



SunSSH with HW crypto support

Jan Pechanec
2008-09-15



The plan

- **SunSSH with HW crypto support**
 - > [6445288](#) ssh needs to be OpenSSL engine aware
 - the main CR for this project
 - > hard dependency on
 - [6685012](#) OpenSSL pkcs#11 engine needs support for new cipher modes
 - [6725903](#) OpenSSL PKCS#11 engine shouldn't use soft token for symmetric ciphers and digests
 - > and also dependent on various other fixes for bugs found during the project

Disclaimer

- this presentation is about a finished project in Nevada, any information on possible backport to Solaris 10 is subject to change at any time
- I assume you know what the Crypto Framework (CF) is and what is its role in Solaris
 - > see References for more info
- **example numbers != benchmarks**

SunSSH with HW crypto support

- problem: „**SSH is slow on Niagara 2**“
- because SunSSH is single threaded
 - > as well as other SSH implementations
- and T2 CPUs are slow comparing to CPUs used in „typical“ 1-4 CPU machines
 - > we could not saturate 100Mbit ethernet with T5220 and SunSSH using default AES mode
 - > the transfer speed was just ~9MB/s (~75Mbit/s)
- most time spent in crypto operations
 - > **really?** Yes - we will see in a moment.

Some numbers on SSH data transfer

- 500MB data transfer through SunSSH without any HW crypto acceleration
- **~9MB/sec** on UltraSPARC T2 (~75Mb/sec)

```
$ time dd if=/dev/zero bs=1024k count=500 | ssh t5220-sfb-06 'cat >/dev/null'
```

```
real    0m53.581s
```

```
user    0m11.951s
```

```
sys     0m6.153s
```

- **~25MB/sec** with two AMD64 machines, 2.4GHz CPU (~200Mb/sec)

```
real    0m20.101s
```

```
user    0m16.359s
```

```
sys     0m2.115s
```

Encryption speed is crucial !

- task: encrypt 500MB of data in memory

```
time dd if=/dev/zero bs=1024k count=500 | openssl enc -aes128 -k xxx | cat >/dev/null
```

> and compare to the previous slide

- on UltraSPARC T2 (~**11MB/s**)

```
real    0m45.743s           (compare to 53.581s)
user    0m42.745s
sys     0m5.836s
```

- on AMD64, 2393 MHz (~**35MB/s**)

```
real    0m13.819s           (compare to 20.101s)
user    0m11.882s
sys     0m1.701s
```

How SSH packet is processed

- **encryption is not the only operation**
 - > HMAC is computed over the whole data
 - see RFC 4253 for more info if interested
 - > and HMAC is about message digests

```
time dd if=/dev/zero bs=1024k count=500 | openssl md5
```

```
real    0m2.830s
```

```
user    0m1.927s      on AMD64, 2.4GHz
```

```
sys     0m0.786s
```

- AES + MD5: 13.82s + 2.83s = **16.65s**
- SSH transfer: **20.1s** **~85% was crypto !**

How to speed up the encryption?

- SW parallelism
 - > **encrypt in parallel using threads**
 - > from cipher modes we support, only AES-CTR can be used here
 - > that's the default mechanism used in SunSSH (note - OpenSSH uses AES-CBC)
 - > this way is suitable for all Niagara machines
- HW solution through the Crypto Framework
 - > **offload encryption to n2cp(7d) or mca(7d)**
 - > suitable for machines with HW crypto providers
 - i.e., not for Niagara 1 without SCA-6000

The way we decided to go for now...

- **...is through the hardware solution**
- not suitable for Niagara 1 (T1)
 - > which can only accelerate RSA/DSA/DH through *modexp* operation offered by ncp(7d)
- SCA-6000 can always help on T1
 - > mca(7d) is the driver name
- n2cp(7d) on Niagara 2 is our primary target
- combining with the software solution could be a future project (no ETA for that)

Encryption versus Message Digests

- SHA1, MD5 fast in software
 - > MD5 more than 6x faster than AES on AMD64
 - > what's more, small data packets (ACKs-like) are processed slower in HW than SW by an order of magnitude
 - > **no speed gained in SSH by offloading digest operations to the hardware**
- AES, DES quite slow in software
 - > significant speed gained by offloading to HW
- RSA/DSA will be offloaded as well but SSH is not used the way as HTTPS is

Catching up with AMD64 on speed

- ...using the Crypto Framework on T2
- again, encrypt 500MB of data in memory
 - > now, we use “**-engine pkcs11**” on T2
- on AMD64, 2393 MHz

```
real    0m13.819s
user    0m11.882s
sys     0m1.701s
```

- T2 + PKCS#11 engine (**36MB/s**)

```
real    0m13.834s
user    0m6.167s
sys     0m10.726s
```

BTW, what is the “engine” ?

- when you hear “the engine” in connection w/ the CF, it is **the PKCS#11 engine**
- an engine is an OpenSSL module
 - > it's a generic container for implementation of cryptographic operations
- the PKCS#11 engine offloads all operations to the CF using the PKCS#11 API
 - > our PKCS#11 engine serves as a liason between OpenSSL and the Cryptographic Framework
 - > CF knows nothing about OpenSSL or an engine
 - > ...and OpenSSL knows nothing about the CF

Back to performance numbers - could we get even faster ?

- HW crypto limits are quite impressive
 - > for AES-128, roughly 700MB/sec per core
 - > we reached 36MB/s with the engine using
“openssl enc ... -engine pkcs11”
- however:
 - > Crypto Framework has overhead
 - > the PKCS#11 engine has overhead
 - > inherent overhead to start HW operation
 - > packets larger than ~8KB are not usually used
 - we do not offload 128KB or so

Some numbers w/ accelerated SSH

- transfer time reduced to **40%** on T2
 - > that is **2.5x** speed-up compared to the SW only case
 - > in default AES-CTR mode
- previous situation on T2 (**~75Mb/sec**)

```
$ time dd if=/dev/zero bs=1024k count=500 | ssh t5220-sfb-06 'cat >/dev/null'  
real    0m53.581s
```

- **current situation using the PKCS#11 engine on T2 in snv_99 (~190Mb/sec)**

```
$ time dd if=/dev/zero bs=1024k count=500 | ssh t5220-sfb-06 'cat >/dev/null'  
real    0m20.839s
```

And more numbers with Niagara 2

- workaround could be using RC4 without any HW acceleration
 - > we get to **~115Mb/sec** (500MB in 34 secs)
 - > however, accelerated RC4 is then only ~2% faster than accelerated AES case
- the worst SW case is using 3DES since single DES is quite slow in SW
 - > we get only **~24Mb/sec** (500MB in 160 secs)
 - HW accelerated case is **~180Mb/sec**
- basic rule - **accelerated cases are of roughly the same speed**

And yet another numbers...

- SCA-6000 is slower than n2cp(7d) w.r.t crypto operations
 - > but is still worth to have it
- experiments show that using mca(7d) with SunSSH prototype on T1 reduces the transfer time to **~60%** when using the default AES CTR mode (= 1.6x speed-up)
 - > e.g., **10** seconds instead of **16** seconds
 - > however, it might be easier to use RC4 cipher which gains similar speed without any HW acceleration (SCA-6000 doesn't support RC4)

Conclusion on example results

- **current results seem quite good for the 1st phase**
 - > we'll try to continue to search for bottlenecks
- with **~190Mb/s** on T2, it looks like using threads might get us to 1Gb link saturation
 - > BTW, T2 has 8 cores, each with a crypto chip
- even parallelization without any HW acceleration might get interesting results
 - > imagine 128 CPUs
 - > however, w/o a prototype **we just speculate**

OpenSSL API or PKCS#11 API?

- SunSSH already uses OpenSSL API
 - > more precisely, it uses generic high-level EVP (= envelope) API
 - > for symmetric ciphers and digests, you must use EVP if you want to use any engine
 - > we chose to use the PKCS#11 engine thru the OpenSSL EVP API to expedite the project
- we may switch to the PKCS#11 API
 - > **when Crypto Framework is FIPS-140-2 certified SunSSH would greatly benefit**
 - > + getting rid of EVP and engine overheads

Switching to the PKCS#11 API ?

- we couldn't now - softtoken is not ready yet
- OpenSSL x softtoken on AMD64, in 32-bits:

type	16 bytes	64 bytes	256 bytes	1024 bytes	8192 bytes
aes-128-cbc	39943.87k	44713.91k	45839.31k	46808.62k	46812.73k
aes-128-cbc	21220.06k	33927.97k	40124.46k	41483.31k	42540.83k

- we see **~9%** regression in 8KB case
- why we care? Softtoken would be used on all machines w/o HW crypto accelerators
 - > remember, the CF knows nothing about the existence of OpenSSL, and can not use it

Softtoken & Crypto Framework

- it wouldn't be fair to blame just softtoken
- between the application and the softtoken, there is the Crypto Framework
- that means one more layer in contrast to using the OpenSSL native code
 - > and not speaking about the PKCS#11 engine
- **which means a logical overhead of CF**
- we would need the softtoken to be slightly better than OpenSSL code
 - > which is not very easy to achieve at all

Switching to the PKCS#11 API (II.)

- we don't care *that* much about small blocks
 - > small data blocks in SSH mean an interactive session
 - every stroke is transferred separately
 - > you can't type fast enough to get affected
- however, **any possible regression in 8KB packets is a serious problem**
 - > SunSSH prototype using the softtoken showed **~10-15%** regression on my AMD64 machine
 - > and **~25%** regression on T2

Switching to the PKCS#11 API (III.)

- on Sun-Fire-V890, we even see **~5%** speed-up on 8KB packets using OpenSSL speed command in 32-bits
 - > of course, processing smaller packets is still slower
- **but**, if we switch to the PKCS#11 API:
 - > **we must not see any regression on any platform at the day of the integration**
 - that probably also includes “using any cipher”
 - > after that, any possibly faster future OpenSSL version will not be relevant any more w.r.t. the switch to the PKCS#11 API beforehand

More on the PKCS#11 engine RFEs

- with 6725903* we check the HW crypto capabilities of the box on engine init
 - > staying in OpenSSL code if the softtoken is the only token available, thus avoiding regression
- we added new cipher modes to the engine
 - > so we can use 192/256 bit keys
 - > AES CTR mode support added
 - however, an app needs its own EVP functions since OpenSSL doesn't support AES CTR through EVP
 - we support 128 bit long counter only; that's what SSH protocol needs (RFC 4344)

* OpenSSL PKCS#11 engine shouldn't use soft token for symmetric ciphers and digests

What we found along the way

- not counting bugs found in SunSSH or the PKCS#11 engine
- the list of bugs found in other categories:
 - > [6674088](#) userland threshold for hw offloading makes it difficult for SSL and SSH protocols
 - > [6681527](#) meta_SetOperationState() doesn't return a slot session to the idle pool
 - > [6693530](#) n2cp lacks support for >64 bits long AES counter
 - > [6693598](#) mca lacks support for >64 bits long AES counter
 - > [6705398](#) kernel_get_operationstate() returns CKR_FUNCTION_NOT_SUPPORTED when no operation is active
 - > [6685034](#) soft token DH implementation is 7x slower than OpenSSL on Niagara1
 - > [6728450](#) 6708125 prevents parent to use the Crypto Framework after the fork(2)
 - > [6730022](#) failed SSL connections when Crypto Framework userlevel treshold is set to 0

Implementation notes

- most changes needed because of how the PKCS#11 standard deals with `fork(2)`
- the privileged ssh daemon forked an unprivileged child **after authentication**
 - > and the child continued to secure the transport layer using inherited crypto context
 - > that is fine with OpenSSL but **the PKCS#11 standard explicitly forbids that**
- several possible solutions exist to overcome this

Pre-fork as a solution

- pre-fork solution was chosen
 - > **now, the child is forked before the crypto contexts are established**, and drops privileges after the authentication is finished
- however, the privileged daemon (AKA the *monitor*) then lacks information gained before through the finished authentication
 - > so, among other changes, the child/monitor protocol must be extended to provide monitor with the auth info (e.g., username)
 - > see *New Features in OpenSolaris and Beyond* for more information about possible solutions

Where pre-fork doesn't fit

- *go-to-the-background-after-authentication* option (-f) of ssh(1) client also posed the same `fork(2)` problem
- however, if we pre-forked here we had 2 ssh clients communicating to each other
 - > could be quite confusing to some users
 - > and using -f option is rare
 - useful with port forwarding, for example
- `~&` escape sequence has the same problem, it also backgrounds the client

Where pre-fork doesn't fit (II.)

- solution: from the client, **we request key re-exchange (REKEX) before daemonizing** and initialize the new crypto contexts with new keys after that
- not suitable for SSH Protocol version 1 since there is no REKEX there
 - > who cares? Protocol version 1 is obsolete and should not be used.
- REKEX is quite expensive but this situation will not happen often

Complexity of the change

- SunSSH has ~65K lines of code
- webrev says we changed ~1000 lines
 - > some parts of the code were just moved around
 - > changes were usually small but frequent
- changes mostly in critical parts
 - > privilege separation
 - > key exchange
- this code can not be applied to OpenSSH
 - > see its `-with-ssl-engine` configure option (I do not know if it works with the PKCS#11 engine)

Backport to Solaris 10?

- project integrated to snv_99 (Sep 2008)
- there is a plan for S10 backport
 - > but we can not promise it will actually happen, or when
 - > and we can not promise any patches for S10
- **please do not ask me, use your proper channel if you have a support**
- the PKCS#11 engine changes must be backported with it
 - > plus quite a few other changes which makes it more difficult

References

- SunSSH page on OpenSolaris.Org
 - > <http://www.opensolaris.org/os/community/security/projects/SSH>
- Pechanec/Schuba/Phalan: *New Features in OpenSolaris and Beyond*
 - > paper presented at OSDevCon 2008 in Prague, June 2008
 - > <http://www.osdevcon.org/2008/files/osdevcon2008-proceedings.pdf>
- UltraSPARC T2 Processor System On a Chip
 - > a presentation on UltraSPARC T2, including its crypto parts, August 2007
- manual pages
 - > openssl(5), engine(3), EVP(3), speed(1)
- Cryptographic Framework on OpenSolaris.Org
 - > <http://opensolaris.org/os/project/crypto>
- Wolfgang Ley: *OpenSolaris Cryptographic Framework*
 - > paper presented in at OSDevCon 2008 in Prague, June 2008
 - > <http://www.osdevcon.org/2008/files/osdevcon2008-wolfgang.pdf>
- RSA Laboratories - PKCS #11: Cryptographic Token Interface Standard
 - > no static link, use Google to look it up

References (II.)

- opening slide with Machu Picchu photo and other photos from Peru
 - > http://www.bagr.cz/hanz/2005_peru/
- my blog at blogs.sun.com
 - > mostly on the PKCS#11 engine and SunSSH
 - > <http://blogs.sun.com/janp>



Questions?

<http://blogs.sun.com/janp>

jan.pechanec@sun.com