Measured interference of network security mechanisms with network performance

Hans-Werner Braun <hwb@sdsc.edu>
Kimberly Claffy <kc@sdsc.edu>
Andrew Gross <grossa@sdsc.edu>

Abstract

Since starting to use Kerberos [1], which provides the option to encrypt rlogin sessions at the application layer, we noticed that using encryption caused a severe performance impact across dial-up lines, e.g., a 14.4kbps line using PPP with modem compression of data.

We took measurements and verified the source of the problem as a non-synergistic interaction between the application, transport, and lower network layers. We modified the Kerberized rlogin *(*klogin*) to accommodate this situation, and will make the modifications publicly available. Since normal ascii text data is quite compressible, this modification could become important for ubiquitous low speed access.*

1 statement of problem

Since starting to use Kerberos,[1] which provides the option to encrypt rlogin sessions at the application layer, we noticed that using encryption caused a severe performance impact for interactive applications across dial-up lines, e.g., even 14.4kbps line using PPP.

To investigate the impact of the move to more Internet-wide use of end-to-end encryption, we took measurements to verify the source of the problem. We found a clearly non-synergistic interaction between the application, transport, and lower network layers. Although the issue became visible to us with Kerberos, it is a generic problem of performing compression and encryption in the right order. Since encryption is inherently a randomization process, it is clear that trying to compress after performing encryption will yield suboptimal results. However, many applications perform encryption at higher layers without first performing compression, leaving lower layers (e.g., modems) unable to effectively compress data before transmission. We modified the Kerberized *rlogin* (*klogin*) to accommodate the situation, and will make the modifications publicly available [2][2].

2 background

With most dial-up lines, often not faster than 14.4kbps, variations in interactive performance are extremely visible to users. Many optimizations can improve performance over slow links; examples include Jacobson's header compression [3] and the V.42bis compression [4] used in many modems. Modem compression can raise the effective end-to-end throughput of a 14.4kbps line to 20-30kbps. For example, after the *rlogin* application sends packets to the network layer, the host sends the packets out via a modem that compresses the data as it transmits it. The benefit of compression is particularly strong for data that exhibits considerable redundancy, e.g., ascii text data. Encrypting at the application layer, e.g., as performed in Kerberos services, makes it virtually impossible for the modem to compress the data before transmitting it. This essentially limits the system to the raw 14.4kbps that the modem can offer. To confirm this hypothesis we verified that disabling compression on the modem yielded similarly sluggish performance for encrypted as well as unencrypted rlogin streams.

Note that one would likely not notice a performance difference on machines connected via an Ethernet or higher speed medium. Since both the end hosts and the network can perform fast enough without compression, even with competing traffic on the LAN, the optimization function of a modem is unnecessary. This effect also will not manifest itself for very small packets, e.g., the one or very few character packets sent from the client to the server side of a telnet session. Such packets are not ideally amenable to compression since most compression algorithms rely on repeated patterns in the data. The effect of the suboptimal ordering on the end user is most acute with interactive sessions, requiring large packets in at least one direction, over low speed lines.

[1] Both versions, Kerberos 4 and 5, exhibit the same behavior.

[2] We are currently cleaning up the code modifications and will make it available by the conference date.

3 performance measurements

In order to determine the extent of the throughput difference between encrypted and unencrypted sessions, we measured the throughput performance of a file transfer (*ftp*), in both directions across a 14.4kbps serial link line, using three textfiles that were equal in length (100,000 characters) but differed in the degree to which they were conducive to compression. We then measured the latency to transfer each file in encrypted and nonencrypted form, across a 14.4kbps dialup PPP line, using modem compression.

The first text file consisted of a series of a repeated character. The second file consisted of the result of a `ls -R` command, presumably somewhat analogous to English text. The third file consisted of a a series of characters selected randomly[3] We created two encrypted versions of each file, one with the unix *crypt* utility, the other with a unix *des* program. We then sent the nine files across a dialup link to a remote system three times in each direction. There was little difference among the throughput performance of the three iterations. Table 1 shows the maximum values of the measured throughput performance of the file transfers. The table verifies our hypothesis; performance drops in half on random text (e.g., the result of encryption) if one is bandwidth-limited rather than CPU-limited. The interference among multiple layers of the network is critical to system performance.

For comparison, we measured the file sizes (in bytes) that the unix *gzip* and *compress* utilities were able to achieve on the nine 100,000 bytes files. Table 2 shows the results; *n/a* indicates that *compress* did not yield a file smaller than the original file.

4 remedy

We now dicuss methods to obtain higher performance in slow speed environments. Note that disabling the modem compression renders comparable the throughput of encrypted vs nonencrypted sessions. The existing order of operations is suboptimal. One solution would be to compress prior to encryption, in *klogin* itself. So rather than encrypting the data within *klogin*, and then letting lower layers compress, we compress the data first, and then encrypt:

Current functionality:

application generates data
application encrypts data
host sends data
(modem) compress
send across network
(modem) decompress
decrypt
consume data

To obtain a higher throughput the application should compress bulk information before encrypting it:

application generates data
application compresses data
application encrypts data
host sends data
(modem) compress (minimally)
send across network
(modem) decompress
decrypt
consume data

As noted above, some cases will not be amenable to performance improvements, such as applications that communicate using only very short packets that are not compressible. We focus instead on areas where improvement is possible, that is, where we have large enough blocks of data that compression will provide a net benefit.

5 implementation details

For slow speed links, the application designers should be aware of the underlying behavior of the network. A designer may generate code that uses many more packets than are necessary to accomplish a given task. This approach will result in an increase in latency that will be apparent in interactive sessions.

Kerberos handles encrypted data by transmitting two separate packets for each transmission of an otherwise unencrypted packet: one packet holds the length of encrypted data, and the second packet holds the data itself. In *klogin* this behavior results in the generation of 7 packets for a single keystroke, caused by the application doing two *write*s rather than a *writev* to merge the data into a single buffer. By using *writev* on both sides, we reduce 7 packets to 3 and save network bandwidth as well as latency. Figure 3 shows packet traces for the transmission of a single character stroke, using both the original *klogin* and our modified version of *klogin*.

We also added a byte at the beginning of the encryped data to indicate whether the data is compressed. This mechanism allows us to forgo compression for cases where it does not provide a benefit.

[3] We used the perl *rand()* function, which returns a random fractional number between 0 and a specified positive value.

Table 1: throughput performance (in kBytes/sec) for unencrypted, *crypt* encrypted, and *des* encrypted files from home machine to sdsc machine across 14.4kbps line (using PPP)

	send			recv		
	plain	*crypt*	des	plain	*crypt*	des
same char (u)	3.63	1.59	1.58	3.52	1.54	1.53
text (lsR)	3.59	1.58	1.58	3.52	1.52	1.53
random char (r)	1.59	1.57	1.59	1.54	1.53	1.52

Table 2: compression performance using *gzip* (.gz) and *compress* (.Z) on the 100,000-byte files described (unencrypted, *crypt* encrypted, and *des* encrypted)

	plain		*crypt*		des	
	.gz	.Z	.gz	.Z	.gz	.Z
same char (u)	138	530	100049	n/a	100056	n/a
text (lsR)	26632	37548	100051	n/a	100058	n/a
random char (r)	100044	n/a	100049	n/a	100056	n/a

6 other examples

Adding any extra functionality to network applications will always detract somewhat from performance. Security is no exception. For example, the Andrew File System (AFS) implements a great deal of security functionality that imposes a performance cost. Similarly, routers and application-level firewalls impact performance when filtering.

We have highlighted only one example of situations where security measures are added long after initial system design, at whatever layer is convenient or easiest at the time. Other authentication and privacy schemes will have similar problems. There is the danger that users will disregard security because it is not worth the performance degradation. Unfortunately, modifying existing applications and architectures so as to avoid the performance degradation is generally more effort than users are willing to expend. Application designers must be aware of this risk as they design implementations.

In summary, it is no surprise that encryption precludes the ability to perform subsequent compression. It is therefore worth examining its implication for the recent popularity of adding network security mechanisms to extant applications. Although it is clear that compression should occur before encryption, implementing it properly above the transport layer is harder than kludging it into lower levels, albeit at the expense of performance. We have offered this study to demonstrate that many cases do not require the sacrifice in performance, if only application builders are willing to consider security issues as integral to design rather than an issue to be dealt with later.

author information

Hans-Werner Braun is a Principal Scientist at the San Diego Supercomputer Center. Current research interests include network performance and traffic characterization, and working with NSF on NREN engineering issues. He also participates in activities fostering the evolution of the national and international networking agenda. San Diego Supercomputer Center, P.O. Box 85608, San Diego, CA 92186-9784; *hwb@sdsc.edu*.

Kimberly Claffy received her doctoral degree from the Department of Computer Science and Engineering at the University of California, San Diego in June, 1994, and is currently a Associate Staff Scientist at the San Diego Supercomputer Center. Her research focuses on establishing and improving the efficacy of traffic and performance characterization methodologies on wide-area communication networks, in particular to cope with the changing traffic workload, financial structure, and underlying technologies of the Internet. San Diego Supercomputer Center, P.O. Box 85608, San Diego, CA 92186-9784; *kc@sdsc.edu*.

Andrew Gross is a doctoral student in the Department of Electrical and Computer Engineering at the University of California, San Diego, and is currently a security analyst at the San Diego Supercomputer Center. San Diego Supercomputer Center, P.O. Box 85608, San Diego, CA 92186-9784; *grossa@sdsc.edu*

References

[1] J. Kohl, C. Neuman, and J. Schiller, "Kerberos: An authentication service for open network systems," in *Proc. Winter USENIX 1988*, 1998. Dallas,TX.

Table 3: packet traces of single keystroke with two versions of *klogin*

original *klogin*, which sends length of encrypted data in one packet and the actual data in another
01:05:05.374333 host1.5120 > host2.eklogin: P 1774:1778(4) ack 39052 win 4096
01:05:05.828330 host2.eklogin > host1.5120: . ack 4 win 4096
01:05:05.828583 host1.5120 > host2.eklogin: P 4:20(16) ack 1 win 4096
01:05:06.036337 host2.eklogin > host1.5120: P 1:5(4) ack 20 win 4096
01:05:06.220141 host1.5120 > host2.eklogin: . ack 5 win 4096
01:05:06.409713 host2.eklogin > host1.5120: P 5:21(16) ack 20 win 4096
01:05:06.420170 host1.5120 > host2.eklogin: . ack 21 win 4096
modified version of *klogin*, which sends data and length in single packet
00:15:07.758686 host1.5120 > host2.eklogin: P 60579:60599(20) ack 36408 win 4096
00:15:07.782656 host2.eklogin > host1.5120: P 1:21(20) ack 20 win 4096
00:15:08.001707 host1.5120 > host2.eklogin: . ack 21 win 4096

[2] A. Gross, "Klogin support for compression," May 1995.

[3] V. Jacobson, "Compressing TCP/IP headers for low-speed serial links." RFC 1144, February 1990.

[4] C. Thomborson, "V.42bis standard for data--compressing modems," *IEEE Micro*, pp. 41–53, October 1992.