

Onion Router Performance on a Niagara 2 System

Wyllys Ingersoll

Sun Microsystems, Inc

September 2009.

1 Background

The purpose of this project was to analyze the performance of an application using AES encryption on a Sun Niagara 2 platform. In particular, we focused on increasing the performance of the AES encryption in the Onion Router software (Tor). Tor was chosen because it makes heavy use of AES for relaying data between onion router nodes and was easily modified to use the encryption interfaces available on the OpenSolaris platform.

We examined the performance of the cryptographic processing of the onion router modified to use the Niagara 2 integrated cryptographic processor and compared it to the performance without the hardware acceleration option. The results are presented below.

2 Analysis

The onion router software currently can be configured at compile-time to either use the AES ECB algorithms from the OpenSSL libraries or to use its own private implementation of the Rijndael (AES) algorithms. Neither of these implementations include the stream cipher mode (CTR) needed by Tor. Instead they use the ECB mode and then perform the necessary XOR feedback of the counter blocks in software by breaking down the overall block of data to be relayed into 16 byte chunks (128 bits) and updating the counters each time. By utilizing the native AES CTR mode provided by the Niagara 2 processor, the performance penalties incurred by having to break each block into 16 byte chunks and manually update the counters are removed. Ultimately, the hardware acceleration of the AES cipher was shown to improve the performance of the Tor data encryption and relay functions by a factor of 25 or more (see below).

For analyzing the performance, a dtrace script was developed to analyze the time spent in various functions that Tor uses in critical places. The script analyzed the time spent in several key functions that tor uses to encrypt and decrypt and pass the data packets along. Specifically the following functions were measured:

- tor_tls_renegotiate
- tor_tls_handshake
- tor_tls_write
- tor_tls_read
- aes_set_key
- connection_handle_write

- connection_handle_read
- aes_crypt_inplace

The functions most impacted by the hardware accelerated crypto were those involved in encrypting and decrypting the data being passed between the relay nodes. The TLS functions do not use AES so their performance is unaffected by the hardware acceleration on the Niagara2 systems. The real key function to watch is *aes_crypt_inplace* since that is where the packet data being relayed is encrypted and decrypted as it passes through the node.

3 Results

FUNCTION	Avg time (usecs) HW= OFF	Avg time (usecs) HW = ON	Change (hw off vs hw on)
tor_tls_read	422594	458793	-8.00%
tor_tls_write	296049	355387	-17.00%
tor_tls_handshake	35270375	34031792	3.00%
tor_tls_renegotiate	65602977	64841835	1.00%
connection_handle_write	435831	555183	-21.50%
connection_handle_read	6124603	1285696	476.00%
aes_set_key	158262	85777	185.00%
aes_crypt_inplace	2912921	100912	2886.00%

The results were gathered using a DTrace script on an internet facing Sun T5220 system running the Solaris Nevada operating system (see Appendices for details).

4 Conclusion

The stats in the results table above indicate minor degradation of the TLS read/write operations with HW mode enabled. When the Tor “HardwareAccel” option is enabled, the OpenSSL engine used is the Solaris pkcs11 engine, which introduces some additional overhead. The pkcs11 engine used does not support SSL or TLS operations in hardware so those operations do not benefit from any additional acceleration and were not expected to show improvement.

However, the benefit seen in the AES operations, specifically the *aes_crypt_inplace* function, greatly offsets the minor penalty in the TLS area and ultimately allows the tor node to relay packets much faster with the Niagara 2 encryption enabled than without.

5 References

- <http://www.opensolaris.org/> - Download OpenSolaris, or browse the source code.

- <http://src.opensolaris.org/source/xref/sfw/usr/src/cmd/tor> – the source for the tor software used in OpenSolaris, including the Patches applied to enable the HW crypto used in the testing described above.
- <http://www.sun.com/servers/coolthreads/t5220/specs.xml> – specifications for the Sun SPARC Enterprise T5220 server used in the testing described above.
- <http://www.torproject.org> – Home of the Onion Router software including documentation and download links for the original source and detailed information about the protocols used.

6 Appendix A – Configuration Details

Below are the configuration details for the system that was involved in the testing.

Hardware Platform	Sun SPARC Enterprise T5220	8 core 1.6GHz UltraSPARC T2 processors running max 64 threads.
RAM	32 GB	
Operating System	Solaris Community Express build 110	Roughly equivalent to OpenSolaris 2009.06
Software	Tor version 2.0.34	http://www.torproject.org

7 Appendix B – Dtrace script for Analyzing Tor AES crypto

Below is the Dtrace script used to analyze the performance.

```
#!/usr/sbin/dtrace -s
#pragma D option quiet

/*
 * User must have dtrace_proc and dtrace_user privileges (or run as root).
 * Usage:
 *   ./torstat.d PID
 *   where PID = the process id of the original tor relay daemon.
 * Allow the script to monitor the process for a period of time (10 minutes
 * or more is sufficient) and then terminate with ^C and the final stats
 * will be displayed.
 */

dtrace:::BEGIN
{
    start = timestamp;
    printf("Tracing... Hit Ctrl-C to end.\n");
    printf("Looking for pid %d\n", $1);
}

pid$1:tor:tor_tls_read:entry,
pid$1:tor:tor_tls_write:entry,
```

```

pid$1:tor:tor_tls_handshake:entry,
pid$1:tor:tor_tls_renegotiate:entry,
pid$1:tor:tor_tls_shutdown:entry,
pid$1:tor:connection_handle_read:entry,
pid$1:tor:connection_handle_write:entry,
pid$1:tor:aes_set_key:entry,
pid$1:tor:aes_crypt_inplace:entry,
pid$1:libcrypto:EVP_EncryptUpdate:entry
{
    self->starttime[probefunc] = timestamp;
}
pid$1:tor:tor_tls_read:return,
pid$1:tor:tor_tls_write:return,
pid$1:tor:tor_tls_handshake:return,
pid$1:tor:tor_tls_renegotiate:return,
pid$1:tor:tor_tls_shutdown:return,
pid$1:tor:connection_handle_read:return,
pid$1:tor:connection_handle_write:return,
pid$1:tor:aes_set_key:return,
pid$1:tor:aes_crypt_inplace:return,
pid$1:libcrypto:EVP_EncryptUpdate:return
/self->starttime[probefunc]/
{
    @avg[probefunc] = avg(timestamp - self->starttime[probefunc]);
    @num[probefunc] = count();
}

dtrace:::END
{
    elapsed = (timestamp - start)/1000000000;
    printf("\nTotal Elapsed time: %d seconds\n\n", elapsed);
    printf("%-32s Count      \tAvg(usecs)\n", "FUNCTION");
    printa("%-32s %@-12d\t%@d\n", @num, @avg);
}

```