



US 20060021021A1

(19) **United States**

(12) **Patent Application Publication**  
**Patel**

(10) **Pub. No.: US 2006/0021021 A1**

(43) **Pub. Date: Jan. 26, 2006**

(54) **SECURITY EVENT DATA NORMALIZATION**

**Publication Classification**

(76) Inventor: **Rajesh Patel**, Los Altos, CA (US)

(51) **Int. Cl.**  
**G06F 15/16** (2006.01)

Correspondence Address:  
**GREENBLUM & BERNSTEIN, P.L.C.**  
**1950 ROLAND CLARKE PLACE**  
**RESTON, VA 20191 (US)**

(52) **U.S. Cl.** ..... **726/13**

(57) **ABSTRACT**

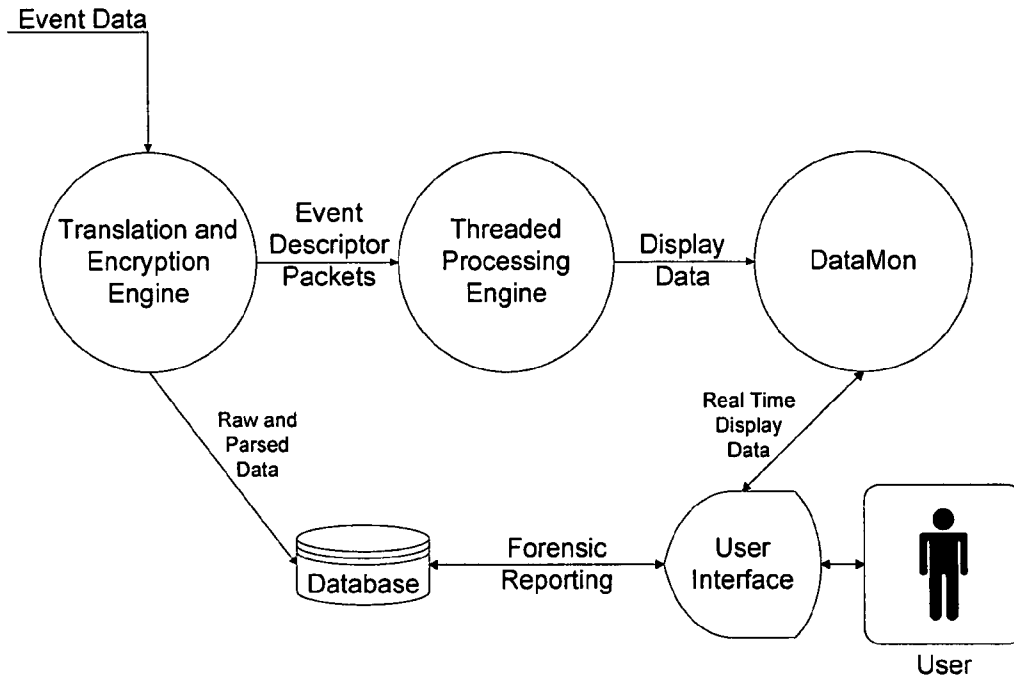
(21) Appl. No.: **11/149,083**

(22) Filed: **Jun. 8, 2005**

**Related U.S. Application Data**

(60) Provisional application No. 60/578,281, filed on Jun. 8, 2004.

Normalizing security event data from multiple different network agents. The data from the multiple different agents is categorized and tagged with a descriptor that includes information about the nature of the event. Multiple different events from multiple different devices can therefore be evaluated using a common format which is common for the multiple different devices from different vendors.



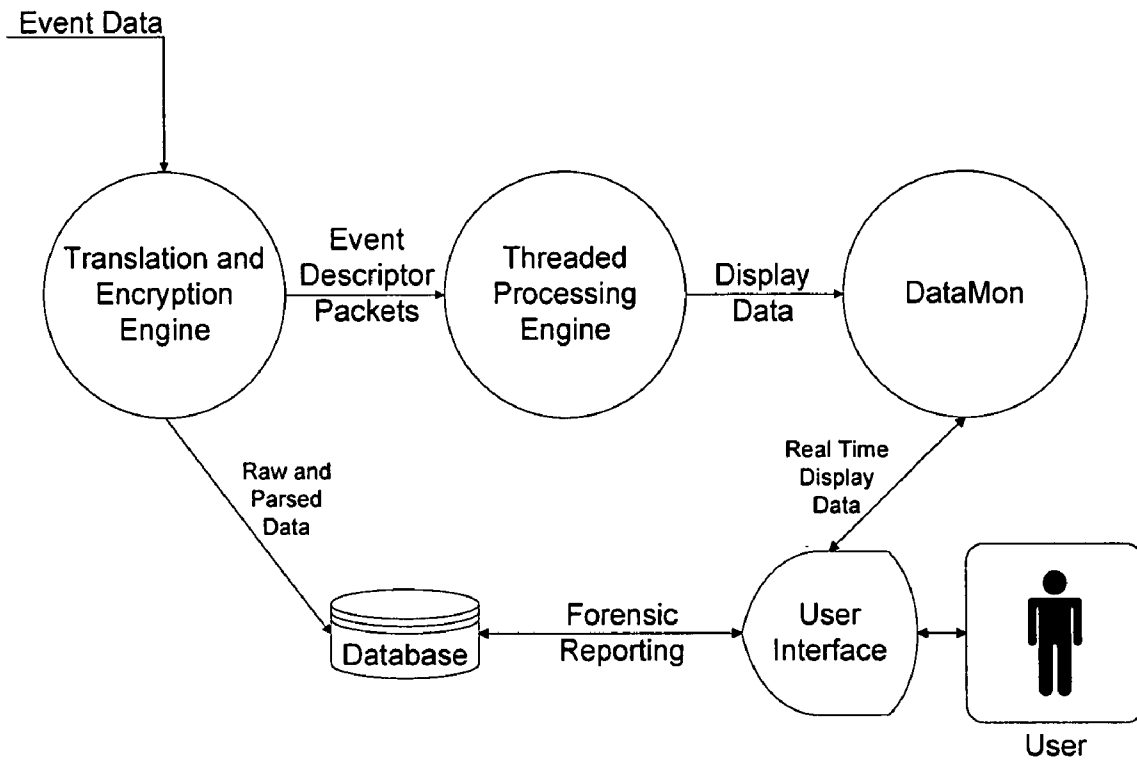


Figure 1. TVS Functional Areas

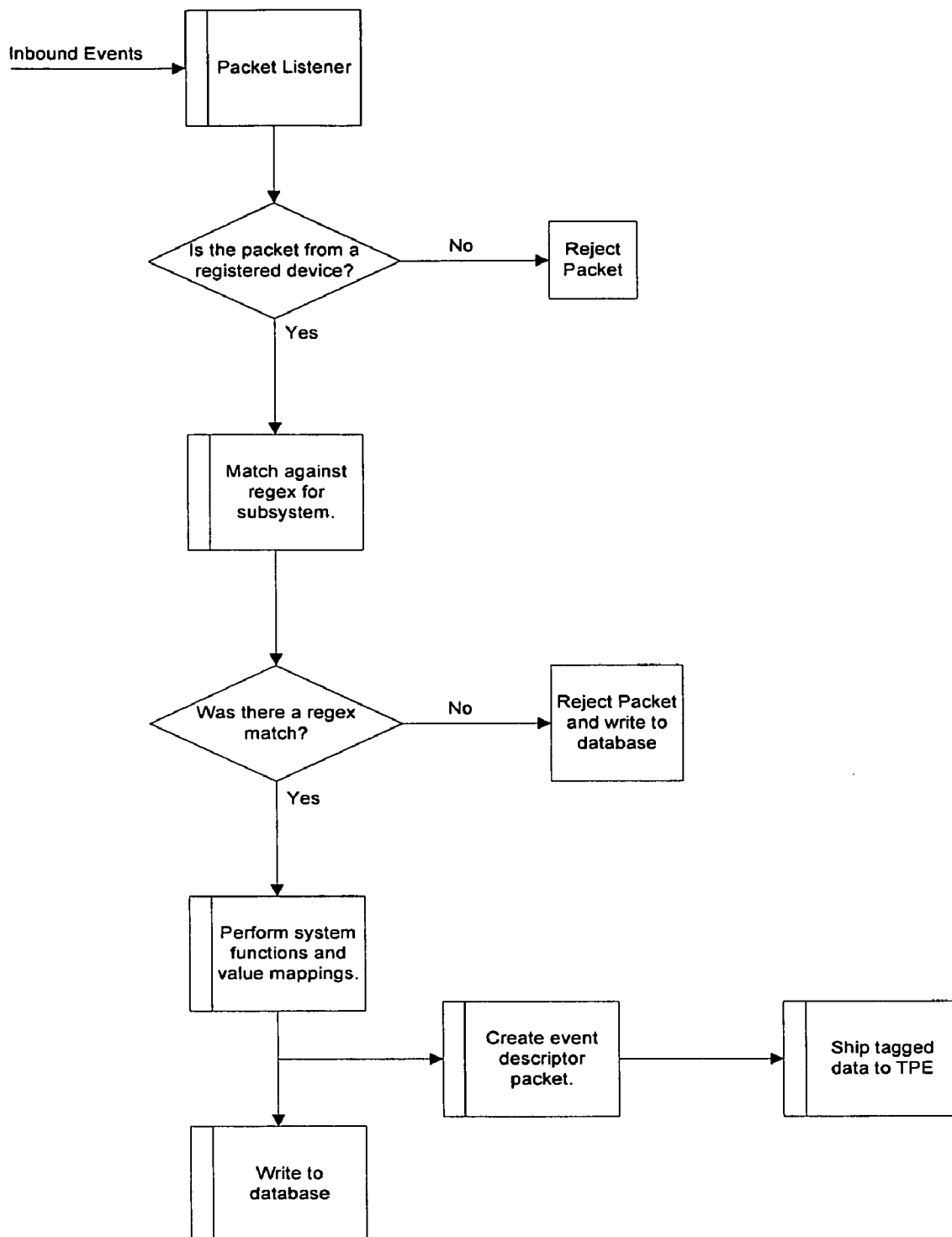
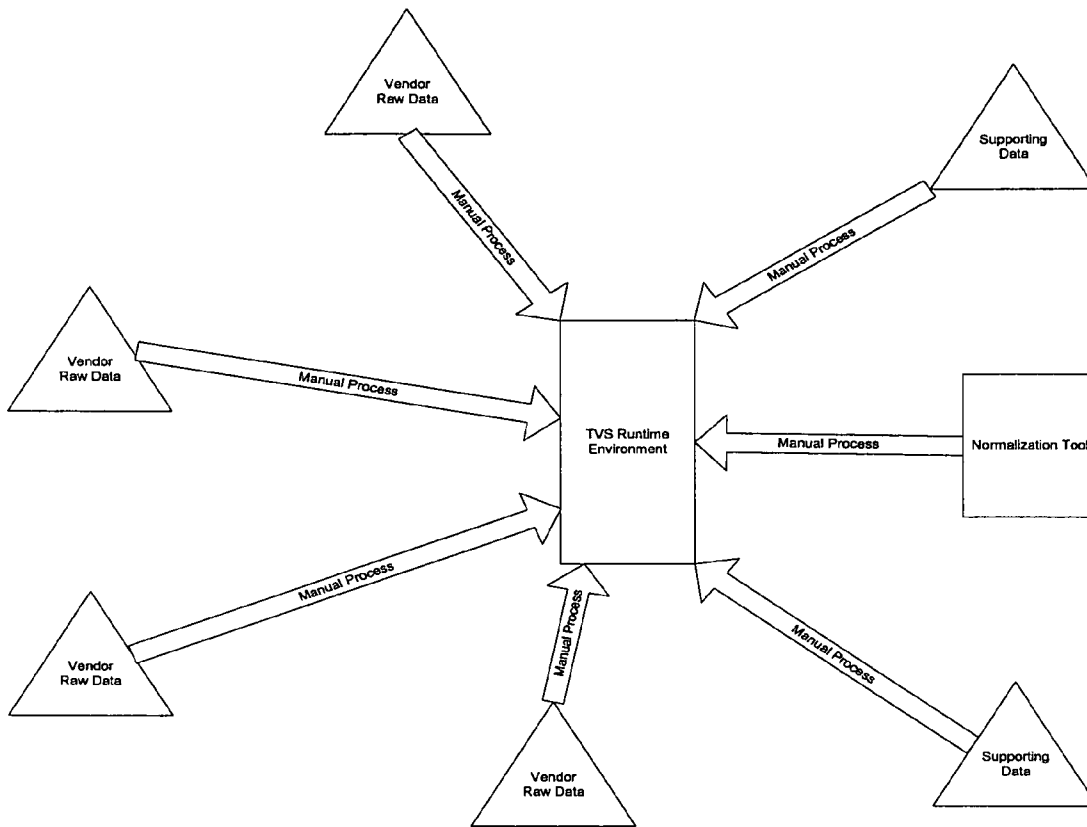


Figure 2. TEE Logical Flow

ht sig cat	ht device cat	ht event cat	ht event id
0000	00000	0000000000	00000000000000

Figure 3. TVS Event Tag



**Figure 4. Initial state of TVS data requirements support.**

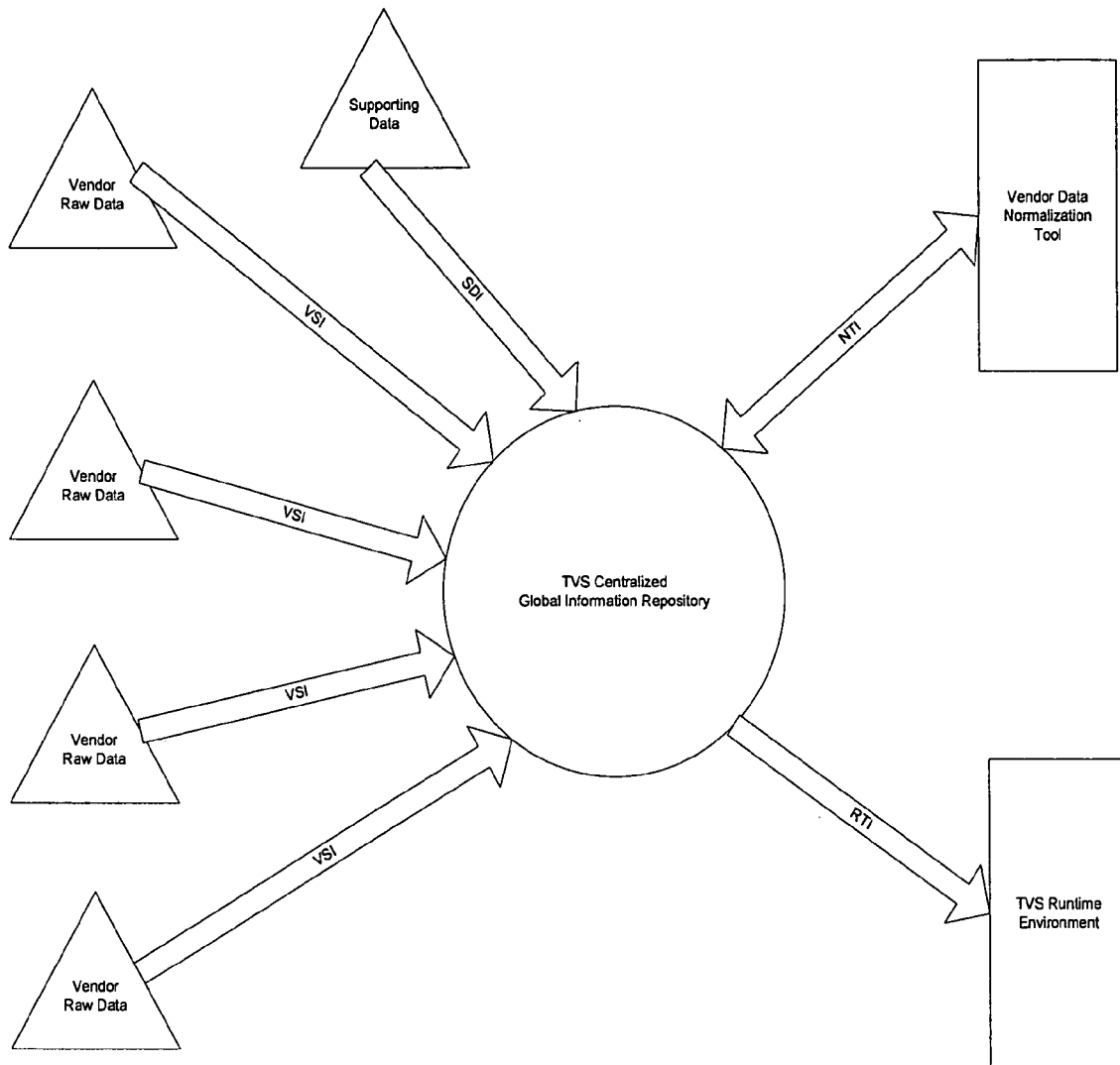


Figure 5. Example of a Centralized Global Information Repository.

/opt/towerview/demo/vm/

default-system.vm	#value map registration point
'subsystem-name1.vm"	#registered subsystem signature id value map
'subsystem-name2.vm"	#registered subsystem signature id value map
~etc.	

/usr/local/mysql/var/tv\_data

vendors	#information about vendor references
agent_refs	#information about vendor references
~etc.	

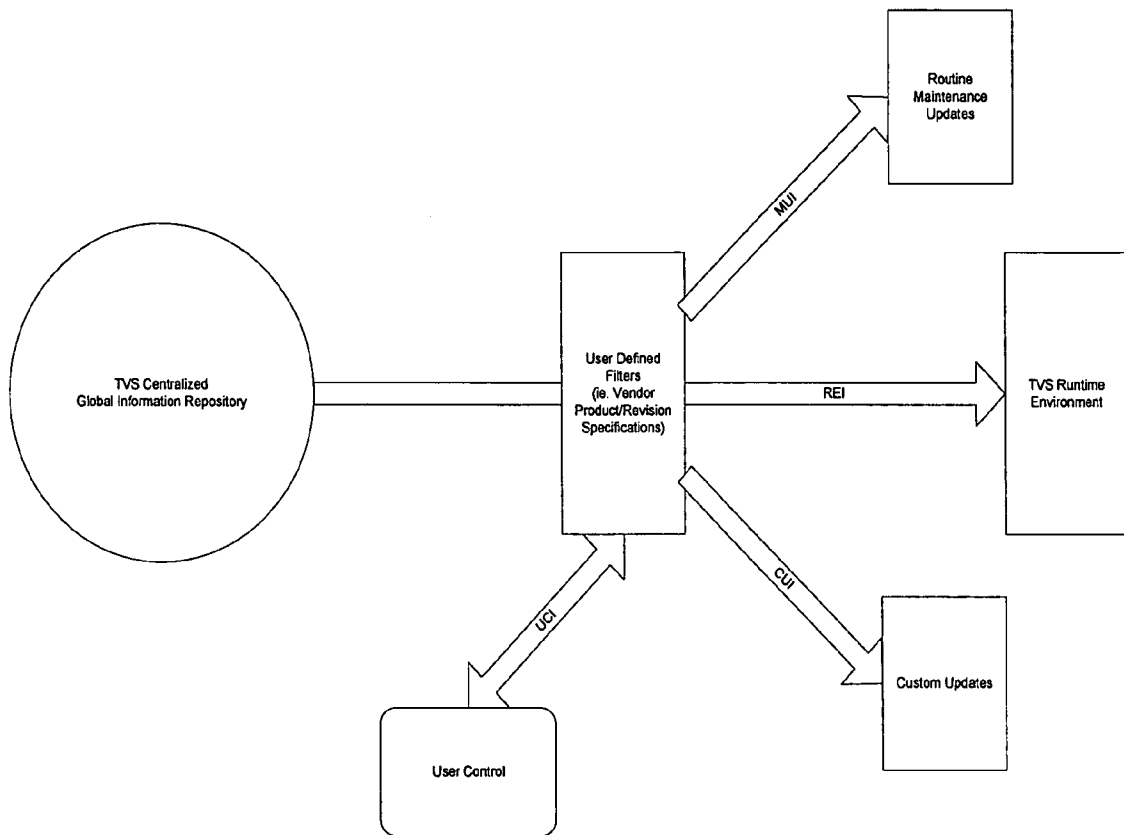
/opt/towerview/demo

config.xml	#information about vendor devices
~etc.	

/opt/towerview/demo/ruletemplates

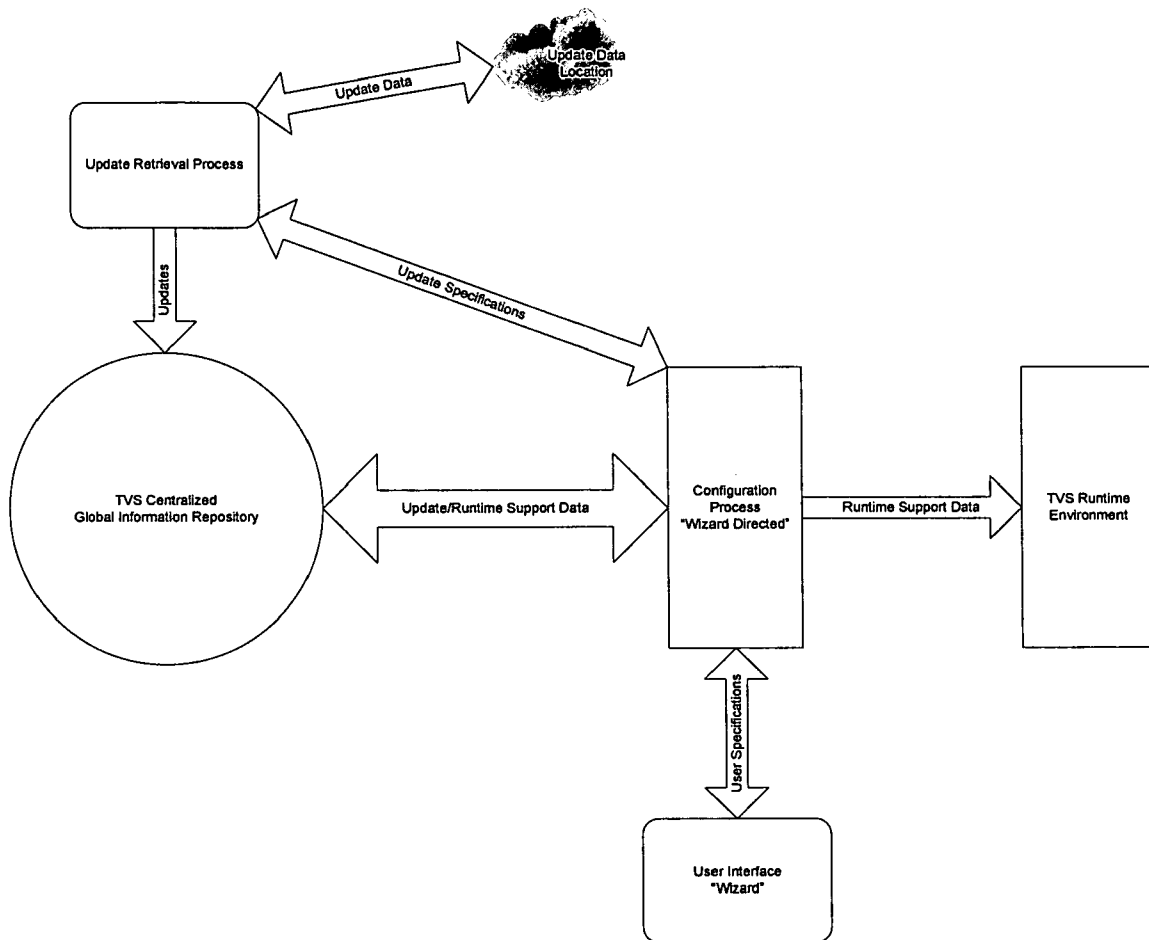
valuemappings	#ht_id to message value mappings for use in the DM
~etc.	

**Figure 6. TVS Runtime Requirements.**



**Figure 7. TVS Runtime Requirements Delivery Automation.**





**Figure 8. TVS Update Process.**

**SECURITY EVENT DATA NORMALIZATION**

[0001] This application claims priority from Application No. 60/578,281, filed Jun. 8, 2004, the disclosure of which is herewith incorporated by reference.

**BACKGROUND**

[0002] In the world of security event management, every one is speaking a different language. This stems from the fact that there is no industry wide standard language used to describe security events. Therefore, each vendor provides notification of detected events in their own proprietary format.

**SUMMARY**

[0003] In order to accommodate the disparity in vendor event notification formats, HighTower Software has developed a method of normalizing events from each vendor's proprietary format into a common format. By development of a categorization scheme and associated tagging process, each event that enters the normalizing application is evaluated and tagged with a descriptor that carries specific information about the nature of the event. This technique allows events to be evaluated in a common context, while retaining the vendor specific context.

**BRIEF DESCRIPTION OF THE DRAWINGS**

[0004] These and other aspects will now be described in detail with reference the accompanying drawings, wherein:

[0005] **FIG. 1** shows a block diagram of the overall system operation;

[0006] **FIG. 2** illustrates the logical information flow within the translation and encryption engine;

[0007] **FIG. 3** illustrates an exemplary event tag;

[0008] **FIG. 4** illustrates the normalization runtime environment and the data that it receives;

[0009] **FIG. 5** illustrates the information registry receiving the data from various items;

[0010] **FIG. 6** illustrates the runtime requirements of the system;

[0011] **FIG. 7** illustrates the automation of delivery of these runtime requirements; and

[0012] **FIG. 8** illustrates the update procedure that allows for application maintenance.

**DETAILED DESCRIPTION**

[0013] **FIG. 1** shows a block diagram with three discreet functional parts, including the Translation and Encryption Engine (TEE) **100**, the Threaded Processing Engine (TPE) **110** and the Data Monitor (Datamon or DM) **120**.

[0014] The TEE is responsible for the validation, initial processing, formatting, database storage, and the forwarding to the TPE of data received from devices that are monitored by TVS. The TPE is responsible for performing analysis, according to user specified configuration directives, in the form of rules assignment, of the event data it receives from the TEE. Once analysis has been performed, the results of that analysis are sent to the Datamon as display updates. The Datamon is responsible for providing the TVS user with 3D

graphical display of the analysis performed by the TPE on events that have been received and processed by the TEE. Additionally a web based portal may be available to provide the TVS user with an environment in which to generate forensic analysis reports from the data contained in the TVS runtime database.

[0015] The event data normalization process is implemented as part of the TEE. Devices which are to be monitored by TVS are registered with the TEE. As part of this registration process, they are assigned a subsystem type. This subsystem type defines the set of rules by which these events are to be processed. As events from registered devices are received by the TEE, they are broken down into their elemental components. These elemental components include such items as device IP Address, Packet Source IP Address, Packet Source Port, Packet Destination IP Address, Packet Destination Port, Signature Identifier and a number of other items of data that are specific to a particular subsystem type. From this set of elemental components the Signature Identifier is used to match against a predetermined list of relationships or value mappings that serve to assign a tag to the event. This tag is carried with the packet of elemental components or event descriptor packet that is used to process the event within the TVS application.

[0016] TVS provides support for many device categories and many device variants within each category. Within a specific device category message format and content may differ based on each vendor's specific message format, message content and message or signature identifier. Different categories of devices may provide messages containing information about an event that is also presented by another category of device about the same event. The format and content of these different messages about the same event will likely differ. TVS provides a methodology first for normalization of data provided by disparate devices within each category and secondly for normalization of data about the same event by devices in different categories.

**TEE Functionality**

[0017] The Translation and Encryption Engine (TEE) **100** includes a software code that receives, validates, transforms, normalizes, buffers and sends data between one or more components in the TowerView Security system. The TEE's primary purpose is to transform the data received from a network agent, such as a firewall, nids, or hids, and convert it into a common format which can be understood by other components in the system. Then this processed data is forwarded to one of the many connected targets, such as the processing engine (PE) or the data monitor (DM). The TEE's secondary purpose is to store the raw and translated data to a database repository for forensic analysis. A high level logical flow of the TEE is depicted in **FIG. 2**.

**Tags**

[0018] In order to provide state-full normalization of data to the processing engine we assign a tag to each event that is processed. This tag contains meta information about the event that will be used by the rule set in the TPE.

**Tag Structure**

[0019] The Hightower Security Event TAG is a 32 binary bit segmented data entity. Each segment provides specific

information about the event it tags. The currently defined structure of the ht\_event\_tag is shown in FIG. 3.

ht\_sig\_cat:

[0020] To support normalization of event data between categories of devices we require broad classification of events. This reflects the severity of the event in the context of network security. These broad classifications have currently been defined as Uncategorized, Unknown, Normal, Reconnaissance, Malicious, Compromised and Health/Status. These classifications are reflected in a four-position binary tag field labeled ht\_sig\_cat. The current mapping for the values of the ht\_sig\_cat tag field are shown below in Table 1.

TABLE 1

ht_sig_cat key	
ht_sig_cat	Category
1	Uncategorized
2	Unknown
3	Normal
4	Reconnaissance
5	Malicious
6	Compromised
7	Health/Status

[0021] The ht\_sig\_cat tag describes at a high level the nature of the event being evaluated.

ht\_device\_cat:

[0022] To support normalization of event data provided by like devices, perhaps from different vendors, with different event data formats we are required to defined device categories. These classifications are reflected in a five-position binary tag field labeled ht\_device\_cat. The current mapping for the values of the ht\_device\_cat tag field are shown in Table 2.

TABLE 2

ht_device_cat key	
ht_device_cat	Category
1	Firewall
2	Intrusion Detection System
3	Router
4	Vulnerability Assessment
5	VPN
6	Tbd
7	Tbd

[0023] The ht\_device\_cat tag describes the type of device has sent the data about the event being evaluated.

ht\_event\_cat:

[0024] Table 2 shows five device categories. The events that may be a device in a particular category, for example an Intrusion Detection System, we will see that these events also fall into a specific number of categories. These event categories are largely common amongst different IDS implementations. To provide normalization of events from different implementations of like devices we are required to

provide classification of events by event category. These classifications are reflected in a ten-position binary tag field labeled ht\_event\_cat. For example for an Intrusion Detection System mappings for the values of the ht\_event\_cat field are shown below in Table 3.

TABLE 3

ht_event_cat key for IDS	
ht_event_cat	Category
101	Attack Response
102	Backdoor
103	Bad Traffic
104	Chat
105	DDOS
107	DNS
108	DOS
109	Exploit
110	Finger
111	FTP
112	ICMP-Info
113	ICMP
114	IMAP
115	Info
116	Local
117	Misc
118	Multimedia
119	Mysql
120	Netbios
121	Nntp
122	Oracle
123	Other-IDS
124	P2p
125	Policy
126	Pop2
127	Pop3
128	Porn
129	Rpc
130	Rservices
131	Scan
132	Shellcode
133	Smtip
134	Snmp
135	Sql
136	Telnet
137	Tftp
138	Virus
139	Web-attacks
140	Web-cgi
141	Web-client
142	Web-coldfusion
143	Web-frontpage
144	Web-iis
145	Web-misc
146	Web-php
147	X11
148	Packet Discarded

ht\_event\_id:

[0025] Within each ht\_event\_cat we have allowed for up to 8192 unique event ids. These classifications are reflected in a thirteen-position binary tag field labeled ht\_event\_id.

Application of the ht Event Tag

[0026] For example consider an event reported by the following snort NIDS rule:

```
[0027] alert tcp $EXTERNAL_NET any->$DNS_SERVERS
53~"thisissometempspaceforthesockinaddrinyeahyehahiknowthisisla mebutanywaywho-
careshorizontogotitworkingsoalliscool"
```

[0028] If this rule triggers it indicates that a DNS compromise has occurred. This vulnerability is described in

CVE-1999-0833, BUGTRAQ ID 788 and CERT CA-1999-14. This event will be assigned a ht\_event\_tag of 610D400C (hex) or 1628258316 (decimal). From the binary perspective that is ht\_sig\_cat 6 (compromised), ht\_device\_cat 2 (IDS), ht\_event\_cat 107 (DNS) and ht\_event\_id 12 (ADM DNS overflow).

**[0029]** Consider the following Dragon NIDS rule: SIGNATURE=T D A B 10 0 53 DNS:ADM-OVERFLOW thisissometempspaceforthesockinaddr

**[0030]** This rule is written to detect the same event as the snort rule above. Detection of these events by disparate IDS systems will produce an identical ht\_event\_tag. This event will be assigned a ht\_event\_tag of 610D400C (hex) or 1628258316 (decimal). From the binary perspective that is ht\_sig\_cat 6 (compromised), ht\_device\_cat 2 (IDS), ht\_event\_cat 107 (DNS) and ht\_event\_id 12 (ADM DNS overflow).

**[0031]** Note: If this event were to be detected by the Cisco PIX Firewall IDS feature, the field ht\_device\_cat in the ht\_event\_tag would reflect a device type of 1 and the resulting ht\_event\_tag would be 608D400C (hex) or 1619869708 (decimal). From the binary perspective that is ht\_sig\_cat 6 (compromised), ht\_device\_cat 1 (FW), ht\_event\_cat 107 (DNS) and ht\_event\_id 12 (ADM DNS overflow).

**[0032]** In the present system, with snort and Dragon, the ht\_event\_tag is identical. Events detected from different points in your network and by devices from different vendors can be correlated and identified as identical events. This enables evaluation of the effectiveness of a network's defense in depth by showing the progress or lack of progress of hostile events across the layers of your network. In the context of TVS, device chains can be identified, which are used to notify operators of the detection of identical events by devices at multiple points in the network.

#### Vulnerability Assessment Correlation

**[0033]** Consider the case of a network that employs use of a Vulnerability Assessment (VA) system. That is a system that by varying methods and vary degrees of intrusiveness examines the nodes on a given network segment for their susceptibility to a set of vulnerabilities known by the vendor of the VA system. VA systems generally produce a set of results of the analysis of networks they scan. Network and System Administrators have traditionally used this data to guide them as to which systems require patching for the various vulnerabilities that have been discovered.

**[0034]** Consider the Foundstone VA tool which runs a series of scripts that are used to detect vulnerabilities in nodes on a given network segment. One of the scripts that the Foundstone VA tool runs is unix-chargen-dos-v2.fasl3. This script is used to detect unix systems that are vulnerable to chargen dos exploits. The chargen service generates random characters and this service can be used in conjunction with other services to cause a denial of service attack via flooding.

**[0035]** The following snort NIDS rule is used to detect instances of this exploit on the networks it monitors.

**[0036]** alert udp any 19 < > any 7 (msg: "DOS UDP echo+chargen bomb"; reference: cve, CAN-1999-0635; reference: cve, CVE-1999-0103; classtype: attempted-dos; sid: 271; rev: 3;)

**[0037]** In order to eliminate false alarms and unnecessary response to events to which we are not vulnerable and to enhance our sense of urgency and highlight those events to which we are vulnerable we can use capabilities enabled by the TVS normalization process to determine if there is a correlation between NIDS detected hostile events and the network vulnerabilities reported by our VA tool.

**[0038]** In order to provide for VA correlation, TVS builds a table which reflects the state of node vulnerabilities on the monitored network segment, based on data provided by the VA tool deployed in the network. To build that table TVS must acquire and process the VA data from the VA tool. The particular method of data acquisition from the VA tool is vendor dependent.

#### Acquisition of VA Data

**[0039]** In order to support VA correlation we acquire the following elements of data from the VA tool results set. Our query, which is performed by the script getvadata.pl, populates three files. These files are va\_vulns.dat, va\_os\_scandate.dat, and va\_ports\_open.dat. The file va\_vulns.dat contains three data items; they are ipaddr\_converted, ht\_event\_tag, and va\_sid. The file va\_os\_scandate.dat contains three items of data; they are ipaddr\_converted, OS, and va\_scandate. The file va\_ports\_open contains three items of data; they are ipaddr\_converted, protocol and port.

#### File Use and Population

va\_vulns.dat:

**[0040]** This file, which contains a listing the vulnerabilities that have been detected by a Vulnerability Scan, will document the IP address of the vulnerable device and the relationship between ht\_tag and VA tools sid for the event. The script getvadata.pl will query the VA tool results set and select the IP address of the vulnerable network node and the sid. The sid is used as an index to assign an ht\_event\_tag from the VA devices value-map file. The VA devices value-map file documents the relationship between the VA devices sid and the associated ht\_event\_tag.

**[0041]** The data item ipaddr\_converted will be constructed from the item in the VA tool results set that represents the IP address of the vulnerable device. The data item ht\_event\_tag is determined by using the VA tools event sid to index the value-map file, retrieving the appropriate value. The data item va\_sid is retrieved directly from the VA tool results set.

va\_os\_scandate:

**[0042]** This file, contains scan dates and Operating Systems (OSs), and documents the IP address of the vulnerable device, its Operating System and the date that the last VA scan was performed.

**[0043]** The data item ipaddr\_converted is constructed from the item in the VA tool results set that represents the IP address of the vulnerable device. For each IP address we will query the VA tool results set for the OSName. For each IP address we will query the VA tool results set for the date/time of the VA scan. The file va\_os\_scandate will be populated with ipaddr\_converted, OS and va\_scandate.

va\_ports\_open:

**[0044]** This file, contains a list of active services, and documents the IP address of the device as well as the protocol and port of any active services.

[0045] The data item ipaddr\_converted is constructed from the item in the VA tool results set that represents the IP address of the vulnerable device. The data item protocol is constructed from the item in the VA tool results set that represents the protocol used to access the vulnerability on the vulnerable device. The data item port is constructed from the item in the VA tool results set that represents the port used to access the vulnerability on the vulnerable device. The file va\_ports\_open will be populated with ipaddr\_converted, protocol and port.

VA Correlation

[0046] Once the VA files have been populated and the runtime table that reflects the state of the VA assessment has been built within TVS, analytics are used monitor events which may exploit particular vulnerabilities. If an event is evaluated and found to be an exploit destined for a node that has a documented vulnerability for that exploit an alarm indicating this can be issued to the TVS operator.

[0047] Consider the previous example of the dos+chargen bomb. The snort rule that detects this event, alert udp any 19 < > any 7 (msg: "DOS UDP echo+chargen bomb"; reference: cve, CAN-1999-0635; reference: cve, CVE-1999-0103; classtype: attempted-dos; sid: 271; rev: 3;), is assigned the ht\_event\_tag 5:2:108:3 (ht\_sig\_cat 5 or malicious, ht\_dev\_cat 2 or IDS, ht\_event\_cat 108 or DOS, ht\_event\_id (in ht\_event\_cat 108) 3 or DOS UDP echo+chargen bomb) and it is represented by the decimal number 1359839235.

[0048] Next consider the vulnerability detected by the Foundstone VA tool script unix-chargen-dos-v2.fasl3. This vulnerability is assigned the ht\_event\_tag 5:4:108:3 and it is represented by the decimal number 1376616451. If the Foundstone VA tool determined that this vulnerability existed on a node that was assigned the IP address 10.1.15.106 the file va\_vulns.dat would contain the following entry:

[0049] ipaddr\_converted<|>ht\_event\_tag<|>va\_sid

[0050] 167841642<|>1376616451<|>673

[0051] At runtime the table that reflects the state of vulnerabilities in the monitored network will reflect this information. When an event is detected with a destination IP address that matches an entry in this table entry the ht\_event\_tags are compared, excluding the ht\_dev\_cat segment, which is different because and IDS detected the event and a VA tool detected the existence of the vulnerability.

[0052] If a system is not vulnerable to an exploit, but is still listening on a port that is used by an exploit, we can alert the TVS operator to this by referencing the data contained in the table va\_ports\_open.dat. If the node was a Sun Server running the sunrpc service the file va\_ports\_open.dat might contain the following entry:

[0053] ipaddr\_converted<|>protocol<|>port

[0054] 167841642<|>tcp<|>111

[0055] The currency of VA scans is evaluated by monitoring the file va\_os\_scandate.dat. The file va\_os\_scandate.dat might contain the following entry:

[0056] ipaddr\_converted<|> Os <|>scandate

[0057] 167841642<|>Linux 2.4.7-13<|>1084386780

[0058] As a result of the above examination of the use of the TVS Event Data Normalization Architecture and process you should have a good understanding of the functionality and application of this technology. We also need to gain an understanding of the nature and magnitude of the data that we are dealing with in our process of Event Data Normalization. Towards that end we will examine the data requirements of TVS.

Data Requirements Overview:

[0059] In order to support the goal of becoming a market ready production class SEM tool it is evident that managing the volume of information required to develop, build and support the product will require a detailed, well thought out information management architecture.

Initial State:

[0060] In order to begin this effort, it is necessary to understand the data requirements that TVS had and the current methods of supporting those requirements. TVS has a number of data requirements, these include value mappings that provide translations from vendor signature identifier to HighTower signature identifier; value mappings that provide translations from HighTower signature identifier to event textual description; value mappings that provide translations from port number to service name and there are many requirements for vendor information support in order to provide the forensic capabilities of the portal. As can be seen in FIG. 4 the initial state of TVS required that a number of manual processes be performed to populate the files and databases that are used in the TVS runtime environment.

[0061] These manual processes are cumbersome at best and non-repeatable at worst. The information used to support or feed these processes resided only on the desktop of staff workstations. Revision control ranged from difficult to non-existent and information updates are not easily implemented.

Solution:

[0062] In order to support the data requirements of TVS, a Centralized Global Information Repository has been defined and developed as has the definition and development of a number of interfaces to that repository. A high level example of this repository and its interfaces can be seen in FIG. 5.

[0063] In order to develop this repository a number of steps are required. At a high level these included:

[0064] Define the components within the TVS environment that will require access to the repository.

[0065] Supported component name

[0066] Function provided

[0067] Define the nature of the access required.

[0068] Read only, read/write, read/modify/write, write only

[0069] Reason for access

[0070] Define change control policy.

[0071] Process

[0072] Revision management

- [0073] Enforcement
  - [0074] Rollbacks
  - [0075] Once the components are identified, the details of their access requirements is defined. This looks like:
    - [0076] Define the detailed data requirements of the TVS runtime environment.
    - [0077] Define the method of interface to the TVS runtime environment
      - [0078] Read only
    - [0079] Define runtime database requirements
      - [0080] Mysql tables
    - [0081] Define all value mappings files
      - [0082] Value maps
      - [0083] Config files
    - [0084] Define vendor support requirements
      - [0085] Vendor references and linkages
    - [0086] Define portal requirements
      - [0087] Table requirements
      - [0088] Query requirements
    - [0089] Define change control policy
      - [0090] Who, what, when, where and why?
    - [0091] Define change control process
      - [0092] How?
    - [0093] Define current data population processes
      - [0094] Identify
      - [0095] Automate
      - [0096] Document
    - [0097] Identify output reasons
      - [0098] Normal runtime builds
      - [0099] Runtime updates
      - [0100] Custom builds
        - [0101] Customer has a custom security device (ie. Snort with custom rules)
    - [0102] Additional discovery if necessary
  - [0103] Define the detailed data requirements of the Vendor Data Normalization Tool.
    - [0104] Define the method of interface to the Vendor Data Normalization Tool
      - [0105] Read/modify/write
    - [0106] Define VDNT runtime database requirements
      - [0107] Tables (view) required
    - [0108] Define change control policy
      - [0109] Who, what, when, where and why?
    - [0110] Define change control process
      - [0111] How?
  - [0112] Define current data population processes
    - [0113] Identify
    - [0114] Automate
    - [0115] Document
    - [0116] Additional discovery if necessary
  - [0117] Define the detailed data requirements for supporting data.
    - [0118] Define the method of interface to the supporting data
      - [0119] Write only
    - [0120] Define the content
      - [0121] Standards
      - [0122] RFC guidelines
      - [0123] Port number to name
      - [0124] Protocol number to name
    - [0125] Define change control process
      - [0126] How?
    - [0127] Define current data population processes
      - [0128] Identify
      - [0129] Automate
      - [0130] Document
      - [0131] Additional discovery if necessary
  - [0132] Define the requirements for vendor raw data.
    - [0133] Define the method of interface to the vendor data
      - [0134] Write only
    - [0135] Volume of data
      - [0136] Size-number of records/events
      - [0137] Number of files or tables
    - [0138] Format of data
      - [0139] Table or file structure
    - [0140] Define change control process
      - [0141] How?
    - [0142] Define current data population processes
      - [0143] Identify
      - [0144] Automate
      - [0145] Document
      - [0146] Additional discovery if necessary
- A Closer Look at Some Data Requirements:
- [0147] In order to understand the reasoning behind and the requirements of such a repository, detail about the runtime data requirements of TVS is provided. Some of the environments requirements are depicted in **FIG. 6**.
  - [0148] This is by no means a complete list of the data requirements of the TVS runtime environment. Up to this point in the history of TVS the data required for runtime had

been compiled bit by bit, file by file from disparate sources. Updates had been tedious manual processes. TVS has developed a methodology for providing easily repeatable, automated data update methods. These data updates encompass the entire product and provide maintenance signature updates or even custom signature updates for example for a customer who writes their own snort or Dragon IDS rules. FIG. 7 depicts the process for providing TVS runtime environment data from the central repository. As shown in the diagram the user specifies the requirements of the data population via a "User Control Interface" or "Wizard". This interface allows the user to specify the information required for this instance of the TVS runtime environment data population. In essence simple input data source to output data file specification. In FIG. 7, there are three destinations for data used in TVS runtime. First and most importantly there is the TVS Runtime Environment which is a set of all data required to support the TVS runtime environment. This is used to support full TVS builds. Secondly are routine maintenance updates. These are provide support for vendor event updates. Thirdly are custom updates which provide TVS support for events that are not part of the vendor's event base but are events that the customer has defined which are implemented in the context of a supported vendor's product. For example in the case of snort and Dragon (two IDS' that are currently supported by TVS) functionality is supported that enables the end user to define their own IDS rules. TVS runtime environment data for clients requiring support of custom events is supported in a "one-off" manner.

[0149] Each of the components within the TVS environment that will require access to the repository is evaluated to determine how it integrates with the repository. In the case of support for the TVS Runtime Environment a configuration wizard has been developed that is integrated to provide the input controls required to define the TVS Runtime Environment build process.

[0150] This functionality requires that some form of the repository ship with the TVS product. The contents of the shipped repository has been scrupulously reviewed and defined so that while providing the customer with the functionality they expect we do not disclose corporate confidential information. For example; we are required to ship the event profiles table, which is the structure that defines the relationships between disparate vendor's definitions of like events, and so that we can support linking to vendors web based documentation in support of the forensic capabilities of the portal, the raw data tables from which the event profiles table is built, but we do not ship the normalization tool that is used to evaluate the raw data tables and produce the event profiles table. The normalization tool, the normalization process and its specific functionality are proprietary items.

[0151] The shipped repository also provides a suitable target for periodic maintenance updates for TVS device support and with additional development will serve as a vehicle for TVS application maintenance.

[0152] A process for providing maintenance update for device support will include distributing new or updated raw

device files, new or updated event profile tables and new or updated value mapping files and any other new or updated files required by the system.

[0153] An overview of this process is depicted in FIG. 8.

[0154] Although only a few embodiments have been disclosed in detail above, other modifications are possible, and this disclosure is intended to cover all such modifications, and most particularly, any modification which might be predictable to a person having ordinary skill in the art.

[0155] Also, only those claims which use the words "means for" are intended to be interpreted under 35 USC 112, sixth paragraph. Moreover, no limitations from the specification are intended to be read into any claims, unless those limitations are expressly included in the claims.

What is claimed is

1. A method, comprising:

receiving a packet from a network security agent indicating a network event;

converting the packet to a security event tag that represents information indicative of the event and information indicative of a type of agent which detected the event; and

using the security event tag to represent the event in place of the packet.

2. A method as in claim 1, further comprising determining if the packet is from a registered device.

3. A method as in claim 1, wherein the network security agent is one of a firewall, a network intrusion system, a router, or a virtual private network.

4. A method as in claim 1, wherein the security event tag has common fields for the same event from different agents.

5. A method as in claim 1, wherein the security event tag represents at least an IP address, at least one port, and at least one signature identifier.

6. A system, comprising:

A port that receives a packet from a network security agent indicating a network event;

A processing engine operating to convert the packet to a security event tag that represents information indicative of the event and information indicative of a type of agent which detected the event; and

A security monitoring system that uses the security event tag to represent the event in place of the packet.

7. A system as in claim 6, further comprising the network security agent.

8. A system as in claim 7, wherein said network security agent is one of a firewall, a network intrusion system, a router, or a virtual private network.

9. A method as in claim 6, wherein the security event tag has common fields for the same event from different agents.

10. A method as in claim 9, wherein the security event tag represents at least an IP address, at least one port, and at least one signature identifier.

\* \* \* \* \*