5 megs took 0,137 seconds, 36.567 MB/s
3DES 5 megs took 1.987 seconds, 2.516 MB/s
MD5 5 megs took 0.093 seconds, 53.760 MB/s
SHA 5 megs took 0.094 seconds, 53.245 MB/s
SHA-256 5 megs took 0.258 seconds, 19.348 MB/s
SHA-384 5 megs took 0.280 seconds, 17.878 MB/s
SHA-512 5 megs took 0.247 seconds, 20.279 MB/s
RSA 2048 encryption took 16.639 milliseconds, avg over 100 iterations
RSA 2048 decryption took 150.919 milliseconds, avg over 100 iterations
DH 2048 key generation 54.941 milliseconds, avg over 100 iterations
DH 2048 key agreement 65.811 milliseconds, avg over 100 iterations

At this point, you have built and run wolfSSL on a single core and all encryption was done without any hardware acceleration. Up until now, there is no advantage of using an Octeon for this, as any CPU capable of running Linux will do the trick. But Octeon has much more in store, so let's move on.

Part 2: Speeding up, using the Octeon hardware encryption engine

In part 1, we have installed wolfSSL on our Linux Host PC, cross-compiled and built it for Octeon and got some benchmark results. These results were created without using the hardware encryption engines of the Octeon. In this part we will modify the source code of wolfSSL so it incorporates the hardware encryption engine and we will re-run the benchmark. Needless to say there should be "some" improvements.

To successfully walk through this part of the tutorial, you need an installed Octeon SDK and the crypto-api core. If your Octeon device does not have an encryption engine, you can still use wolfSSL on Octeon, but it will not benefit from the hardware acceleration. In this case, skip to part 3.

Get the patch and the configure file for modifying your version of wolfSSL from the download section of www.sirius-networks.com and put them in the directory that was created by the unzip utility.
In our case this is */root/wolfSSL/wolfssl-x.x.x*. Apply the patch

#patch –p1 < sirius_wolfssl-x.x.x.patch

Look for the settings.h file in the *wolfSSL/wolfssl/wolfcrypt* directory and uncomment the OCTEON define.

Replace the configuration file configure in the *wolfssl-x.x.x* directory with the version you downloaded from our website. Go to the *wolfssl-x.x.x* directory and configure the system for Octeon with hardware acceleration

#./configure --host=mips64 CC="mips64-octeon-linux-gnu-gcc -mabi=64" --with-octeon --enable-dh --enable-sha512

Before building wolfSSL for Octeon we need some libraries from the Octeon SDK that access the Octeon hardware. These libraries are generated when the test-api example is build. (Important: make sure the flag OCTEON_OPENSSL_NO_DYNAMIC_MEMORY has been disabled in crypto.mk!) The generated libraries are *libcrypto-linux_64.a, libcvmx.a* and *libfdt.a*. For building the test-api example and creating the libraries, do the following:

#cd $OCTEON_ROOT/components/crypto-api/core/test-api
#make OCTEON_TARGET=linux_64 clean
#make OCTEON_TARGET=linux_64

When using SDK3.x, you need to perform an extra step to complete the library build. The object *cvmx-config-init.o* is not present in the *libcvmx.a*, but it is needed by the benchmark examples. So the missing object must be added to the *libcvmx.a* library

#cd obj-linux_64-octeon2
#mips64-octeon-linux-gnu-ar –r libcvmx.a cvmx-config-init.o
#cd ..

Find the location of the libraries you just created:

#find $OCTEON_ROOT -name libcrypto-linux_64.a
#find $OCTEON_ROOT -name libcvmx.a
#find $OCTEON_ROOT -name libfdt.a

And copy these libraries to the wolfSSL directory

#cp obj-linux64/*.a /root/wolfSSL/wolfssl-x.x.x/lib/.

Go back to the */root/wolfSSL/wolfssl-x.x.x* directory and build wolfSSL for Octeon again

#make

Transfer the benchmark to your Octeon system and run it on a single core

#schedtool –a 0x1 –e ctaocrypt/benchmark/benchmark

The output should be something like this

```
AES          5    megs   took    0.019 seconds,          268.140 MB/s
ARC4         5    megs   took    0.073 seconds,          68.673 MB/s
3DES         5    megs   took    0.031 seconds,          161.394 MB/s
MD5          5    megs   took    0.025 seconds,          200.136 MB/s
SHA          5    megs   took    0.018 seconds,          275.664 MB/s
SHA-256      5    megs   took    0.025 seconds,          201.410 MB/s
SHA-512      5    megs   took    0.017 seconds,          290.799 MB/s
RSA  2048    encryption took     0.311 milliseconds,     avg over 100 iterations
RSA  2048    decryption took     7.174 milliseconds,     avg over 100 iterations
DH   2048    key generation      18.810 milliseconds,    avg over 100 iterations
DH   2048    key agreement       18.830 milliseconds,    avg over 100 iterations
```

Comparing these results with the results of testing without hardware acceleration gives a good impression of the extra speed Octeon brings to wolfSSL. AES is 30 times as fast in hardware as it is in software, 3DES gives an increase of 68 times.

(All tests were conducted on a CN5860-SCP running at 600MHz)

Part 3: Going down the fast lane - using Sirius and multi core

In part 1 we have installed wolfSSL on our Linux Host PC, cross-compiled and built it for Octeon and got some benchmark results. In part 2 the wolfSSL source code was changed to include Octeon's encryption engine and the benchmark tests were executed again. This turned out to be up to 68 times faster compared to using a software only solution.

In this part we introduce the Sirius TCP/UDP/IP stack which unleashes the real power of Octeon. Using this stack and the accompanying example program, it is easy to build a full featured high performance SSL/TLS multi core echo server. This echo server can be used as a start for your own application. It can run as Simple Exec or on Linux. In this example we will use Linux so a file system will be at our disposal. A file system comes in handy because it can hold a X509 certificate and we can easily load it using standard file commands.
Due the setup of the Sirius stack and the way it uses the atomic tagging mechanism of the Octeon, performance will increase linearly with the number of cores used to run the echo server.

No hardware acceleration onboard

When your Octeon device does not have encryption engines on board, it still is possible to benefit from wolfSSL on multi core Octeon. In this case, comment the *#define OCTEON* in the settings.h file. This file is located in the */root/wolfSSL/wolfssl-x.x.x/wolfssl/wolfcrypt* directory.

Important notice

When using Linux on Octeon, one might want to use the Octeon-driver that comes with the SDK. This driver interfaces Linux to the hardware of Octeon and one can use Linux as a generic version. A big disadvantage of the Octeon-driver is its disruptive behavior. Each time it processes a packet, it receives an interrupt, changing the atomic tag, does its processing and exits. Changing the atomic tag is a killer as the Sirius stack uses atomic tags for locking its bookkeeping. So using the Octeon-driver together with Sirius is not an option if you want to run the application on more than one core. If the Octeon-driver is loaded, you should unload it before using any other application using atomic tags (using *modprobe –r octeon-driver*), or you should use the Sirius tap driver. This driver enables the network function of Linux when running a Sirius application without changing the atomic tag. You can find more details about the disruptive behavior of the Octeon-driver in the SDK documentation: 'Linux Userspace on the OCTEON', chapter 8: Checklist for Simple Executive Programming.

Modifying the crypto-api

Given the above warning and because we do not want to make it too complicated, we assume at this point you have a way of getting your application onto Octeon Linux, either via the Octeon-driver, a flash card, the PCI-express bus, or via the Sirius tap driver.

Some modifications must be made to the Octeon crypto-api before it can be used with Sirius and wolfSSL. Enabling the *OCTEON_OPENSSL_NO_DYNAMIC_MEMORY* flag in the *crypto.mk* file in the *$OCTEON_ROOT/components/crypto-api/core* directory is the first modification.

Secondly, the memory manager included with the crypto-api was not designed with multi core in mind. By default, it will only allow to free allocated memory on the same core as it was allocated. When going true multi core, it must be possible to free memory independently of the core it was allocated on, because processing of the RSA key exchange might end on a different core than the one it was started on. The crypto-api memory manager is located in mem.c in the $OCTEON_ROOT/components/crypto-api/core/crypto directory. Download the patch from the download section from our website www.sirius-networks.com and put it in the directory where *mem.c* resides.
Apply the patch:

#patch –p2 < sirius_crypto.patch

The amount of memory allocated by the memory manager is defined in memconfig.h. By default, the amount of memory per core is defined here. In our application, only one core will initialize the amount of memory, so look into this file and check if the allocated memory is sufficient for your application. Increase it when necessary. You might want to enable the OCTEON_OPENSSL_SW_FPA_POOL_STATS flag and call the *sw_fpa_stats()* function in *mem.c to* check the available memory.

Building the Echo server
As mentioned in the prerequisites in part 1, your SDK needs to be capable of handling C++ files. Get the tutorial named *Using C++ on Cavium Octeon* on how to change your SDK from the white paper section on our website: www.sirius-networks.com.

Download the echo server named *Sirius SSL Echo Example* from our download section. Unzip it to an appropriate directory

#cd /root
#unzip sirius_ssl_example.zip

After unzipping, the directory structure that was built lacks two main things. First, there is a directory *wolfssl*, but it is empty, except for the files *malloc_cavium.c* and *wolfssl.mk*. Copy the needed directories and files that were created in part 2 of this tutorial to the wolfSSL directory

#cd /root/Sirius_SSL_Example
#cp –r /root/wolfSSL/wolfssl.x.x.x/wolfssl wolfssl/.
#cp –r /root/wolfSSL/wolfssl.x.x.x/wolfcrypt wolfssl/.
#cp –r /root/wolfSSL/wolfssl.x.x.x/src wolfssl/.
#cp –r /root/wolfSSL/wolfssl.x.x.x/certs wolfssl/.
#cp –R /root/wolfSSL/wolfssl.x.x.x/config.h wolfssl/.

Secondly, you need a copy of Sirius' source code or a compiled version of it. The latter will be a library (libsirius.a) and has to be put into the *Sirius/lib-linux_64* directory. Contact Phact BV (www.phact.nl or www.sirius-networks.com) for a copy of the Sirius library.

When the wolfSSL source code and the Sirius library are in place and the SIRIUS define is enabled in the *settings.h* file , build the SSL echo server:

#make OCTEON_TARGET=linux_64 clean
#make OCTEON_TARGET=linux_64

After transferring the executable (sirius_ssl_echo-linux_64_strip) and the wolfssl/certs directory to Octeon Linux, you can start the application on Octeon on multiple cores

#schedtool –a 0xf –e ./sirius_ssl_echo-linux_64_strip &

The output should be something like this

```
CVMX_SHARED: 0x120460000-0x120480000
Active coremask = 0xf
Version: Cavium Inc. OCTEON SDK version 2.3.0-p10, build 518
Ethernet driver detected
Waiting for ethernet module to complete initialization...
Link status port 0. Duplex = 1, speed = 1000, status = 1
Resource pool 0 1 2 3 4: 16285 24322 1000 1000
Resources in use by socket: 1, 0, 1: 2
Resources in use by IP layer:                   0
Resources in use by CMemList:                   0
Resources in use by CMemListAux:                0
Resources in use by CSynCache:                  0
Resources in use by CSiriusTimer:               1
Core load: 00 00 00 00
Resource pool 0 1 2 3 4: 16277 24330 1000 1000
Resources in use by socket: 1, 0, 1: 2
Resources in use by IP layer:                   0
Resources in use by CMemList:                   0
Resources in use by CMemListAux:                0
Resources in use by CSynCache:                  0
Resources in use by CSiriusTimer:               1
Core load: 00 00 00 00
```

You can verify the echo server by using the small python script listed below. It will set up an SSL connection to the echo server and send a message. The echo server will return this message.

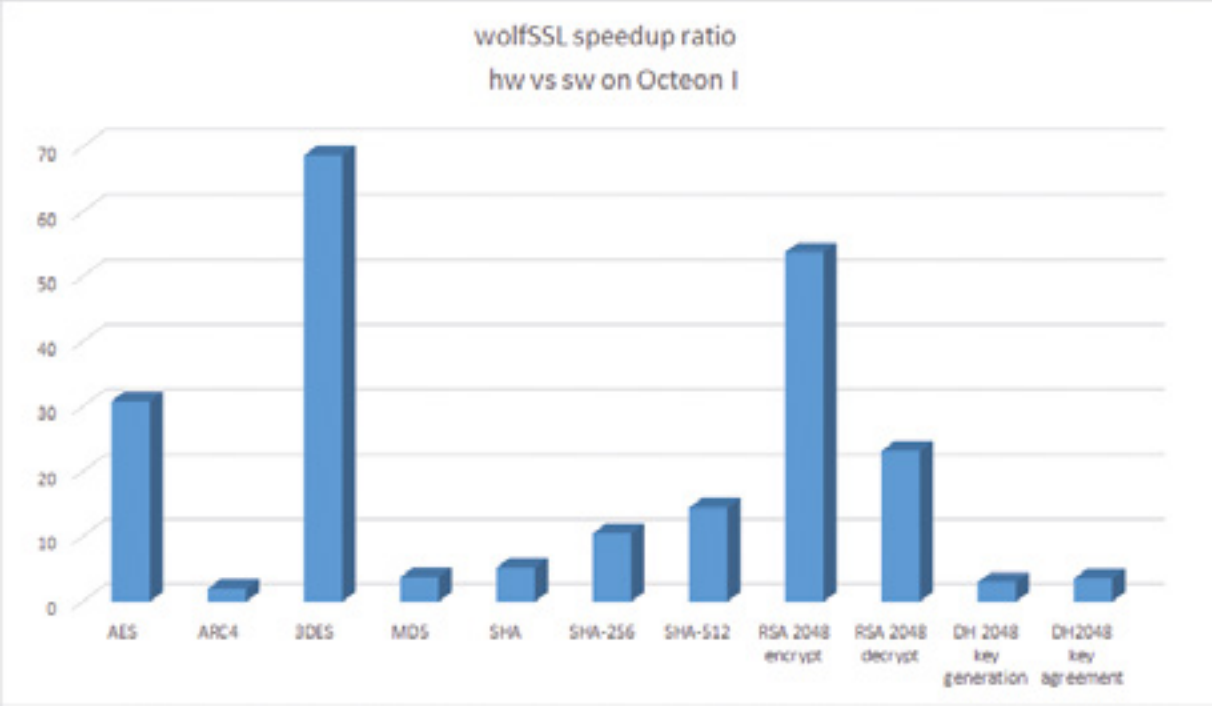
```
from socket import socket
import ssl

s = socket()
c = ssl.wrap_socket(s)
c.connect(('10.101.41.170', 443))
c.write('This is wolfSSL on Octeon, can you hear me?\n')
data = c.read()
print data
c.close()
```

The maximum number of simultaneous connections is determined by the version of Sirius. If you have a demo version, the maximum number of SSL/TLS connections will be limited to one. If you have a full featured version, the maximum number of simultaneously handled connections will be limited by the amount of available memory and can be up to tens of thousands!

In conclusion

In this tutorial we have modified the wolfSSL source code and the Octeon crypto-api to create a multi core hardware accelerated SSL/TLS echo server. This example can be build with SDK2.x and SDK3.x and works on Octeon, Octeon-II and Octeon-III processors.

In the diagram below the acceleration of hardware versus software is shown. This diagram shows that when the hardware encryption engine is incorporated, speed increments of up to 68 times can be achieved. Performance can be boosted further if multiple cores are used to run the echo server. Due to the use of atomic tagging the performance will increase in a linear fashion with the number of cores used. Doubling the number of cores will double the performance.



The following table contains absolute performance results as created by the wolfSSL benchmark utility. Multiply these numbers by the number of cores if a multi-core system is used.

	OCTEON+ @600MHz	OCTEON-II @1100MHz	OCTEON-III @1600MHz
AES [MB/s]	268,2	468,2	688,0
ARC4 [MB/s]	70,6	117,7	156,0
3DES [MB/s]	171,7	167,7	205,6
MD5 [MB/s]	202,4	295,0	319,6
SHA [MB/s]	277,2	577,2	499,1
SHA-256 [MB/s]	202,0	402,1	395,9
SHA-512 [MB/s]	292,6	577,3	499,3
RSA 2048 en [it/s]*	3225,0	4830,0	6993,0
RSA 2028 de [it/s]	139,0	366,7	680,7
DH 2048 gen [it/s]	53,0	135,4	238,1
DH 2048 agr [it/s]	53,0	135,2	237,6

Powered by Phact

Sirius is a TCP/UDP/IP protocol stack and is Phact's solution of daily handling millions of TCP and SSL connections on OCTEON hardware. Sirius is available as source code or as a library build for your OCTEON platform. It runs as Simple Exec or on OCTEON-Linux.

Phact is a one stop shop for software and hardware solutions, support and consultancy and was founded in 2015 with 25 years of experience in the business.

How? Phact originated in VSN Systemen. VSN was founded in 1991. The last couple of years, the company grew and evolved tremendously. It was time to adjust the look and feel of the company to what it had become. That's why Phact was born.

We started developing telecom hardware and software for VoIP- and SS7-carriers in 1991. For them, our engineers developed stable and professional hardware and software solutions. Additionally, we provided our customers with support and management of their solutions.

Knowledge is a key part of our business. Therefore, we are engaged in close partnerships with universities and our professional suppliers.

Phact might still be very young, but we carry years of experience in software and hardware development, support, management and consulting.

Phact BV
Keizersveld 83, 5803AP, Venray, The Netherlands
T +31 (0) 478555000 F +31 (0) 478589563
Email: info@phact.nl
www.phact.nl
www.sirius-networks.com