Improving the Database Logging Performance of the Snort Network Intrusion Detection Sensor

Lambert Schaelicke, Matthew R. Geiger, Curt J. Freeland

Department of Computer Science and Engineering
University of Notre Dame

Technical Report 03-10

November 2003

1 Introduction

Network intrusion detection systems have become one of several invaluable tools to safeguard critical infrastructure and information. Publicly available network intrusion detection systems (NIDS) such as Snort [3][5] and Bro [2] as well as a large number of commercial systems complement other security mechanisms by passively monitoring a network link for possible intrusions and other security breaches. Alerts about possible violations are forwarded to security personal and are often also stored in databases for further analysis and correlation [1].

The performance of a NIDS can be described by its ability to detect true attacks in the stream of network traffic it observes. In addition to the sophistication of the intrusion detection algorithm employed, processing speed is a key consideration for the overall performance. If the NIDS is unable to process network traffic at the rate it arrives, packets are dropped and valuable information may be lost. Significant packet loss negatively affects the overall NIDS effectiveness.

The performance requirements of the popular Snort NIDS has been studied before [4]. However, in addition to the performance of the NIDS sensor itself, the database that receives and stores alerts can play a role in determining overall performance. On a system under attack, the NIDS sensor can potentially generate a large number of alerts over a short period of time. If the database server is unable to absorb alerts at the offered rate, important alert data is lost and the entire intrusion detection system is rendered inefficient. This problem is compounded if multiple NIDS sensors report to the same database system.

For example, a dual-processor 800 Mhz Pentium-III system running MySQL is able to log 412 alerts per second into a MySQL database server configured for the ACID intrusion detection analysis console [1]. A 100 Mbit Ethernet link at 50 percent utilization can carry approximately 23855 packets with an average size of 512 bytes per second. Given the maximum database throughput, at most 1.7 percent of all packets can trigger an alert before the database server is overloaded and alerts are lost. Clearly, in certain attack situations, the rate of alerts can exceed the database throughput, resulting in loss of alert information.

2 Performance Optimization

The ACID intrusion detection analysis console [1] stores intrusion alerts generated by one or more sensors in an SQL database server for further analysis and correlation. The open-source Snort NIDS includes a plug-in module that interacts with a database server and logs alerts in an ACID-compatible format. Intrusion alerts are stored in a variety of tables, depending on the alert and packet type.

Each alert issued by a sensor is added to a table of events. Event table entries reference signatures in a separate table, and refer to additional information about the alert in tables that store IP, TCP, UDP or ICMP header information, the packet payload and other optional information. The relationship between events and signatures is established by unique, ACID-internal signature identifiers that map events to signatures.

On startup, a Snort sensor obtains a sensor ID from the database server, and registers itself in a table of active sensors. When generating an alert, the sensor looks up the alert signature in the signature table to obtain the database internal signature ID. If a signature is encountered for the first time, it is not present in the table and the sensor inserts the signature and related information in a number of database tables, before obtaining the unique signature ID.

This general design of the ACID database tables offers significant flexibility as it supports any number of sensors and does not assume a static set of signatures. Furthermore, alert information is distributed over a number of tables, thus minimizing record size in each table and reducing the amount of wasted space in case not all elements are applicable to a specific alert.

However, the flexibility of ACID comes at the cost of higher overhead. For instance, inserting an alert related to an IP/TCP packet into the database requires at a minimum:

- a lookup of the signature ID
- insertion of the event into the event table
- insertion of IP header information into the iphdr table
- insertion of TCP header information into the tcphdr table
- possibly, insertion of optional information and payload into the respective tables

Note that the last four steps involve updates to the index in addition to inserting the record, generally implemented as expensive read-modify-write sequences on the index file. An analysis of the I/O behavior of the database server reveals that a significant amount of time is spent accessing the signature table. The table is accessed for every alert and accounts for almost 25 percent of all I/O system calls. Interestingly, most I/O system calls complete in under 15 microseconds, indicating that the majority of file data is successfully cached in main memory. Signature table accesses, however, exhibit a significantly higher access time as a result of frequent long-latency disk accesses. This effect is in part due to the file cache policy that prefers to evict read-only data over modified disk blocks. It is furthermore observed that during normal operation, the signature table is almost never updated, once it has been populated with the commonly encountered signatures.

Combined, these observations indicate that caching the signature table at the NIDS sensors is able to eliminate an expensive database access without loss of functionality of the ACID database, resulting in improved alert insertion performance. The signature cache is implemented as a hash table inside the Snort database plug-in. Each hash table entry consists of the Snort signature identifier, signature name, revision number and the ACID signature ID. The hash table is indexed by the Snort signature ID and is arranged as an array of linked lists. When generating an alert, the database plug-in first attempts to look up the signature in the hash table. Upon success, the internal signature ID is used to insert the event and related information into the various ACID tables. If a given signature is not present in the local hash table, the plug-in looks it up in the ACID signature table. The signature may be found in the database table if a sensor is encountering a signature for the first time, but it has been inserted previously by another sensor (or an earlier session of this sensor). If the signature is not found in the database table, it is inserted along with related information such as the reference ID and signature class. In either case, the database provides the internal signature ID to the sensor, which then inserts a new record in its hash table.

Note that even though the local hash tables cache the central database table, coherency is not a concern since signatures are normally not deleted from the table. Individual sensors populate their hash tables lazily as signatures are encountered, with the central database assigning unique internal signature identifiers. The local cache increases Snort's memory requirements by a modest amount of 272 bytes per signature. The overhead of the hash table search is negligible compared to the savings in database query latency. Consequently, optimizing the local hash table layout to minimize average hash chain length is not a concern. Note that this optimization has no impact on the database table structure or on the database server.

3 Evaluation

To evaluate the effectiveness of the sensor signature cache, trace-based experiments are performed involving two PC/Linux hosts acting as a sensor and database server. The sensor is a 1.7 Ghz Pentium-4 system with 256 Mbyte of main memory, running Debian Linux 2.4.19 and Snort 1.9.1. The sensor is connected to a database server with two 800 Mhz Pentium-III Coppermine processors, 1.2 Gbyte of memory, running Debian Linux with the 2.4.19 kernel and MySQL 3.23.56 [6]. Database tables are stored on a 210 Gbyte RAID-0 array consisting of 6 Ultra-SCSI-3 disk drives hosted on a Smart-2P RAID controller. In the experiments, the Snort sensor is reading a network trace in tcpdump format from disk.

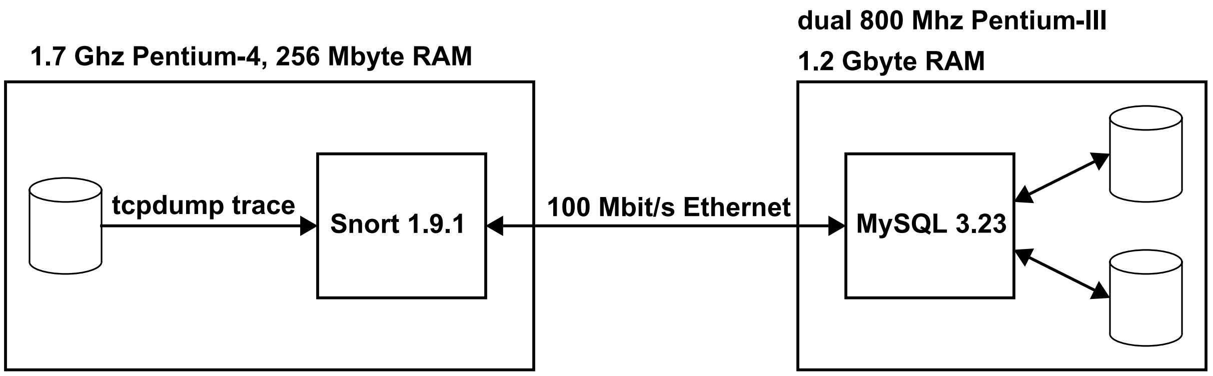

Figure 1: Experimental Setup

To expose the database performance bottleneck, an artificial Snort rule is used that logs every packet as an alert. While this setup does not correspond to the normal use of network intrusion detection systems in production environments, it eliminates the NIDS sensor as a bottleneck and thus approximates the system behavior in the event of an attack when the NIDS sensor rapidly generates a large number of alerts.

In this environment, the unmodified system is able to read, process and issue alerts for 444,611 packets in 18 minutes, at an average rate of 412 alerts per second. The trace file contains the first 68 bytes of every packet captured off a campus internet connection on Sunday, December 8, 2002, between 10:21 and 10:36 a.m. With the sensor signature cache, the processing time for the same trace is reduced to 14 minutes, resulting in an average rate of 530 alerts per second and an improvement of 28.6 percent over the unmodified configuration. In both experiments, all database tables are initially empty.

A second experiment uses the default Snort 1.9.1 ruleset on a larger trace, consisting of 15,561,199 packets captured on the same campus network link on Monday, December 2, 2002, between 12:01 and 12:10 a.m. This setup corresponds more closely to a realistic NIDS environment, as only a subset of the observed packets trigger alerts. The unmodified system requires 31.5 minutes of processing during which it logs 237,279 alerts, while the signature cache reduces this time by 9.5 percent to 28.5 minutes.

These results demonstrate that alert logging overhead can limit overall NIDS performance, especially under high load such as an attack situation. Caching frequently used database information at the sensors reduces logging overhead by over 20 percent, resulting in a direct and corresponding performance improvement of the NIDS sensor in cases of high alert rates.

4 Summary

The performance of a network intrusion detection system depends on both the method of detecting intrusions as well as its processing overhead. If the sensor is not able to process network traffic at the rate offered by the link under observation, packets are dropped and valuable information may be lost. The method of logging alerts has a direct impact on the sensor overhead, and can further limit the NIDs sensor performance in scenarios with high alert rates. This paper presents and evaluates a performance optimization technique that caches the contents of a database table to reduce the number of queries. When applied to the Snort intrusion detection sensor and the ACID database, this technique reduces alert logging overhead by 25 percent.

A modified database plug-in for Snort 1.9.1 is available in source code form at no cost at:

```
http://www.cse.nd.edu/~spanids/software.php
```

The source code has been extensively tested with Snort 1.9.1 and is compatible with Snort 2.0.

The authors wish to thank the entire Spanids research team for their support. This material is based upon work supported by the National Science Foundation under Grant No. ANI02-31535. Any opinions, findings, and conclusions expressed in this material are those of the authors and do not necessarily reflect the views of the National Science Foundation.

5 References

[1] R. Danyliw, *"ACID: Analysis Console for Intrusion Databases,"* http://acidlab.sourceforge.net, 2001

[2] V. Paxson, "Bro: A System for Detecting Network Intruders in Real-Time," *Computer Networks*, vol. 31, no. 23-24, 1999, pp. 2435-2463.

[3] M. Roesch, *"Snort – Lightweight Intrusion Detection for Networks,"* Proc. Usenix LISA '99 Conf., November 1999. http://www.snort.org/docs/lisapaper.txt

[4] L. Schaelicke, T. Slabach, B. Moore and C. Freeland, "Characterizing the Performance of Network Intrusion Detection Sensors," *Proc. Sixth International Symposium on Recent Advances in Intrusion Detection* (RAID 2003), Lecture Notes in Computer Science, Springer-Verlag, Berlin - Heidelberg - New York, 2003.

[5] *"Snort 2.0 - Detection Revisited,"* whitepaper, Sourcefire Network Security Inc, 2002.

[6] TcX AB, Detron HB, and Monty Program KB, *"MySQL Reference Manual Version 3.2.2,"* http://www.mysql.com/documentation/mysql/