A FRAMEWORK FOR THE RAPID DEVELOPMENT OF ANOMALY DETECTION ALGORITHMS IN NETWORK INTRUSION DETECTION SYSTEMS

Richard J Barnett[1] and Barry Irwin[2]

Security and Networks Research Group
Department of Computer Science
Rhodes University
Grahamstown, South Africa

[1]barnettrj@acm.org, [2]b.irwin@ru.ac.za

ABSTRACT

Most current Network Intrusion Detection Systems (NIDS) perform detection by matching traffic to a set of known signatures. These systems have well defined mechanisms for the rapid creation and deployment of new signatures. However, despite their support for anomaly detection, this is usually limited and often requires a full recompilation of the system to deploy new algorithms.

As a result, anomaly detection algorithms are time consuming, difficult and cumbersome to develop. This paper presents an alternative system which permits the deployment of anomaly detection algorithms without the need to even restart the NIDS. This system is, therefore, suitable for the rapid development of new algorithms, or in environments where high-availability is required.

KEY WORDS

NIDS, prototyping, rapid development, anomaly detection, Perl, Snort, frameworks

A FRAMEWORK FOR THE RAPID DEVELOPMENT OF ANOMALY DETECTION ALGORITHMS IN NETWORK INTRUSION DETECTION SYSTEMS

1 INTRODUCTION

With the increased frequency of intrusion attempts from the Internet, simple passive technologies such as firewalls are no longer sufficient on their own. Systems which actively monitor traffic within a network are now commonplace and act as a second line of defence for cases when intruders evade first line defences. These systems, known as Network Intrusion Detection Systems (NIDS), act on traffic in one of two specific ways. They either perform signature detection, or they perform anomaly detection. Most current NIDS perform signature detection as a primary feature and perform anomaly detection on a more ad-hoc basis.

While these systems have well defined mechanisms for the addition and alteration of signatures, they have fairly labour intensive methods for the introduction of new algorithms for anomaly detection. This is because these systems usually require new modules to be developed in a compiled language (such as C) and then be loaded. Many of these systems require that these modules be compiled into the NIDS and this process increases the mean time to deployment.

This paper proposes an alternative system which permits the rapid development and deployment of anomaly detection algorithms. This system has its primary use in development environments where the additional time required for other systems makes development cumbersome, and in environments where the deployment of additional (new) algorithms is essential with little or no downtime of the IDS.

1.1 Paper Organisation

The remainder of this paper is structured as follows. Section 2 highlights some of the literature which influenced the design of this framework. Section 3 then presents this design, followed by the implementation in Section 4. Thereafter, Section 5 presents a case study of its use, and highlights

some performance considerations therein. Finally, the paper is concluded in Section 6.

2 BACKGROUND

The field of Intrusion Detection, and Network Intrusion Detection inparticular, is one with a considerable volume of literature, some of which is related to the construction of this framework, and our extended research. This section highlights some of this work, and describes the context of our extended study.

The authors' ongoing research is focused into the effective detection of network scanning, and was fueled by previous research at Rhodes University [4] which discovered several flaws in the detection methods in popular NIDS such as Snort [10] and Bro [8].

The design of the framework needed to consider numerous factors in scan-detection, network intrusion detection and IDS design. For the purposes of our research, we define scanning as it is defined by Allman *et al.* in [1]. That is, connections being determined as good, bad and unknown (based on the success of the connection) and the classification of hosts as scanning hosts if they produce a majority of bad connections. This definition, coupled with our own taxonomy of network scanning [2] form the basis of a number of algorithms into scan-detection.

Existing research into the construction of a NIDS is scarce, however there is a small selection of research which is of interest. Lee and Stolfo [6] present research into the construction of a novel framework for automated data mining based intrusion detection. Their system, MADAM ID, performed admirably in the 1998 DARPA Intrusion Detection Evaluation [7]. Yang *et al.* [12] present a framework for the use of expert systems and clustering analysis in amalgamating misuse and anomaly detection systems.

In their seminal papers on Snort and Bro (respectively), Roesch [10] and Paxson [8] describe the structure and internal workings of each system. Both Snort and Bro have design features based on the older NIDS, The Network Flight Recorder (NFR) [9] which pioneered the field.

3 FRAMEWORK DESIGN

The design of a framework for the rapid development of anomaly detection algorithms takes a number of important considerations. This section details a number of those considerations and how they influenced the design of the system.

The design of the framework was based on a number of criteria. These criteria were formulated both from our own need, and from the observed literature. The following were considered:

Light-Weight The system needed to be small and efficient with a minimal code base, to keep resource usage by the base system as low as possible.

Efficient While the system was designed for prototyping algorithms, it was important to be able to test these algorithms on production traffic. To this end, the system needed to perform well on commodity desktop grade hardware and be able to process traffic from gigabit networking in real time.

Interpretable One of the biggest flaws with the development of preprocessors for systems such as Snort, is the extensive compile time whenever a change is made. For this reason, our framework was required to use an interpreted language.

End User Simplicity Finally, the base framework should be independent from each test. This permits authors constructing tests to have limited knowledge of how the base system works. This has the added benefit of permitting changes to the base system without tests having to be rewritten, provided that the interface is well thought out and remains unchanged.

Many of these features are common with Snort and Bro's original design. However, other considerations made for those systems are either implicit in our design, or have been ignored completely.

As a result of these considerations, the framework has been designed to be simple. Figure 1 illustrates this design, and it can be seen that there are only three components to the system. Each of these will be discussed separately.

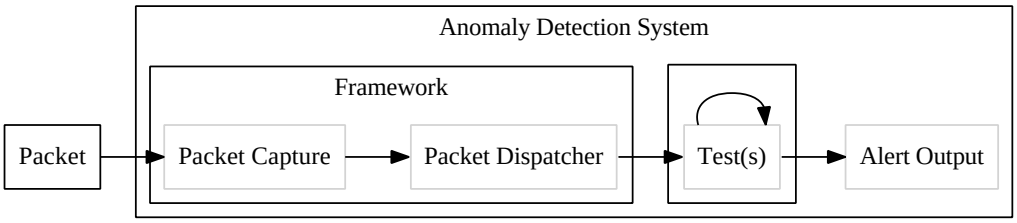

Figure 1: Framework Structure

3.1 Packet Capture

The packet capture component of the system is designed to provide an abstract packet data structure to the remainder of the system. This provides a mechanism for different Packet Capture and Dispatcher implementations. As the system is intended to be Light-Weight, the design strips all unnecessary information from each packet before passing it onto the Dispatcher. The packet capture component needs to be as efficient as possible as it is the point at which the most intensive IO occurs within the system, and is discussed further in Section 5.1.

3.2 Packet Dispatcher

The packet dispatcher component is possibly the most important in the system. It is this component that loads each of the tests discussed in Section 3.3 and passes the packets captured from the component discussed in Section 3.1. It ensures that incoming packets are processed by the tests, without being dropped and that the tests are executed in order. It also provides an alert mechanism for tests, and the ability to load tests in real-time.

3.3 Test(s)

The final component(s) of the framework is(are) the test(s), or the algorithms themselves. As illustrated in Figure 1, the tests are not strictly part of the framework, but a separate element of the overall system. The interface provided by the dispatcher permits several tests to be loaded in a specific order and executed, in real time. The framework allows tests to be loaded and executed without restarting the system. This is to permit the rapid development of algorithms, but at the same time, to permit high availability.

4 FRAMEWORK DEVELOPMENT

Given the design considerations discussed in Section 3. This section presents the development decisions made in order to achieve the desired design objectives. The authors settled on Perl as an appropriate language for the construction of the framework. This was to permit both the interpreted requirement, but to retain some level of efficiency. Each of the components illustrated in Figure 1 is implemented as a Perl module.

The Comprehensive Perl Archive Network (CPAN) [3] contains numerous modules to perform a wide variety of functions. Amongst these are modules for packet capture, and the authors have made full use of the `Net::Pcap` library, and its interface to *libpcap* to perform efficient packet capture.

The framework was constructed to be threaded, and as a result we could make use of the built in queue system in Perl for the dispatching of packet data. However, we found that Perl's threading system does not share variables between threads by default, and that it is unable to share complex data structures. This limitation was overcome by developing a sharable wrapper for complex data structures, which abstracts the complexities of sharing data from the end user, at a fairly limited performance cost.

The threading system processes each packet in a single thread, and therefore guarantees that tests will run in the correct order for a packet. It does not, however, ensure that packets will be processed in the order that they are lifted from the wire. As IP does not guarantee that packets are received in order, this is of little significance. Perl's ability to compile and recompile code on the fly permitted us to allow the loading and reloading of modules automatically. This fulfils the design goal of permitting rapid development.

5 CASE STUDY: NETWORK SCANNING DETECTION

The authors have developed this framework for use in their ongoing research into Scan Detection. This section presents a look at the usability of the framework in the context of rapid development of scan-detection algorithms. The developed algorithms have been tested against both live traffic as it is read off the wire, and traffic previously captured in *pcap* files.

We find that we are able to manipulate algorithms in real time, and observe

the effects of our changes. An unintentional side effect of the implementation of the module loading is, however, that memory is garbage collected during a reload, and it is equivalent to restarting the system.

5.1 Performance Considerations

Given that Snort [10] is written in C/C++, it follows that it should offer significantly better performance to our own system, and that the interpreted nature of Perl should offer lower performance. However, since Perl compiles on demand, it offers good performance with a start-up time cost, and a memory cost. Neither of these limitations are of concern, however, and the overall performance remains good.

Despite this we find that the system has a few performance limitations. The threaded testing system permits tests to run effectively, however, we note that the interface to *libpcap* is not as efficient as it could be, and under heavy load the packet capture and dispatch thread pushes the CPU load up. We have not, however, observed any packet loss.

Memory use is significant and is largely due to the design of the tests, rather than due to the framework. The packet data structure is, however, fairly well designed, and a large number of packets can be easily stored in memory.

Our observations are that the packet capture framework can be deployed in a production environment, but that (like with most IDSs [5]) it would perform significantly better when run on its own dedicated system.

5.2 Comparison with Snort

Snort is a popular, Open Source NIDS, but is focused on signature detection, rather than anomaly detection. This paper will not comment on the relative effectiveness of the two systems, only on how they operate. Our framework was developed to permit rapid deployment of new algorithms, something which is not performed effectively by Snort. Our system is more CPU intensive than Snort, under load, but not as memory intensive.

Snort offers more sophisticated pre-processing of packets, including stream reassembly and a rule matching system, which make it a fairly large and cumbersome application, and while it is a superb signature detection system, we find that our system processes packets more efficiently if you are only interested in scan-detection. This is largely because we have no interest in

passing packets through the signature engine.

6 CONCLUSIONS

This paper has presented the design and implementation of a novel framework for the development of anomaly detection algorithms in the context of network intrusion detection. It has presented the design, and implementation of the framework, and has illustrated how it could be used in the development of anomaly algorithms, in our case scan-detection algorithms.

We have seen that the concept of developing NIDS in an interpreted language is viable, and that Perl is a suitable choice. We have also seen that the overhead for an anomaly detection framework is significantly lower than that for a full blown signature-detection system such as Snort.

6.1 Future Work

This research has primarily focused on the development of a framework for the rapid development of anomaly detection algorithms. In particular, we have considered scan-detection. Our research into scan-detection is ongoing and is largely supported by this framework. The framework could, however, be extended in a number of ways. It is a problem which could be offloaded to commodity graphics cards (in a similar way to Gnort [11]) and to integrate it into existing NIDS, such as Snort.

ACKNOWLEDGEMENT

The authors would like to acknowledge the support of Telkom SA, Comverse, Tellabs, Stortech, Mars Technologies, Amatole Telecommunication Services, Bright Ideas Project 39, THRIP and the NRF through the Telkom Centre of Excellence in the Department of Computer Science at Rhodes University.

References

[1] Allman, M., Paxson, V., and Terrell, J. A brief history of scanning. In *IMC '07: Proceedings of the 7th ACM SIGCOMM conference*

on Internet measurement (New York, NY, USA, 2007), ACM, pp. 77–82.

[2] Barnett, R. J., and Irwin, B. Towards a taxonomy of network scanning techniques. In *SAICSIT '08: Proceedings of the 2008 annual research conference of the South African Institute of Computer Scientists and Information Technologists on IT research in developing countries* (New York, NY, USA, 2008), ACM, pp. 1–7.

[3] Hietaniemi, J. Cpan - comprehensive perl archive network. Online: `http://www.cpan.org/`, April 2001.

[4] Irwin, B., and van Riel, J.-P. Inetvis: a graphical aid for the detection and visualisation of network scans. In *Conference on Vizualisation Security (VizSec2007)* (2007).

[5] Kohlenberg, T., Alder, R., Dr. Everett F.Carter, J., Foster, J. C., Jonkman, M., Marty, R., and Poor, M. *Snort Intrusion Detection and Prevention Toolkit*. Syngress Publishing Inc., 2007.

[6] Lee, W., and Stolfo, S. J. A framework for constructing features and models for intrusion detection systems. *ACM Trans. Inf. Syst. Secur. 3*, 4 (2000), 227–261.

[7] Lippmann, R. P., Fried, D. J., Graf, I., Haines, J. W., Kendall, K. R., McClung, D., Weber, D., Webster, S. E., Wyschogrod, D., Cunningham, R. K., and Zissman, M. A. Evaluating intrusion detection systems: The 1998 darpa off-line intrusion detection evaluation. In *in Proceedings of the 2000 DARPA Information Survivability Conference and Exposition* (2000), pp. 12–26.

[8] Paxson, V. Bro: a system for detecting network intruders in real-time. In *SSYM'98: Proceedings of the 7th conference on USENIX Security Symposium* (Berkeley, CA, USA, 1998), USENIX Association, pp. 3–3.

[9] Ranum, M. J., Landfield, K., Stolarchuk, M., Sienkiewicz, M., Lambeth, A., and Wall, E. Implementing a generalized tool for network monitoring. In *LISA '97: Proceedings of the 11th USENIX conference on System administration* (Berkeley, CA, USA, 1997), USENIX Association, pp. 1–8.

[10] Roesch, M. Snort - lightweight intrusion detection for networks. In *LISA '99: Proceedings of the 13th USENIX conference on System administration* (Berkeley, CA, USA, 1999), USENIX Association, pp. 229–238.

[11] Vasiliadis, G., Antonatos, S., Polychronakis, M., Markatos, E. P., and Ioannidis, S. Gnort: High performance network intrusion detection using graphics processors. In *RAID '08: Proceedings of the 11th international symposium on Recent Advances in Intrusion Detection* (Berlin, Heidelberg, 2008), Springer-Verlag, pp. 116–134.

[12] Yang, D.-g., Hu, C.-y., and Chen, Y.-h. A framework of cooperating intrusion detection based on clustering analysis and expert system. In *InfoSecu '04: Proceedings of the 3rd international conference on Information security* (New York, NY, USA, 2004), ACM, pp. 150–154.