Distributed Sensor Objects for Intrusion Detection Systems

Robert F. Erbacher
U.S. Army Research Laboratory
2800 Powder Mill Road
Adelphi, MD 20783
Robert.F.Erbacher.civ@mail.mil

Steve Hutchinson
ICF International Inc., for
U.S. Army Research Laboratory
2800 Powder Mill Road
Adelphi, MD 20783
Steve.E.Hutchinson.ctr@mail.mil

***Abstract*— Effective intrusion detection requires the analysis of enormous volumes of network data collected from distributed sensor objects. This analysis has most often been performed on central servers. The alternative has been to limit data collection to network flow data, with the side effect of reducing intrusion detection effectiveness. This research examines an alternative, namely the incorporation of more intelligent sensor objects. We discuss the infrastructure required to support such a metaphor, the potential tradeoffs, and a novel algorithm for such an intelligent distributed sensor object.**

Keywords- intrusion detection, distributed sensor objects, network infrastructure, network performance, software architecture.

I. INTRODUCTION

When network intrusion detection systems (IDS) are discussed, what is being considered are the algorithms and techniques being employed to analyze raw network-based data for known malicious activity or the indications of anomalies. However, there is a substantial infrastructure required to provide the raw data to the analysis engine. The first component of this infrastructure is a series of sensors distributed around the network with the goal of collecting the raw data. The optimal placement of such sensors is itself an open problem. The goal is to ensure that the sensors possess a purview over the entire network while minimizing data duplication.

The sensors will send this raw data to a central repository. This re-distribution of the data is intrinsically one of the biggest problems with IDS infrastructures. Such data redistribution, especially when attempting to be complete, can significantly add to the network load and delay analysis due to the time required to transmit the data. Typical solutions are to send only a sample of the data or to limit analysis to flow data; in either case, some packets can be immediately categorized as not being of interest and filtered out before transmission to the central server. Network flow data essentially describes an entire transaction sequence from initiation to termination as a single structure. The biggest limitation of network flow data is that it does not incorporate payload information, though this does greatly reduce the volume of the data. The limiting of IDS analysis to flow data can prevent certain types of analysis, e.g., malware identification, since such identification requires payload information. However, more tools appear to be geared towards flow-level analysis than packet trace analysis due to the reduced bandwidth and data storage requirements [13].

The challenge of needing to perform complete analysis, i.e., through complete packet traces, and the requisite data requirements of such complete analysis implies the need for alternative solutions. Our approach is to examine novel sensors that are more intelligent in their distribution of computation between the sensors and the central server than is currently done, as in techniques such as that by Maciá et al. [7]. Tradeoffs remain to be resolved, e.g., how much work should be performed on the distributed sensors and how much work should be performed on the central server. A key question is how this should be handled by the infrastructure, what is an effective algorithm for such distributed sensor objects, and what should be sent to the central server. These questions are the focus of this research; Barford et al. [1] more fully define the problem space. Additionally, Spoor [14] describes practical experiences in rolling out sensors running on COTS hardware.

II. RELATED WORK

While not providing relevant algorithmic techniques, Jahnke [4] provides a relevant modular software infrastructure that specifically allows for the specification of sensor data preprocessing, analyzing, and storage components. Clearly, our techniques can be integrated into this architecture as relevant modules.

Maciá et al. [7] incorporated the idea of a smart sensor that will adaptively transmit analyzed data to other IDS components, e.g., a central server. Fundamentally, their approach is based on using a fully distributed IDS, performing IDS tasks on the sensor and distributing the results of the analysis as needed.

Barford et al. [1] used simple threshold-based methods for filtering at the sensors. They also employed analytic methods for aggregation. Their filtering research most closely relates to our work, though such simple thresholding will clearly not provide detection results as good as our proposed approach.

Eid et al. [2] discuss a mobile agent based approach. Their approach, while using mobile agents to collect data from sensors, collects all IDS related data. No attempt is made to reduce the scope of the collected data.

Quanz et al. [12] explored the feasibility of *fully* distributed anomaly detection. This is in contrast to the

hybrid approach we proposed. A full comparison of the two approaches remains to be done.

An enormous amount of work has been performed in distributed sensor networks. While this work covers areas outside of intrusion detection and computer security in general, a long-term goal must be to perform a full survey and identify relevant research that can be applied to this domain; the survey by Yick et al. provides a starting point for this effort [17].

Related to the work on wireless sensor networks, Mamun et al. [8] explored hierarchical models for intrusion detection. This approach is particularly relevant to widely distributed, geographically, networks. This approach essentially implements a fully distributed IDS but in a hierarchical fashion such that alerts ripple up the hierarchy.

Another domain seeing large-scale research efforts is in the domain of data aggregation. McEachen et al. [9][10] discuss an approach for data aggregation in sensor networks in which there are tens, hundreds, or thousands of sensors. This aggregation work focuses on performing all aggregation on the central server, with the individual sensors still reporting all data.

Additionally, a lot of related work has been performed in the robotics field that cannot be ignored. For instance, Stroupe et al. [15] examined the representation, fusion, and communication of sensor data from multiple robots in noisy scenarios.

While there is enormous work in relation to distributed sensors and distributed IDSs, previous work has primarily focused on fully distributed IDSs or has incorporated only limited filtering mechanisms. Our approach to develop sensor data reduction techniques in a hybrid sensor/central server approach is clearly novel.

III. Technical Approach

The fundamentally distributed nature of intrusion detection systems suggests that performance can be improved by opportunistic utilization of sensor processing. Consider that even the simplest sensors these days will have extensive processing power; e.g., consider the compute resources of even the simplest of today's smart phones. Clearly, the compute capability of the sensor can be used to aid in the intrusion detection analysis process.

Thus, a key design decision in a distributed architecture is where to locate this transmission process within the pipeline of analytical processing.

In most architectures, the sensor performs minimal processing. Instead, all raw data must be transmitted to the central server for processing. The alternate extreme would require that the totality of processing be performed on each sensor. This would move the data transmission step from immediately after raw data collection to occur after aggregation, but still preceding presentation, which must be provided to the analyst. This architecture has been studied theoretically and has been described as a "biologically-inspired" or "human-immune-system inspired" IDS. The work of J. Kim [2][6] is representative of this approach, in which numerous, fine-grained software objects run concurrently on each sensor. Each object is designed to detect and alert on a specific signature or pattern. Such objects often follow a discrete life cycle; they are instantiated from a repository of object generators, execute on the sensor, and are extinguished or replaced with updated configurations in the future.

Table 1: Architectural process elements comprising an IDS; distilled from [11].

Process Element #	Label and Description
1	Network traffic data collection
2	Information, i.e., feature, extraction
3	Tool execution. A range of tools may be incorporated with distinct or overlapping responsibilities; each tool will use individually configured signatures and rule sets
4	Alert generation
5	Aggregation and detection
6	Presentation

The selection of an IDS architecture determines the location of each of the process elements, table 1, in the IDS process. IDS architectures, all the way back to syslog, incorporate distributed data collection; collecting traffic from multiple sites and networks with multiple sensors. In such IDS architectures, all collected traffic is transmitted to a central analysis facility for subsequent processing. Thus, we see that process element 1 (traffic collection), from table 1, is mapped to the sensor, while process elements 2-6 are mapped to a central analysis facility.

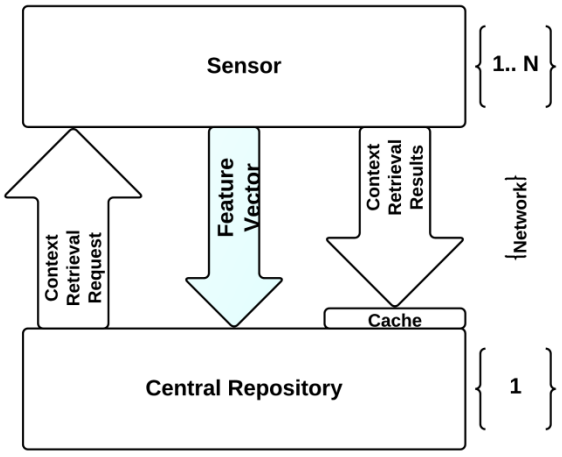

Figure 1: Overall DSO architecture.

A. *Transitioning to Distributed Sensors*

Our distributed sensor object's (DSO) software architecture, figure 1, is intended to act as an experimental infrastructure to allow exploration into the optimization of data handling, information recognition, and to allow prioritization of information feeds to a central analysis facility while supporting existing IDS techniques.

Using DSO, we choose to perform process elements 1 and 2 on the sensor. This architecture has significant and immediate advantages:

- Creation of feature vectors on the sensor creates content that is optimized for information content; noise, content that is not of interest, having been removed.
- The resulting feature vectors possess the same information, but smaller size, and thus can be transmitted in less time than raw traffic.
- The feature vectors will allow the existing tools and alert generators to function properly without re-design and implementation.
- Sensor-side processing can enable a future architectural change to support 2-phase detection, whereby alerts are generated at the sensor, used as indicators, and sent as priority traffic to the IDS. Confirming data can then be requested only as needed by a new 2-phase analysis and detection tool-set on the central facility.

B. Design of a Distributed Sensor Object

A base DSO type, Figure 2, represents and stores network flows in an efficient, indexed form. Each unified modeling language (UML) object contains contextual information, such as the timeframe of the contained-data, the number of distinct nodes (cardinality), a pre-calculated adjacency matrix, Figure 3, and indices into the adjacency matrix. Cell values of the adjacency matrix reflect the number of flows for each internet protocol (IP) address pair (Source IP, Destination IP), which is referred to here as a *transaction*. Each transaction can be described in more detail using the network flow data. This combination of the adjacency matrix, reflecting the topology of the static network, and attributes for each transaction forms a 3-dimensional structure, referred to here as a transaction hyper-cube (tHc), which is an extension of traditional concepts of adjacency matrices [16].

Using this design model allows detailed data for each transaction to be easily indexed and returned in response to an analyst's query. The practice of using disk-based linear files as the infrastructure's pseudo database containing capture data cannot support the associations and indexing proposed by our model. Thus, this linear file based approach will require a linear, sequential search to generate a response to each analyst query. By forming a pre-indexed structure based upon the ip_address or node_id parameters, we can optimize query requests based upon the parameter of interest. Similar indexing can be applied to sets of port numbers, chronological time sequences, or packet byte counts, all of which are available in a network flow file format. In practice, a storage structure with indexing of ip_addresses is preferred since the ip_address is the most common primary identifier for data retrieval. A chronological sequence of traffic data could also be generated from the tHc to feed certain tools and algorithms. By definition and construction, flow-files are sets of packet data stored in time order, thus no additional processing is required other than locating the appropriate starting and ending packets in the time series.

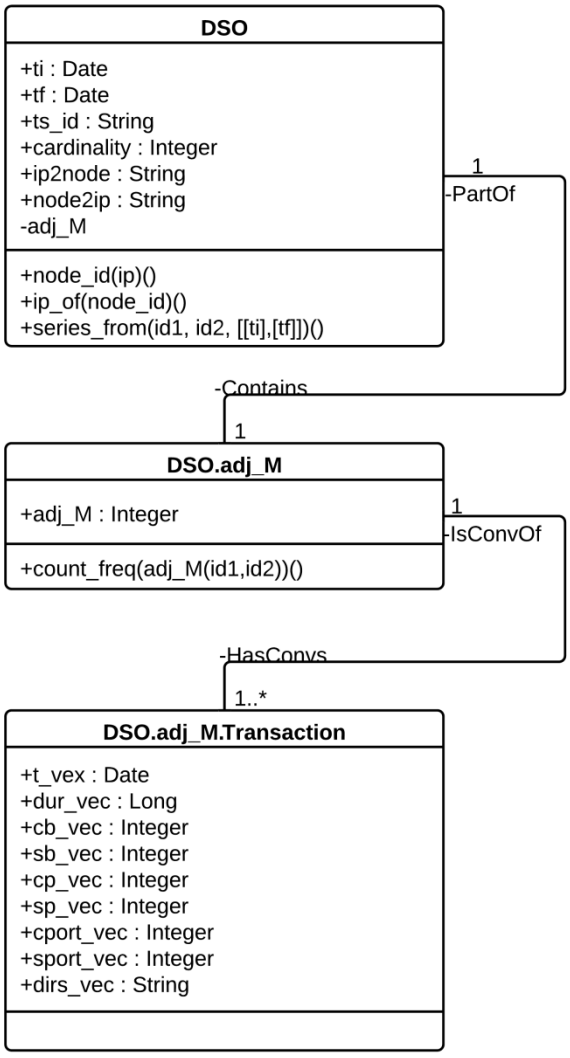

Figure 2: The base DSO type exemplified through UML.

C. DSO Method: Time series request-responder

A representative query and response scenario is as follows:

```
series_from(IP1, IP2, startTime,
endTime) -> sensor X
```

This query requests the time-series data corresponding to transactions between *IP1* and *IP2* during the specified time span; in essence, this query is identifying communication between the two hosts represented by their corresponding IP addresses during the period beginning at *startTime* and ending at *endTime*. A series of tuples will be returned by the DSO on sensor *X* pertaining to these transactions. This form of query is common when an analyst needs to report on the specific activity of an IP address within a known time interval. Other common IDS tools are used to perform set intersection operations, such as comparison of all IPs seen with one or more sets of known IPs, such as whitelists, blacklists, site directories, etc. The most effective use of the tHc data structure for this task is to query the ip2node index for the desired time span and return those parameters to the

IDS tool. The intersection operation would then generate a subsequent (IP1, *) query to obtain all known transactions and attributes pertaining to IP1, for further processing by the tool. In performing this task as a sequence of two steps, we save on unnecessary processing, i.e., unneeded data transfers, and generate result data only when needed by the tool.

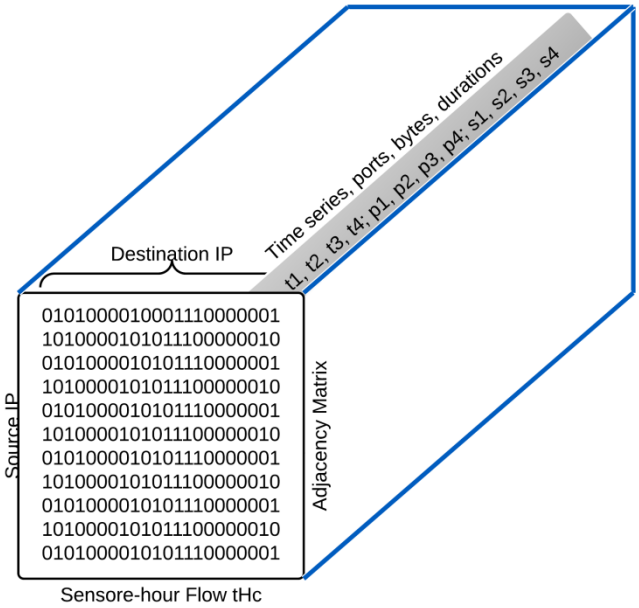

Figure 3: Example adjacency matrix of tHc.

D. DSO - Content Feature Vectors

Advanced IDSs can often process traffic data other than flow files. Modern threats often encode communications in textual content, often within HTTP traffic. Effective detection of these threats depends on using tools that can analyze such traffic since flow files do not contain the content to reveal these communications. We show an extended implementation of DSO, Figure 4, with data and methods to represent such HTTP traffic. Consider the below snort rule taken from bleeding snort [18]. This rule has several features indicative of our concept of feature vectors. First, there must be a human readable string "PRIVMSG", indicated by the content keyword. Second, this text must occur within the first 8 bytes, as indicated by the depth keyword. Third, this text is case insensitive, directed by the nocase keyword. Fourth, the maximum size of the payload must be 128 bytes, as indicated by the dsize keyword.

```
alert tcp $HOME_NET any ->
$EXTERNAL_NET !6661:6668 (msg:
"BLEEDING-SNORT ATTACK RESPONSE IRC -
Private message on non-std port";
flow: to_server,established; dsize:
<128; content:"PRIVMSG "; nocase;
offset: 0; depth: 8; tag:
session,300,seconds; classtype:
trojan-activity; sid: 2000347; rev:5;
)
```

As shown in Figure 4, the basic tHc structure has been extended to include sequences of packet content for each transaction. Furthermore, the parsing of packet content is

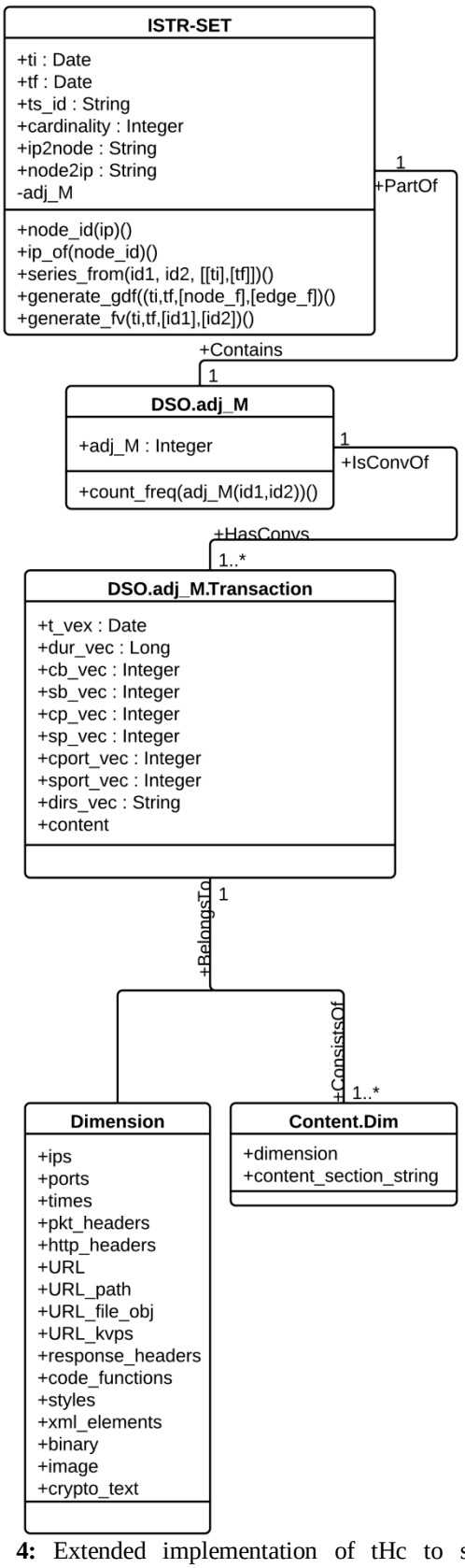

Figure 4: Extended implementation of tHc to support internet string content from associated transaction types, e.g., HTML, ftp, SMTP, etc.

Table 2: Raw data from captured HTTP traffic.

```
0.911310 145.254.160.237 65.208.228.223 HTTP 533 GET /download.html HTTP/1.1

Frame 4: 533 bytes on wire (4264 bits), 533 bytes captured (4264 bits)
Src: 145.254.160.237, Dst: 65.208.228.223
Src Port: tip2 (3372), Dst Port: http (80), Seq: 1, Ack: 1, Len: 479
Hypertext Transfer Protocol

..  ..........E....E@.........A....,.P8....La.P.%..X..GET  /download.html  HTTP/1.1..Host:
www.ethereal.com..User-Agent:  Mozilla/5.0  (Windows;  U;  Windows  NT  5.1;  en-US;  rv:1.6)
Gecko/20040113..Accept:
text/xml,application/xml,application/xhtml+xml,text/html;q=0.9,text/plain;q=0.8,image/png,im
age/jpeg,image/gif;q=0.2,*/*;q=0.1..Accept-Language:        en-us,en;q=0.5..Accept-Encoding:
gzip,deflate..Accept-Charset:  ISO-8859-1,utf-8;q=0.7,*;q=0.7..Keep-Alive:  300..Connection:
keep-alive..Referer:   http://www.ethereal.com/development.html....4.846969   65.208.228.223
145.254.160.237 HTTP 478 HTTP/1.1 200 OK (text/html)

Frame 38: 478 bytes on wire (3824 bits), 478 bytes captured (3824 bits)
Src: 65.208.228.223, Dst: 145.254.160.237
Src Port: http (80), Dst Port: tip2 (3372), Seq: 17941, Ack: 480, Len: 424
Hypertext Transfer Protocol
Line-based text data: text/html

HTTP/1.1 200 OK..Date: Thu, 13 May 2004 10:17:12GMT..Server: Apache..Last-Modified: Tue, 20
Apr  2004  13:17:00  GMT..ETag:  "9a01a-4696-7e354b00"..Accept-Ranges:  bytes..Content-Length:
18070..Keep-Alive:  timeout=15,  max=100..Connection:  Keep-Alive..Content-Type:  text/html;
charset=ISO-8859-1....<?xml  version="1.0"  encoding="UTF-8"?>.<!DOCTYPE  html.    PUBLIC  "-
//W3C//DTD      XHTML      1.0      Strict//EN".            "DTD/xhtml1-strict.dtd">.<html
xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" lang="en">.  <head>.     <title>Ethereal:
Download</title>.      <style  type="text/css"  media="all">..@import  url("mm/css/ethereal-3-
0.css");.      </style>.</head>.   <body>.      <div  class="top">.      <table  width="100%"
cellspacing="0" cellpadding="0" border="0" summary="">.      <tr>.        <tdvalign="middle"
width="1">..   <a  href="/"><img  class="logo"  title="Ethereal  home"  src="mm/image/elogo-64-
trans.gif"  alt=""  width="64"  height="64"></img></a>.</td>.             <td  align="left"
valign="middle">.                <h2>Ethereal</h2>.               <h5  style="white-space:
nowrap;">Download</h5>.        </td>.         <td align="right">..    <table style="margin-
right: 10px;" cellspacing="0" cellpadding="0" border="0" summary="">.
```

driven by a content model consisting of multiple *dimensions* of evidence. The content model shows where within each packet to obtain specific pieces evidence. An alternative to actually storing all of the packet content in the database itself would be to have the structure point to the physical location on disk where the specific packet elements can be found.

Given a query and the evidence locations from the DSO structure, the DSO algorithms creates a tuple of the feature vectors corresponding to the parsed evidence from each packet. The DSO can then stream these feature vectors to the appropriate tools in the IDS to drive primary detection algorithms and confirming evidence collectors. Tools and other detector algorithms can be designed to utilize one or more such dimensions of evidence. Since these dimensions are known a priori, the DSO feature vector generator will parse, represent, and return evidence corresponding to these requested dimensions. All other content can be suppressed since it is the equivalent of noise for the tool and detection processes. This dimension-identification and parsing serves to effect a lossless compression of traffic since the discarded data could not have triggered an alert or a detection in the tool processes.

A query request to generate feature vectors from internet strings corresponding to a list of specified dimensions will appear as follows:

```
generate_fv(
      startTime,
      endTime,
      {IP,PORT,PROTO,TIME,REFERER,
      USERAGENT,URL,HOST}
      ) -> sensor X
```

E. DSO - Extraction of Selected Dimensions of Evidence

Table 2 illustrates an excerpt of HTTP capture data in plain text format, which retains all raw data from the capture (http.cap from

http://wiki.wireshark.org/SampleCaptures).

As mentioned, the goal will be for the analyst to a priori create a list of desired dimensions to be used to create feature vectors from the raw traffic. The goal will be for these dimensions to accurately retain the information that can be used by detector and tool processes. An example of the

Table 3: Select dimensions and associated feature vectors from the raw HTTP data in table 2. These feature vectors represent the content usable by IDS tools and detectors; approximately a 50% reduction in data load is exhibited.

Dimension	Sample Values
IP / PORT / TIME / PROTO	`0.911310 145.254.160.237:3372 65.208.228.223:80 HTTP 533`
URL	`GET /download.html HTTP/1.1`
Referer	`Referer: http://www.ethereal.com/development.html`
User-Agent	`User-Agent: Mozilla/5.0 (Windows; U; Windows NT 5.1; en-US; rv:1.6) Gecko/20040113`
Host	`Host: www.ethereal.com`
	`4.846969 65.208.228.223:80 145.254.160.237:3372 HTTP 478`
Content	`charset=ISO-8859-1....<?xml version="1.0" encoding="UTF-8"?>.<!DOCTYPE html. PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN". "DTD/xhtml1-strict.dtd">.<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" lang="en">. <head>. <title>Ethereal: Download</title>. <style type="text/css" media="all">..@import url("mm/css/ethereal-3-0.css");. </style>.</head>. <body>. <div class="top">. <table width="100%" cellspacing="0" cellpadding="0" border="0" summary="">. <tr>. <tdvalign="middle"`

feature vectors parsed from the raw data in table 2 is shown in table 3.

The novel feature vector concept provides several other advantages. First, the feature vectors allow for the automatic aggregation of data from multiple sources or tools. By having the analyst identify the feature vectors a priori for each data source or tool result, the analyst in essence is identifying a schema or semantic organization of the data such that it can be aggregated based on consistent features across data sources and tool results. Second, the semantic content provided by the feature vectors provides a more meaningful organization for analysis and storage in databases. Third, the feature vectors can potentially inspire additional analysis paradigms, given the large body of research on semantic analysis [3].

F. tHc Data-Structure

The source code listing in table 4 is a reference implementation of the tHc data-store as a Perl Storable object. The anonymized excerpt shows the overall organization of instance metadata, forward and reverse indices, and node flows with corresponding time series attributes. These tHc structures are created in memory for rapid access by the DSO to support queries and generation of feature-vectors. The Perl Storable object methods provide functions to save the data-structure to disk, and to restore such files again as in-memory data-stores.

G. Validation

To validate the accuracy and completeness of tHc for flow data, we generated tHc data structures from representative flow files. Typical results of the conversion result in the following characteristics:

- Raw data flow file size 2.1MB
- Raw number nodes 1527
- tHc storage object size 1.6MB
- tHc number nodes 1527

The information-to-noise ratio for a flow file is approximately 1:1 since each data element is potentially of interest to the IDS processing. The size difference exemplified by this flow file is due to different representations of data, time, and numbers in packed versus string representations. The two representations incorporate identical node counts indicating that no information has been lost in the conversion; the equivalent information has been structured in memory to optimize DSO query requests.

To validate the feature-vector generation process for internet string traffic, the DSO must parse the raw traffic to preserve only the evidence dimensions requested. As a result, the volume of data of the feature-vectors will be less than the volume of the raw data. It is important that the feature-vectors contain equivalent information to support the tool and detection processes. We show that while the volume of data is decreased, the resulting tool/alert generation is

Table 4: Sample Implementation of tHc as an associative memory array through a Perl Storable object generated from CISCO NetFlow (v5) traffic.

SVAR1 = {
'tHc_version' => 3,
'flows' => {
'A.B.C.92' => {
'A.B.C.221' => {
'srcport'=> '34,',
'prot' => 'TCP(0x06),',
…
'A.B.C.71' => {
'A.B.C.333' => {
'srcport'=> '8,8,',
'prot' => 'IPv6/TCP(0x06),IPv6/TCP(0x06),',
'tcp_flags' => '****APRS*,****APRS*,',
'AdjMat_coords' => {
'y' => 102,
'x' => 122
},
'svr_bytes' => '5362,5362,',
'clt_pkts' => '12,12,',
'cli_bytes' => '2704,2704,',
'srcaddr' => 'A.B.C.71',
'durations' => '00:01:25.6727854,00:01:26.968210,',
'dstaddr' => 'A.B.C.333',
	Individual Flows and Times

'metadata' => { 'nodecounts' => 1521, 'storable' => '20110712.11.xxx.storable', 'lastTime' => 1310471999, 'firstTime' => 1310468402, 'srcfile' => '20110712.11.xxx', 'edgecount' => 5108 },	**Context Data for the Instance**

'indices' => {
'indx2ip' => {
'1049' => 'A.B.C.90',
'127' => 'A.B.C.220',
'71' => 'A.B.C.143',
'1481' => 'A.B.C.61',
'882' => 'A.B.C.37',
	INDX to IP And IP to INDX

statistically equivalent. In practice, the feature-vector set requires only 20% of the volume to produce a slight increase in alert count. This +0.3% discrepancy is due to differences in the alert generation algorithms; in the feature-vector case, multiple alerts can be generated from the appearance of multiple string literals in the same packet.

- # of alerts generated using raw data 47746
- # of alerts generated using tHc 47889
- Volume of alert data using raw data 43MB
- Volume of alert data using tHc 8MB

IV. Future Work

One task for future work will be to explore the tradeoffs inherent in allowing the transmission of distributed data collection to a central server more fully. For instance, as sensor capabilities improve, more computation can be offloaded to the sensors. However, this has certain limitations such as the impact of a compromised sensor or network linkage. There is also the potential for having more than two-phases in the architecture, such as: collection and partial processing at the sensor, aggregation and final processing on a central server, and presentation on a mobile device. We must identify the complete performance requirements and capability distribution techniques for maximum performance given a specific network infrastructure and associated high bandwidth network. Allowing this distribution to be determined and adjusted dynamically is the ultimate goal.

V. Conclusions

We designed and implemented a novel IDS architecture. This resulted in the development of a distributed sensor

object that incorporates features in the sensor data collection process that optimizes the overall IDS process. This developed sensor object satisfies the following objectives:

- Collect all data, dependent of course on the storage capacity of the sensor
- Parse raw network traffic to extract only the specified evidence dimensions
- Generate lossless feature-vectors sufficiently representing the original raw network data to drive the IDS detection algorithms
- Create a query-optimized local data store for the raw data
- Support a query responder to select and return precise feature-vectors on request
- Support a feature-vector stream to feed traffic to new and legacy IDS tools, removing non-interesting evidence, e.g., noise
- Support an extensible data-set generator, as would be needed to generate a network graph for data visualization at the client
- Support an object-oriented interface API, implementable in any modern language environment, including Perl, Python, C++, Java, etc.

In addition, we provided examples taken directly from the reference implementation. This reference implementation was used to provide validation, showing that the proposed sensor design not only reduces the amount of data needing to be transmitted to the central server but also maintains a statistically similar number of generated alerts. A secondary effect resulted in that the volume of the generated alert output also occupies significantly less data space.

REFERENCES

[1] P. Barford, S. Jha, and V. Yegneswaran, "Fusion and filtering in distributed intrusion detection systems," In *Proceedings of the 42nd Annual Allerton Conference on Communication, Control and Computing*, 2004.

[2] Mohamad Eid, Hassan Artail, Ayman Kayssi, and Ali Chehab, "A New Mobile Agent-Based Intrusion Detection System Using Distributed Sensors," in *Proceedings of the IEEE International Conference on Pervasive Services* (ICPS'2004), March 2004, Beirut, Lebanon, pp. 114-125.

[3] Cliff Goddard, *Semantic Analysis: A Practical Introduction*, Oxford University Press, USA, 2011.

[4] M. Jahnke, 2002, "An Open and Secure Infrastructure for Distributed Intrusion Detection Sensors," *Proceedings of the NATO Regional Conference on Military Communications and Information Systems* RCMCIS 2002, 9-11 October 2002. [19] P. Baskerville, 2006.

[5] J. Kim and P. Bentley, "Towards an artificial immune system for network intrusion detection: An investigation of dynamic clonal selection," In *Proceeding of the Congress on Evolutionary Computation* (CEC-2002), Honolulu, Hawaii, pp. 1015 - 1020, May 2002.

[6] J. Kim and P. J Bentley, "Towards an artificial immune system for network intrusion detection: An investigation of clonal selection with a negative selection operator," In *Proceeding of the Congress on Evolutionary Computation* (CEC-2001), Seoul, Korea, pp. 1244-1252, 2001.

[7] Maciá, -Pé, F. rez, F. Mora-Gimeno, D. Marcos-Jorquera, Gil-Martí, J.A. nez-Abarca, H. Ramos-Morillo, I. Lorenzo-Fonseca, "Network Intrusion Detection System Embedded on a Smart Sensor," *IEEE Trans. on Industrial Electronics*, vol. 58, no. 3, pp. 722 - 732, March 2011.

[8] Mohammad Mamun and A.F.M. Kabir, "Hierarchical Design Based Intrusion Detection System for Wireless Ad Hoc Sensor Networks," *International Journal of Network Security & Its Applications* (IJNSA), Vol.2, No.3, July 2010, pp. 102-117.

[9] John C. McEachen and Cheng Wai Kah. 2007, "An analysis of distributed sensor data aggregation for network intrusion detection," *Microprocess. Microsyst.* 31, 4 (June 2007), 263-272.

[10] John C. McEachen, Cheng Kah Wai, and Vonda L. Olsavsky. 2006, "Aggregating Distributed Sensor Data for Network Intrusion Detection,". In *Proceedings of the 11th IEEE Symposium on Computers and Communications* (ISCC '06). IEEE Computer Society, Washington, DC, USA, 916-922.

[11] T. R. Metcalf and L. J. Lapadula, "Intrusion Detection System Requirements: A Capabilities Description in Terms of the Network Monitoring and Assessment Module of CSAP21," Mitre Paper # MP00B0000046, September 2000.

[12] B. Quanz, H. Fei, J. Huan, J.B. Evans, V. Frost, G.J. Minden, D.D. Deavours, L. Searl, D. DePardo, M. Kuehnhausen, D. Fokum, M. Zeets, and A. Oguna, "Anomaly Detection with Sensor Data for Distributed Security," in *Proc. ICCCN*, 2009, pp.1-6.

[13] Chakchai So-In, "A Survey of Network Traffic Monitoring and Analysis Tools," http://www.cse.wustl.edu/~jain/cse567-06/ftp/net_traffic_monitors3/

[14] Spoor, Rogier, "A Distributed Intrusion Detection System Based on Passive Sensors," *Surfnet*, 11 Aug. 2005. 3 Apr. 2006.

[15] A.W. Stroupe, M.C. Martin, and T.R. Balch, "Distributed Sensor Fusion for Object Position Estimation by Multi-Robot Systems," in *Proc. ICRA*, 2001, pp.1092-1098.

[16] J. Sun, Y. Xie, H. Zhang, and C. Faloutsos, "Less is More: Compact Matrix Decomposition for Large Sparse Graphs," Carnegie Mellon University, Computer Science Department, 2007, Paper 532.

[17] Jennifer Yick, Biswanath Mukherjee, and Dipak Ghosal, 2008, "Wireless sensor network survey," *Comput. Netw.* 52, 12 (August 2008), 2292-2330.

[18] Bleeding Snort, http://www.bleedingsnort.com/