

Network Stream Splitting for Intrusion Detection

J. D. Judd[†], J. C. McEachen[‡], J. B. Michael[†], and D. W. Ettlich^{†‡}

[†]Department of Computer Science

[‡]Department of Electrical and Computer Engineering

Naval Postgraduate School

833 Dyer Road

Monterey, California, USA 93943

Email: {jdjudd | mceachen | b michael | dwettlich}@nps.navy.mil

Abstract--One of the most significant challenges with modern intrusion detection systems is the high rate of false positives they generate. In order to lower this rate, we propose to reduce the amount of traffic sent to the intrusion detection system via a filtering process termed stream splitting. Each packet arriving at the system is treated as belonging to a connection, with each connection assigned to a network stream. A network stream is sent to an analysis engine tailored specifically for that type of packet data. To demonstrate a stream-splitting capability, both an extendable multi-threaded architecture and prototype were developed. Results from testing the prototype demonstrate its ability to capture traffic with a low rate of packet loss at network speeds up to 20 Mbps, and that the stream splitter correctly implements the traffic separation scheme specified for use in the tests.

I. INTRODUCTION

A. Background

In general there are two types of intrusion detection systems (IDS): signature- and anomaly-based systems. The former rely on a database of known attacks with which to compare network traffic in an effort to determine if an attack is in progress. In contrast, the latter rely on the ability to determine what characterizes normal traffic and compare sensed network traffic to what is expected. There are several challenges associated with improving the performance and capabilities of both types of IDS. One of the challenges identified in the results of the study reported in [2] is to reduce the number of false positives generated by such systems.

One possibility for addressing the aforementioned challenge is to apply a type of filtering termed “IDS stream splitting,” which consists of classifying each packet as a member of a stream when it is encountered. Network traffic can be viewed as a collection of connections. For the purpose of this paper, a connection is a data path between a system on the outside of the network to be protected and a system inside the network. A connection can be characterized by the source IP address, the destination IP address, and the service that is in use. Each packet can then be associated with either an existing active connection, or a new Never-Before-Seen (NBS) connection. This classification allows for the stream of network traffic to be split up into sub-streams based on type of service (e.g., http, ftp) or some other criterion.

The results of a recent industry study indicate that the traffic on a production-level network can cause many IDSs to fail [5]. These systems consume all available resources with logging processes or false alarms. In contrast, a stream splitter can in theory reduce the amount of traffic going to any one IDS in a logical grouping of such systems—a special type of sensor network—by distributing the workload amongst the IDS. In the worst case, all traffic would be of the same type and as such would be directed to a single IDS. If the splitter can distribute the traffic to any extent, *ceteris paribus*, system up-time should in theory be as good as or better than that of the single-channel approach. In other words, a single channel introduces a bottleneck, whereas the splitter can reduce the likelihood that any one channel to an IDS will reach its saturation point.

The overall detection scheme can be implemented using primarily commercial-off-the-shelf (COTS) products, with the exception of the implementation for the splitter, providing for both ease of deployment.

B. Related Efforts

In [8], previous efforts to reduce false positive rates are listed; this list includes placing the IDS behind a firewall, tuning the signatures used for detection, and using network analysis to filter the false positives from the alarms that are generated. Placing the IDS behind a firewall is one of the easiest reduction techniques to implement. Performing network analysis on generated alarms is both time consuming and requires a detailed understanding of the network. The effect of using a stream splitter is similar to that produced by placing a firewall between the IDS and the network stream, albeit an intelligent firewall. However, a major difference between a firewall and our splitter is that no traffic will be dropped; it can only be diverted.

The results of the 1999 DARPA Intrusion Detection Evaluation performed by MIT Lincoln Laboratory brought to light some of the problems that plague modern IDS [1]. The signature-based systems were able to alert the operator for a number of the data set attacks. Unfortunately, recognizing these attacks in the presence of heavy network-traffic loads, and recognizing a legitimate alarm amongst a sea of false positives remains a challenge. We hypothesize that by using a stream splitter the performance

of a COTS IDS can be improved. The black-box nature of the stream splitter allows for ease of deployment in a wide range of detection schemes. Through the use of a stream splitter, it is likely that operators will be able to more easily detect attacks on high-traffic networks and also see a reduction in the number of false positives due to both the reduction in traffic and the ability to finely tune their existing IDSs to the type of traffic each IDS is monitoring.

C. *Stream Splitter Overview*

The splitter operates through the use of sensors. There are two types of sensors: active and passive. Active sensors examine packets and then communicate the results back to a sensor control structure, whereas passive sensors simply receive traffic and act on the data in the packet unilaterally.

The use of these two types of sensors allows the splitter to act as a simple stream isolator or to be the base for a more elaborate traffic-separation scheme, for instance routing packets based on a level of trust. In this case, the splitter could treat each packet as part of a connection and assign a trust ranking to the connection in the range [0..1] using a fuzzy logic model; fuzzy logic (see, for example, [6] for a primer on this subject) can be used to partially associate an object with a set of objects. In [7], fuzzy set theory was used to determine the confidence of an IDS that an event had been correctly classified as an intrusion.

For network intrusion detection, the challenge of paramount importance becomes that of discerning the trustworthiness of each distinct stream of packets. For the splitting operation to be effective, each packet must be assigned a trust level. The trust level is based on the type of connection and the number of times the connection has been seen in a given time frame. If no operator action is taken, then over time as the connection is frequently encountered, the connection will be assigned ever increasing levels of trust until it is no longer sent to the IDS evaluating un-trusted traffic. All NBS connection traffic is viewed as un-trusted. Traffic not sent to the un-trusted traffic IDS will be sent to a trusted-traffic IDS to ensure that all network traffic is continuously monitored.

When the splitter acts as a stream isolator, all traffic is sent to a sensor for isolation. If the traffic matches what the sensor is isolating, then further analysis is done. If the traffic does not match, then it is simply ignored by the sensor. In this way multiple sensors can be employed to examine the same data, allowing for only those sensors that find the data useful to act on it. Isolators, being passive sensors, route traffic themselves; thus, there is no additional information that must be communicated back to the sensor control structure.

The idea for investigating the technical feasibility of using splitters is founded on the principles of Huffman coding [3]. The fewer the number of times a specific pattern is detected, be it a connection type or a specific connection, the more information that is present simply by the existence of that item. For example, a mail server that connects to a network every day to deliver mail is not as

suspicious as a NBS telnet connection. Consequently, traffic associated with this mail server can be directly forwarded.

A stream splitter also allows for a detection scheme to grow. For instance a more comprehensive network monitoring system could be built with the addition of an anomaly-based IDS to an existing signature-based IDS: this would provide a network anomaly detection capability in addition to the ability to monitor network traffic for attack signatures.

II. STREAM-SPLITTING MODEL

A. *Streams*

A router takes a network stream as an input and then splits it into multiple streams, typically according to the destination IP address. Just as a router increases a network's performance, by parsing traffic, a stream-splitting mechanism can increase a network's intrusion-detection performance. By parsing traffic before it is sent to an IDS, each IDS can be optimized for processing specific types of streams.

In the context of a computer network, traffic can be split into streams by using a number of different metrics. The split can be based on source, destination, type of service, protocol, etc. The ability to split a traffic stream into sub-streams can, in theory, increase the effectiveness of traffic management.

Network information can be categorized to facilitate its use by the stream splitter. Each connection is viewed as a category, and each packet belongs to a connection: this association can be used to infer additional information about a packet.

It is this additional information that the stream splitter relies on, whether performing fuzzy classification or stream isolation; this is what distinguishes the stream splitter from a router. Existing commercial routers, to our knowledge, do not make a judgment as to the trustworthiness of the packets they route. The stream isolators, apart from performing standard router-style splitting, can also split traffic based on how often the connection has been seen. In this way a low-data-rate, infrequent connection can be separated out from the main network stream for a detailed evaluation. This ability to single out slow, infrequent connections, such as stealth scans, distinguishes the stream splitter from other network analysis tools.

B. *High Level Design*

To get the desired level of performance, a multi-threaded design was necessary. In general there are three parts to the system. Each part is given its own thread of execution.

- The Packet-Capture Engine
- The Packet-Analysis Engine
- The Packet-Injection Engine

The packet-capture engine captures traffic from a network. This traffic is then passed to the analysis engine where it can be analyzed by the appropriate sensors. The final step is to route the packet based on the results of the analyses. The high-level view of information flow is shown in Fig 1. In addition to these three threads of execution, each sensor is given its own thread of execution.

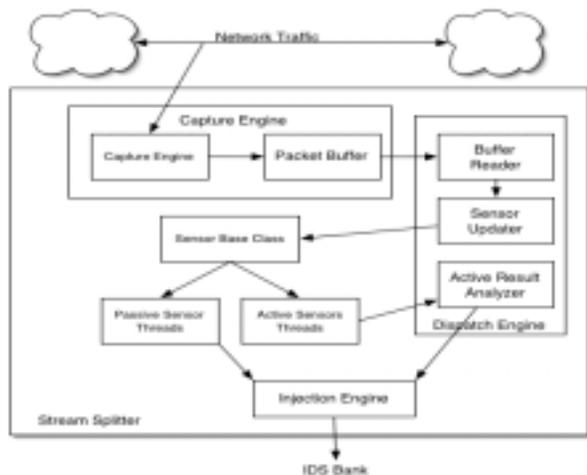


Fig. 1. Stream Splitter Information Flow

Fig. 2 depicts the major components of the stream splitter and their interactions; although it is not explicitly shown here, there can be multiple active and passive sensors attached to a dispatcher. The design of the stream splitter is derived from an extensible architecture that accommodates a wide spectrum of user-defined traffic-separation schemes. Additional sensors may be readily attached to the splitter as needed. In addition, the architecture provides for both designing protections systems with both dynamic separation schemes and reusing stream-splitter and IDS code for unique network configurations.

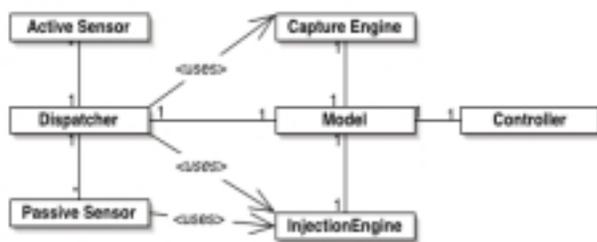


Fig. 2. UML Class-Interaction Diagram

1) Packet Capture

As noted earlier, the capture engine runs in its own persistent thread. The analysis engine takes longer to analyze each packet than the capture engine takes to capture it from the wire. Further, the capture engine must be able to keep up with network traffic. A requirement to be met in the design described here is to be able to capture as many packets as Snort (www.snort.org), a well-known network-analysis engine. As with Snort, the stream splitter is designed for Ethernet traffic.

In order to keep up with network traffic the capture engine reads frames from the wire and places them into a buffer. Once a frame is buffered the capture engine is finished with it and is free to capture the next frame. This allows the system to handle bursts of traffic without significant loss. Packets are then removed from the buffer by the analysis engine. A mutex lock for the buffer is necessary to prevent simultaneous access by the capture engine and the analysis engine. The sequence of events for the capture of a packet is shown in Fig. 3.

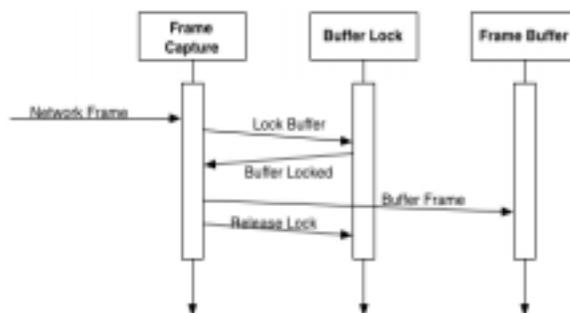


Fig. 3. Frame Capture Sequence

2) Analysis Engine

The core of the stream splitter is a number of sensors. Each sensor looks for a particular metric in the stream of traffic. If the sensor finds that the current packet is part of the sub-stream it is intended to evaluate then it takes the appropriate action. The use of sensors allows for a fine granularity in the type of information that is used to split a stream since a sensor can be made to look at very specific traffic properties. As an example, if web traffic is the metric of interest, a sensor that evaluates all packets that have a source or destination port equal to 80 would accomplish this.

Sensors can be either active or passive. An active sensor will send data to a shared memory location where the system will then act upon the result of the sensor. In a trust scheme where a sensor evaluates a packet for level of trustworthiness, the return would be a trust value. A stream isolator on the other hand simply routes the packet and the splitter does not need to know anything about what the sensor did.

Active sensors are used for any traffic separation scheme requiring multiple sensors to collaborate with each other, such as in a trust-based scheme. These sensors are active because they do not route the packet but must send back information to some control loop that will then make the routing decision. Each active sensor looks at a particular metric and communicates back a trust value based on that metric. Several active sensors may work together to perform a more thorough evaluation of a packet.

When more than one active sensor is used, an additional control structure is needed to gather the aggregate result and make the routing decision. In the prototype, fuzzy logic was used in both the individual sensor analysis and the final routing decision. Fuzzy logic is favorable for

the speed with which a decision can be reached. More precision may be possible using Bayesian statistics; however the speed of fuzzy logic computations outweighs the benefit of the additional accuracy provided by a Bayesian model.

Passive sensors are used for stream isolation. These sensors process each packet: if the packet does not match the criteria the sensor is looking for, then no action is taken. Otherwise, the sensor performs further analysis. For instance a web traffic isolator would look for all TCP traffic with a source or destination port of 80. Once the packet is identified as being of interest, the sensor may perform additional analysis. In the version of the isolator described here, rate analysis is performed on traffic that matches the criteria of the isolator. Routing of the packet is then done based on the results of this rate analysis.

The system was designed with the isolation scheme separated along the same lines as the ISO OSI model. There is a layer-3 isolator that will separate layer-3 traffic out of a stream. There is also a layer-4 isolator that will separate out layer-4 traffic. There is no benefit in performing isolation at layer-2 since this data only represents the current link information. Any isolation above layer-4 is application-specific and requires an indepth knowledge of the applications running on the network. Adding this functionality would be a matter of extending the layer-4 class to accommodate the types of applications being isolated.

The analysis engine consists of the dispatcher and sensors shown in Fig. 1. The dispatcher queries the buffer and then updates all sensors with the new data obtained from the buffer. To ensure that all the sensors are working with the same information, they work in lock step. Though each sensor is independent, one sensor may not begin to work on the next frame until all sensors have completed their analysis of the current one.

A way to achieve this is to use a lock for each sensor similar to the buffer lock used in the capture engine. The dispatcher locks all the sensors and then updates them with the latest data from the capture buffer. The dispatcher then unlocks the sensors, which allows them to lock themselves and process the new information. On completion of its analysis, a sensor will unlock itself, allowing the dispatcher to once more take control. Fig. 4 illustrates this concept.

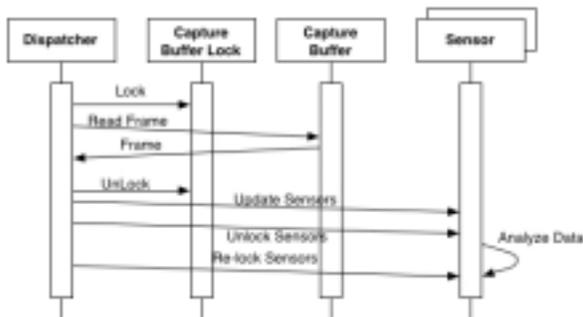


Fig. 4. Analysis Sequence

To ensure that a sensor has processed the information, there is a variable that is reset each time the sensor is updated and set, by the sensor, on completion of analysis. This is necessary because there is no guarantee that a sensor thread will run before the dispatcher tries to relock that sensor. Fig. 5 demonstrates this problem situation in which the sensor is re-locked prior to the completing the analysis. This results in the sensor losing synchronicity with the other sensors. The dispatcher, thinking that all sensors have completed the analysis of the previous packet, will send the next packet, resulting in the second packet not being analyzed.

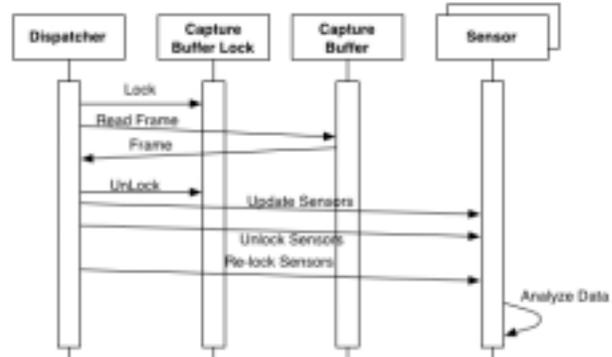


Fig.5. Incorrect Analysis Sequence

3) Injection Engine

Once a frame is captured and analyzed, it is then replayed back to the network. There are two methods of handling the injection. First, the frame can be replayed out of an interface, maintaining the integrity of the layer-2 data. Although this is a simple solution, it requires a separate interface for each output stream. An example of this injection type is depicted in Fig. 6.

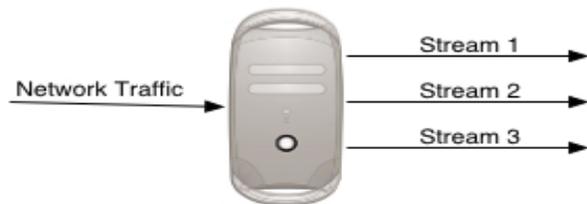


Fig. 6. One Interface Per Stream

The second method explored for replaying and routing outbound traffic is to adjust the destination MAC address. By adjusting the destination MAC address to the address of the interface on the IDS the stream is destined for, several streams may be sent out the same interface and then fed into a switch that will handle the stream separation. Fig. 7 illustrates this.

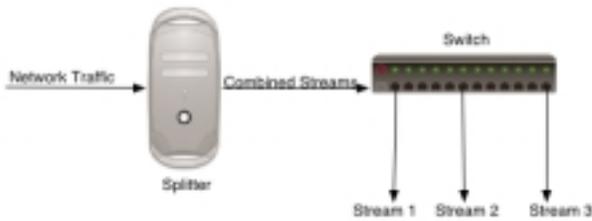


Fig. 7. Multiple Streams Using One Interface

Implementation of this method requires a buffer for the outgoing frames, as one frame may belong to several streams. Once a sensor has determined the destination for a packet and the MAC is overwritten, the frame is then placed into a buffer from which the injection engine reads and replays the Ethernet frames. A lock on the buffer ensures that only one thread is accessing the buffer at a time, allowing all sensors requiring the ability to inject traffic to use the same injection engine. Using one injection engine eliminates both the possibility that more than one entity will try to control the injection mechanism and the need for an additional mutual exclusion lock surrounding the injection device.

III. TESTING AND RESULTS

The goal was to show that a robust architecture for traffic separation could be implemented. To accomplish this two things that had to be demonstrated: (1) the splitter must be able to keep up with network traffic and (2) the splitter must implement a separation scheme that would not be possible with a router. Two tests were devised for this purpose: a capture-efficiency test and a test to measure the accuracy of traffic separation.

A. Capture Efficiency

To test the ability of the splitter to capture network traffic, Tcpreplay (tcpreplay.sourceforge.net) was used to replay a tcpdump file from the 1999 DARPA IDS Evaluation conducted at MIT Lincoln Laboratory. Specifically, the week-one Tuesday inside-dump data was used. Tcpreplay gives the option of specifying how fast to replay the file. The testing started at 5 Mbps and then increased in 5 Mbps increments up to 75 Mbps. After the file had been completely replayed, the capture engine was examined to see how many packets it had captured. This test was run with the splitter, Snort 1.9 and Snort 2.0. The base configuration of Snort was used in both cases. Snort 1.9 was tried with both ASCII and binary logging, while Snort 2.0 used only binary logging. All tests were run on a Macintosh dual 1.42 GHz G4 processor with 2 GB of RAM.

This first test measured how many packets the splitter could capture out of the total packets sent. Analysis of the packets takes longer than the capture process, so once the stream had been sent, analysis was halted and the number of packets sent to the buffer of the splitter was taken to be the number of packets that was captured. This does not reflect the number of packets that can be captured and processed during continuous operation. For the Snort

test, only packets that were analyzed were counted since if a packet is dropped prior to analysis it will not be analyzed. Snort does not buffer packets like the stream splitter.

Snort, implemented in C, slightly outperformed the stream splitter. The results of this test are detailed in Fig. 8. Only the Snort 1.9 data is shown, as there was virtually no difference between versions of Snort 1.9 and 2.0 in this test. After running the test it became apparent that a hardware solution to capture traffic is necessary for any bandwidth greater than 30 Mbps, which corresponds to roughly 10,000 packets per second.

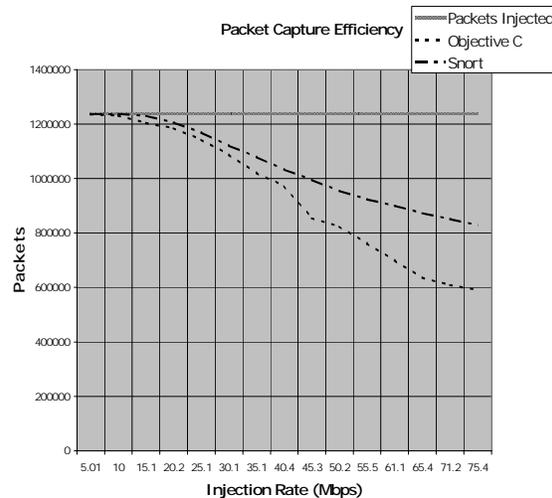


Fig. 8. Packet Capture Efficiency

B. Traffic Separation

To prove that the splitter can implement a traffic-separation scheme, an IDS test bed was used. This test bed consists of a Smart Bits 6000 chassis and six two-port Terametric traffic-generation blades. Each blade can be configured to send out a variety of traffic. For this experiment, two ports were used, one for fast traffic generation and one for slow traffic generation. The output of the splitter was passed to an eight-port Linksys switch to separate the traffic. Connected to the switch was two ports on a Dell 2650 computer running Microsoft Windows 2000 Server, as shown in Fig. 9.

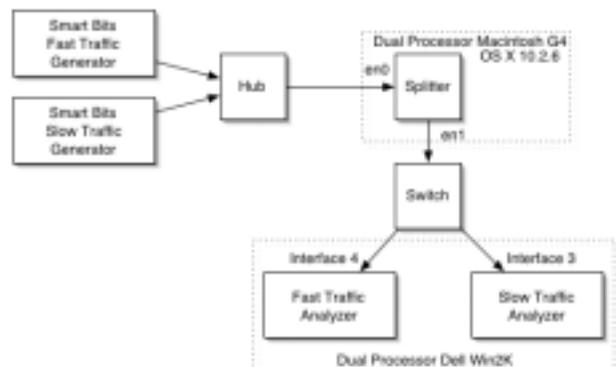


Fig. 9. Test Bed Setup

The splitter was configured to separate TCP traffic utilizing port 80. The window size was set to ten seconds with a high-low traffic threshold of 100 packets; hence, any connection with greater than ten packets per second would be sent to the fast web traffic stream. The IP addresses were chosen so that when viewed in a large table they would stand out from one another. The test traffic is given in Table 1; MAC information is not shown because the splitter overwrites the destination MAC address in the process of routing the traffic.

Table 1. Traffic Stream Description

Traffic Stream	Src. IP	Dest. IP	Src. Port	Dest. Port
Fast	10.10.1 0.1	10.10.10. 2	80	11000
Slow	192.168 .10.10	190.168. 10.20	80	12000

Configuration of the splitter consisted of setting the following parameters:

- MAC addresses for the two interface on the windows server
- A threshold level of 100
- A service port of 80
- A ten-second window
- A protocol type of TCP

Ethereal was used to capture traffic on both of the monitored interfaces for sixty seconds. This experiment was run three times, with the corresponding results shown in Table 2.

Table 2. Traffic Capture Test Results

Test Number	Fast Traffic Packets	Slow Traffic Packets
1	6000	120
2	5900	120
3	5900	120

The results of the experiments are in accord with expectations. Since the starting and stopping of Ethereal could not be synchronized with the traffic generators, the fast traffic should be within 100 packets of 6000 and the slow traffic should be within two packets of 120.

IV. CONCLUSION

We have shown a robust architecture for a network stream splitter capable of parsing traffic based on a large number of metrics. The parsing of traffic potentially allows the IDSs to be tuned for each stream's specific traf-

fic type, thus increasing the network's intrusion detection performance.

Preliminary results described in [7] indicate that fuzzy logic can be used to reduce the false positive rate generated by an IDS. The stream splitter provides a robust architecture that could potentially be used to apply such a traffic separation scheme.

In addition, further work is needed to explore the extent to which the stream splitter can be used to apply multiple traffic isolation and inspection schemes simultaneously in real time.

Lastly, it remains to be demonstrated whether one can improve the effectiveness of software decoys—constructs used to implement deception strategies and tactics (vid. [4])—by fusing data from the stream splitters with data collected by both the decoys' probes (i.e., traces of system-level calls and the responses of suspicious processes to the actions of the decoys) and the supervisors of the decoys (i.e., aggregate information about the interaction among decoys and suspicious processes).

ACKNOWLEDGEMENTS

This research is supported by grants from both the U.S. Department of Navy and Department of Homeland Security. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of the U.S. Government. The U.S. Government is authorized to reproduce and distribute reprints for Government purposes notwithstanding any copyright annotations thereon.

REFERENCES

- [1] Haines, J. W., Lippmann, R. P., Fried, D. J., Tran, E., Boswell, S., and Zissman, M. A. 1999 DARPA intrusion detection system evaluation: Design and procedures, Tech. rpt. no. 1062, MIT Lincoln Laboratory, Lexington, Mass., 26 Feb. 2001.
- [2] Haines, J., Rossey, L., Lippmann, R., and Cunningham, R. Extending the 1999 off-line intrusion detection evaluation, in *Proc. DARPA Inf. Survivability Conf. and Expo.*, IEEE (Anaheim, Calif., June 2001), vol. 1, pp. 35-45.
- [3] Huffman, D. A. A method for the construction of minimum-redundancy codes, in *Proc. Inst. Radio Engineers* 40, 9 (Sept. 1952), pp. 1098-1101.
- [4] Michael, J. B., Auguston, M., Rowe, N. C., and Riehle, R. D. Software decoys: Intrusion detection and countermeasures, in *Proc. Workshop on Inf. Assurance*, IEEE (West Point, N.Y., June 2002), pp. 130-138.
- [5] Newman, D., Snyder, J., and Thayer, R. Eight IDSs fail to impress during the month-long test on a production network, *Network World*, 24 June 2002.
- [6] Pedrycz, W. *Fuzzy Modelling: Paradigms and Practice*, Boston: Kluwer Academic Publishers, 1996.
- [7] Shajari, M. and Ghorbani, A. A. Using fuzzy system to manage false alarms in intrusion detection, in Grizalis, D., di Vimercati, S. C., Samarati, P. and Katsikas, S., eds., *Security and Privacy in the Age of Uncertainty*, Boston: Kluwer Academic Publishers, 2003, pp. 241-252.
- [8] Timm, K. Strategies to reduce false positives and false negatives. <http://online.securityfocus.com/infocus/1463>, 11 Sept. 2001.