MERGING MULTI-LINE LOG ENTRIES

RELATED APPLICATIONS

[0001] The following related applications are each incorporated by reference herein:

[0002] U.S. Application Serial No. 10/308,548 of Hugh S. Njemanze et al., entitled "Modular Agent For Network Security Intrusion Detection System," filed: December 2, 2002.

[0003] U.S. Application Serial No. 10/308,584 of Hugh Njemanze et al., entitled "Method For Aggregating Events To Be Reported By Software Agent," filed December 2, 2002.

[0004] U.S. Application Serial No. 10/821,459 of Kenny Tidwell et al., entitled "Comparing Events From Multiple Network Security Devices," filed April 9, 2004.

[0005] U.S. Application Serial No. 10/975,962 of Debabrata Dash, entitled "Security Event Aggregation At Software Agent," filed November 27, 2004.

[0006] U.S. Application Serial No. 11/070,024 of Hector Aguilar-Macias et al., entitled "Message Parsing In A Network Security System," filed March 1, 2005.

TECHNICAL FIELD

[0007] The disclosed embodiments relate generally to monitoring of network activity. More particularly, the disclosed embodiments relate to a system and method for merging multiple entries representing related network activity.

BACKGROUND

[0008] It is desirable to monitor log entries received from various devices and pieces of software in a network. Frequently, those other devices or pieces of software may create several logging messages for reasons of convenience, speed, or reliability. This is done, for example, so that some information will reach the central point for the event, even if not all information does. For instance, it may be desirable to send a log message before the work is completed to make sure something is recorded even if the system later crashes before completely finishing the work in question.

[0009] In addition, certain types of log events occur in the device over time. It is considered desirable to send loggable events as they occur, instead of waiting until all loggable occurrences have happened for an event at a device.

[0010] If multiple devices send log entries to one or more central collection points in the network, the log entries for the various events from the various devices will most likely arrive interspersed with each other. The various log entries may not be adjacent in the log. They may be interleaved with very similar events. They may be spread across several log files. The sequence of entries may not be complete (perhaps the sensor crashed before the operation was completed).

[0011] What is needed is a way to automatically collect high-level event information from log entries that were generated under the problematic conditions described above.

SUMMARY

[0012] Preferred embodiments of the present invention define an agent containing a parser, a grouping tracker module, and a mapping module. The parser separates arriving log entries into tokens. The grouping tracker analyzes these tokens to determine which merged events the tokens belong to (if any). In the described embodiment, the grouping tracker operates in accordance with configurable merge properties, although other embodiments may have these properties hard-coded. The merge properties allow configuration of various properties associated with the act of grouping the log entries into high-level merged events. In the described embodiment, these properties include some or all of: what types of log entries will be considered for each merged event, which IDs are used to identify each merged event, which entries begin and end a merged event, a timeout value that automatically ends collection of entries for an existing merged event, even if no end entry is found.

[0013] In the described embodiment, the mapping module receives log entries associated with specific merged events and maps them to fields in the merged event data structure in accordance with mapping properties (although these mapping properties could also be hard-coded).

[0014] The described embodiments of the invention use regular expressions in the merge properties to describe values that are searched for in the received log entries. For example, a regular expression may define which entries are part of a multi-entry event, may detect the first entry in a multi-entry event, and may detect the last entry in a multi-entry event. The merge properties also declare which field in the entries must contain the same values in order to be merged (for instance, the entries might both have the same numeric id or mention the same ip address). The described embodiment of the present invention can process log entries for events that are interspersed with each other.

BRIEF DESCRIPTION OF THE DRAWINGS

[0015] FIG. 1 shows a block diagram of a system in accordance with an embodiment of the present invention.

[0016] FIG. 2 is a flowchart of an embodiment of a method performed to process log entries in accordance with merge properties.

[0017] FIG. 3 is a flow chart of an embodiment of a method performed to add a log entry to a merged event in accordance with mapping properties.

[0018] FIG. 4 is a flow chart showing a oneOf function used in the mapping properties in an embodiment of the present invention.

[0019] FIG. 5 shows an example in which multiple merged events are being constructed, as interspersed log entries for the various merged events are received.

[0020] FIG. 6 shows an example format of one merged event.

DESCRIPTION OF EMBODIMENTS

[0021] Embodiments of the present invention are now described with reference to the figures where like reference numbers indicate identical or functionally similar elements.

[0022] FIG. 1 shows a block diagram of a system 100 in accordance with an embodiment of the present invention. System 100 preferably contains an agent 104 in one or more central points in a network. Agent 104 receives log entries from multiple devices and pieces of software over a network, such as the internet, a LAN, WAN, wireless network, mobile network, or any other appropriate mechanism that allows remote devices to send log entries to agent 104.

[0023] Log entries are received by a parser 102 and parsed into tokens in a manner known to persons of ordinary skill in the art. In another embodiment, parsing is performed as described in U.S. Application Serial No. 11/070,024 of Hector Aguilar-Macias et al., entitled "Message Parsing In A Network Security System," filed March 1, 2005, which is herein incorporated by reference.

[0024] The received log entries can be any appropriate format that parser 102 is able to parse. Parser 102 outputs tokens based on the received log entries. These tokens are received by a grouping tracker module 110.

[0025] Grouping tracker module 110 is connected to receive merge properties from a memory or other storage module or device 112. The merge properties specify how received log entries are to be interpreted as they are used to build merged events.

Grouping tracker module outputs log entries that are associated with specific merged events into a mapping module where the log entries are mapped into merged events that are being built up from the received log entries. This mapping occurs in accordance with mapping properties 122. The output of mapping module 120 is one or more merged events resulting from multiple log entries. The process generally described in FIG. 1 will be described in more detail below in connection with an example.

Example

[0026] Here is an example of how event merging works in an embodiment of the invention:

[0027] Assume the following lines of log entries (these are also sometimes called "messages"):

[0028] [18/Jul/2005:12:30:20 -0400] conn=8 op=0 msgId=82 - BIND uid=admin

[0029] [18/Jul/2005:12:30:25 -0400] conn=7 op=-1 msgId=-1 - LDAP connection from 10.0.20.122 to 10.0.20.12.

[0030] [18/Jul/2005:12:30:30 -0400] conn=8 op=0 msgId=82 - RESULT err=0

[0031] Parser 102 parses these received log entries into key-value pairs. For each log entry this yields a set of tokens. For example, the log entry:

[0032] **[18/Jul/2005:12:30:20 -0400]] conn=8 op=0 msgId=82 - BIND uid=admin**

[0033] Yields tokens having the following key/value pairs:

[0034] Date=18/Jul/2005 12:30:20

[0035] Connection=8

[0036] Operation=0

[0037] MessageId=82

[0038] OperationName=BIND

[0039] UserId=admin

[0040] Similarly, the other two log entries yield their own key/value pairs:

[0041] **[18/Jul/2005:12:30:25 -0400]] conn=7 op=-1 msgId=-1 - LDAP connection from 10.0.20.122 to 10.0.20.12**

[0042] Date=18/Jul/2005 12:30:25

[0043] Connection=7

[0044] Operation=1

[0045] MessageId=-1

[0046] OperationName=LDAP

[0047] Source=10.0.20.122

[0048] Destination=10.0.20.12

[0049] **[18/Jul/2005:12:30:30 -0400]] conn=8 op=0 msgId=82 - RESULT err=0**

[0050] Date=18/Jul/2005 12:30:30

[0051] Connection=8

[0052] Operation=0

[0053] MessageId=82

[0054] OperationName=RESULT

[0055] ResultCode=0

[0056] FIG. 2 is a flowchart 200 of an embodiment of a method performed to process received log entries in accordance with merge properties 112. In a preferred embodiment, the method is performed by grouping tracker module 110. If a timeout 202 is reached for a merged event currently being built, the merged event is ended 204 and control returns to element 202. Thus, even if no explicit ending log entry is found, a merged event will be closed when its timeout occurs. The timeout value may differ for different types of logging devices and for different merged events from a single device. As described below, the timeout value is contained in the merge properties.

[0057] Element 206 receives a next log entry to process. If the log entry is to be considered for merging 208 (as defined in merge properties 112), the processing continues, otherwise a single event is sent 209 and processing returns to element 202.

[0058] If the log entry is a beginning log entry for a new merged event 210 (as defined in merge properties 112), a new merged event is opened 212 (see FIG. 5 for an example of multiple merged events in the process of being built). In some embodiments, the timeout clock for the merged event is started 212.

[0059] If the log entry is not a beginning log entry, but it contains an ID of an existing merged event currently being built 214, then an exception is logged and a single event is sent 215. Otherwise, processing continues and the tokens and log entry are passed 220 to the mapping module so that its information can be added to the merged event. In an embodiment, an ID can be a single field in the log entry or can be multiple fields in the log entry that have common values for all log entries of a merged event.

[0060] If the log entry is an end log entry for a new merged event 216 (as defined in merge properties 112), an existing merged event is ended and removed 218 from the grouping tracker module (see FIG. 5 for an example of multiple merged events in the process of being built). If a log entry indicates an event end, the corresponding merged event will be ended and removed from the structure of FIG. 5.

[0061] To continue the example, the merge properties 112 in this example are defined as:

[0062] merge.count=1

[0063] merge[0].pattern.count=1

[0064] merge[0].pattern[0].token=OperationName

[0065] merge[0].pattern[0].regex=(BIND | RESULT)

[0066] merge[0].starts.count=1

[0067] merge[0].starts[0].token=OperationName

[0068] merge[0].starts[0].regex=BIND

[0069] merge[0].ends.count=1

[0070] merge[0].ends[0].token=OperationName

[0071] merge[0].ends[0].regex=RESULT

[0072] merge[0].id.tokens=Connection,Operation,MessageId

[0073] merge[0].timeout=60000

[0074] First we indicate that we have only 1 merge operation:

[0075] merge.count=1

[0076] Then we define that we want all the messages with **OperationName** set to **BIND** or **RESULT** to be considered for merging:

[0077] merge[0].pattern.count=1

[0078] merge[0].pattern[0].token=OperationName

[0079] merge[0].pattern[0].regex=(BIND | RESULT)

[0080] Now we specify that the messages that have an **OperationName** set to **BIND** will start the merge operation:

[0081] merge[0].starts.count=1

[0082] merge[0].starts[0].token=OperationName

[0083] merge[0].starts[0].regex=BIND

[0084] And that the merge operation will end once we find a message **OperationName** set to **RESULT**:

[0085] merge[0].ends.count=1

[0086] merge[0].ends[0].token=OperationName

[0087] merge[0].ends[0].regex=RESULT

[0088] We also need to define how to identify that events belong to the same group, we do that by specifying that the values of **Connection** , **Operation** and **MessageId** must be identical (forming an ID for the merged event):

[0089] merge[0].id.tokens=Connection,Operation,MessageId

[0090] Finally we define a timeout so that if we do not get the message with **OperationName** set to **RESULT** after 60 seconds, then we will send the event as is:

[0091] merge[0].timeout=60000

[0092] FIG. 3 is a flow chart of an embodiment of a method performed to add a log entry to a merged event in accordance with mapping properties. Received log entries and their tokens have already been identified as being relevant to at least one merged event being built. Mapping module 120 maps information in the log entries to one or more merged events being built (see FIG. 5 for an example of multiple merged events being built. See FIG. 6 for examples of a format for a merged event.)

[0093] In this example, mapping properties 122 are defined as:

[0094] event.deviceReceiptTime=Date

[0095] event.name=__oneOf(mergedevent.name,OperationName)

[0096] event.deviceAction=ResultCode

[0097] event.destinationUserId=UserId

[0098] These properties indicate that we will use the **Date** as the timestamp for the event, the **ResultCode** as the device action and the **UserId** as the destination user id. The name is defined as:

[0099] event.name=__oneOf(mergedevent.name,OperationName)

[00100] Because this framework also allows you to refer to the "tracking" event that is being used to store the final data. In this case the operation means that either we should use the **OperationName** or the name of the "tracking" event (if any). For example, the first event will contain the following key-values:

[00101] **[18/Jul/2005:12:30:20 -0400]] conn=8 op=0 msgId=82 - BIND uid=admin**

[00102] Date=18/Jul/2005 12:30:20

[00103] Connection=8

[00104] Operation=0

[00105] MessageId=82

[00106] OperationName=BIND

[00107] UserId=admin

[00108] And a new "tracking" event will be created that will end up with the following mappings:

[00109] mergedevent.name=BIND

[00110] mergedevent.deviceReceiptTime=18/Jul/2005 12:30:20

[00111] mergedevent.destinationUserId=admin

[00112] The name of the mergedevent will be BIND because this is a **new** mergedevent, so mergedevent.name does not exist and the value of **OperationName** is used (BIND). Now when the second event for the merging group is processed:

[00113] **[18/Jul/2005:12:30:30 -0400]] conn=8 op=0 msgId=82 - RESULT err=0**

[00114] Date=18/Jul/2005 12:30:30

[00115] Connection=8

[00116] Operation=0

[00117] MessageId=82

[00118] OperationName=RESULT

[00119] ResultCode=0

[00120] The merged event will be mapped as follows:

[00121] mergedevent.name=BIND

[00122] mergedevent.deviceReceiptTime=18/Jul/2005 12:30:30

[00123] mergedevent.destinationUserId=admin

[00124] mergedevent.deviceAction=0

[00125] Notice that **mergedevent.name** will be set to BIND because when this event is processed there was already a "tracked" event (mergedevent) with the name set to BIND, so in this case **OperationName** will NOT be used and the mergedevent keeps the value BIND. Notice how the **mergedevent.deviceReceiptTime** now was set to **18/Jul/2005 12:30:30** that is because by default the values of mergedevent will be replaced, so deviceReceiptTime will assume the newer value.

[00126] FIG. 4 is a flow chart 402 showing the oneOf function 400 used in the mapping properties in an embodiment of the present invention. To process the oneOf function for, for example, an event name, if the event name is currently blank 404, the current token name is used 406. If the name is not blank, the non-blank name is retained 408.

[00127] It will be understood that _oneOf is only an example of operations that can be used in the mappings component. The mapping component may contain other "operations" that can make reference to the merged event fields. __oneOf is just an example, in the actual mapping framework Other examples of operations include __concatenate, type conversion operations and others.

[00128] FIG. 5 shows an example 500 in which multiple merged events are being constructed, as interspersed log entries for the various merged events are received.

[00129] FIG. 6 shows an example format 550 of one merged event. For example, each of the various merged events of FIG. 5 will have this format, although not all the

values may be filled in for each merged event. Various implementations of the present invention will contain other examples of merge operations including concatenate, type conversion, counting, and others. Other embodiments include merged event aggregation so that statistics can be kept for numbers of various types of merged events. These aggregated data can be sent to a monitor alone or as part of a combination of other sent data.

[00130] The following paragraphs provide a short description of example merge properties 112 included in one embodiment of the invention:

[00131] **merge.count**

[00132] Defines the number of merge operations that will be defined.

[00133] **merge[*{mergeindex}*].pattern.count**

[00134] Defines how many patterns will be defined. Merge operations require patterns to define which events will be considered in the merge operation, if no patterns are given then ALL events will be considered.

[00135] **merge[*{mergeindex}*].pattern[*{patternindex}*].token**

[00136] Defines the token that will be used for this pattern.

[00137] **merge[*{mergeindex}*].pattern[*{patternindex}*].regex**

[00138] Defines the regular expression to use for this pattern.

[00139] **merge[*{mergeindex}*].starts.count**

[00140] Defines how many start patterns will be defined. Merge operations require start patterns to define which events will start a merge operation, if no patterns

are given then ALL events will start a merge operation. Once the operation has been started it can only be ended via a timeout or an end pattern match.

[00141] **merge[*{mergeindex}*].starts[*{patternindex}*].token**

[00142] Defines the token that will be used for this start pattern.

[00143] **merge[*{mergeindex}*].starts[*{patternindex}*].regex**

[00144] Defines the regular expression to use for this start pattern.

[00145] **merge[*{mergeindex}*].ends.count**

[00146] Defines how many end patterns will be defined. Merge operations require end patterns to define which events will end the merge operation, if no patterns are given then no event will end a merge operation, the operation will only end via a timeout.

[00147] **merge[*{mergeindex}*].ends[*{patternindex}*].token**

[00148] Defines the token that will be used for this end pattern.

[00149] **merge[*{mergeindex}*].ends[*{patternindex}*].regex**

[00150] Defines the regular expression to use for this end pattern.

[00151] **merge[*{mergeindex}*].timeout**

[00152] Defines the timeout in milliseconds for the merging operation. If the timeout is reached then the merge operation will end and the events will be sent. Be aware that these events will be sent via a different thread, so event order is not guaranteed.

[00153] **merge[*{mergeindex}*].id.tokens**

[00154] Defines the list of tokens that will be used to group the events. This property is required.

[00155] **merge[*{mergeindex}*].id.delimiter**

[00156] Defines an optional delimiter to use for the list above, if it is not defined then the delimiter is a comma (,).

[00157] **merge[*{mergeindex}*].sendpartialevents**

[00158] This property is optional and set to false by default. Basically it specifies if each event in the merge operation must be sent individually as it is merged with other events.

[00159] **merge[*{mergeindex}*].capacity**

[00160] This property is optional and set to 1000 by default. An event merging operation requires a cache of events that hold the merged results. This defines how big the cache will be, if the cache overflows then events will be sent as they are and an error will be logged.

[00161] Reference in the specification to "one embodiment" or to "an embodiment" means that a particular feature, structure, or characteristic described in connection with the embodiments is included in at least one embodiment of the invention. The appearances of the phrase "in one embodiment" in various places in the specification are not necessarily all referring to the same embodiment.

[00162] Some portions of the above are presented in terms of algorithms and symbolic representations of operations on data bits within a computer memory. These algorithmic descriptions and representations are the means used by those skilled in the

data processing arts to most effectively convey the substance of their work to others skilled in the art. An algorithm is here, and generally, conceived to be a self-consistent sequence of steps (instructions) leading to a desired result. The steps are those requiring physical manipulations of physical quantities. Usually, though not necessarily, these quantities take the form of electrical, magnetic or optical signals capable of being stored, transferred, combined, compared and otherwise manipulated. It is convenient at times, principally for reasons of common usage, to refer to these signals as bits, values, elements, symbols, characters, terms, numbers, or the like. Furthermore, it is also convenient at times, to refer to certain arrangements of steps requiring physical manipulations of physical quantities as modules or code devices, without loss of generality.

[00163] It should be borne in mind, however, that all of these and similar terms are to be associated with the appropriate physical quantities and are merely convenient labels applied to these quantities. Unless specifically stated otherwise as apparent from the following discussion, it is appreciated that throughout the description, discussions utilizing terms such as "processing" or "computing" or "calculating" or "determining" or "displaying" or "determining" or the like, refer to the action and processes of a computer system, or similar electronic computing device, that manipulates and transforms data represented as physical (electronic) quantities within the computer system memories or registers or other such information storage, transmission or display devices.

[00164] Certain aspects of the present invention include process steps and instructions described herein in the form of an algorithm. It should be noted that the process steps and instructions of the present invention can be embodied in software, firmware or hardware, and when embodied in software, can be downloaded to reside on and be operated from different platforms used by a variety of operating systems.

[00165] The present invention also relates to an apparatus for performing the operations herein. This apparatus may be specially constructed for the required purposes, or it may comprise a general-purpose computer selectively activated or reconfigured by a computer program stored in the computer. Such a computer program may be stored in a computer readable storage medium, such as, but is not limited to, any type of disk including floppy disks, optical disks, CD-ROMs, magnetic-optical disks, read-only memories (ROMs), random access memories (RAMs), EPROMs, EEPROMs, magnetic or optical cards, application specific integrated circuits (ASICs), or any type of media suitable for storing electronic instructions, and each coupled to a computer system bus. Furthermore, the computers referred to in the specification may include a single processor or may be architectures employing multiple processor designs for increased computing capability.

[00166] The algorithms and displays presented herein are not inherently related to any particular computer or other apparatus. Various general-purpose systems may also be used with programs in accordance with the teachings herein, or it may prove convenient to construct more specialized apparatus to perform the required method steps. The required structure for a variety of these systems will appear from the

description below. In addition, the present invention is not described with reference to any particular programming language. It will be appreciated that a variety of programming languages may be used to implement the teachings of the present invention as described herein, and any references below to specific languages are provided for disclosure of enablement and best mode of the present invention.

[00167] While the invention has been particularly shown and described with reference to a preferred embodiment and several alternate embodiments, it will be understood by persons skilled in the relevant art that various changes in form and details can be made therein without departing from the spirit and scope of the invention.

[00168] Finally, it should be noted that the language used in the specification has been principally selected for readability and instructional purposes, and may not have been selected to delineate or circumscribe the inventive subject matter. Accordingly, the disclosure of the present invention is intended to be illustrative, but not limiting, of the scope of the invention, which is set forth in the following claims.

CLAIMS

What is claimed is:

1. A method for merging multiple log entries received by a data processing system, comprising:

receiving a plurality of log entries;

for each received log entry:

determining if the log entry contains an ID common to any potential merged event in accordance with merge properties;

beginning a new merged event, if the log entry is a beginning log entry of a merged event in accordance with the merge properties; and

ending an existing merged event, if the log entry is an ending log entry of an existing merged event in accordance with the merge properties; and

mapping each log entry containing an ID common to an existing merged event to that merged event in accordance with mapping properties for the merged event.

2. The method of claim 1, further comprising

ending an existing merged event if a timeout as defined in the merge properties for that merged event occurs.

3. The method of claim 2, further comprising identifying a log entry indicating a beginning of the merged event.

4. The method of claim 2, further comprising identifying a log entry indicating an end of the merged event.

5. The method of claim 2, further comprising identifying a log entry that neither implies a start nor an end of the merged event.

6. The method of claim 2, further comprising an ability to consider the merged event as it exists so far when merging in a new entry's token.

7. The method of claim 6, wherein the ability is used in a mapping operation.

8. The method of claim 1, further comprising

determining whether each received log entry is to be considered for merging in accordance with the merge properties.

9. The method of claim 1, wherein mapping a log entry to a merged event further comprises determining a time of the merged event, the time being the time of a beginning log event for the merged event.

10. The method of claim 1, wherein mapping a log entry to a merged event further comprises determining a time of the merged event, the time being the time of an ending log event for the merged event.

11. The method of claim 1, wherein mapping a log entry to a merged event further comprises mapping an event ID in accordance with the mapping properties.

12. The method of claim 1, wherein mapping a log entry to a merged event further comprises mapping an event name in accordance with the mapping properties.

13. The method of claim 1, wherein mapping a log entry to a merged event further comprises mapping a name parsed from the log entry, the mapping performed in accordance with a oneOf function in the mapping properties.

14. The method of claim 1, wherein mapping a log entry to a merged event further comprises mapping a device action in accordance with the mapping properties.

15. The method of claim 1, wherein the received log entries contain log entries corresponding to more than one merged event mixed together.

16. The method of claim 1, wherein one received log entry is used to build more than one merged event.

17. The method of claim 1, wherein an ID comprises multiple fields in the log entry, the multiple fields acting to identify log entries that contribute to a merged event.

18. A system for merging multiple log entries received by a data processing system, comprising:

a module for receiving a plurality of log entries;

a parser for parsing the log entries into tokens;

a grouper that, for each received log entry:

determines if the log entry contains an ID common to any potential merged event in accordance with merge properties;

begins a new merged event, if the log entry is a beginning log entry of a merged event in accordance with the merge properties; and

ends an existing merged event, if the log entry is an ending log entry of an existing merged event in accordance with the merge properties; and

a mapper that maps each log entry containing an ID common to an existing merged event to that merged event in accordance with mapping properties for the merged event.

19. A computer program product comprising instructions stored on a computer readable medium for causing a computer to perform a method, comprising:

receiving a plurality of log entries;

for each received log entry:

determining if the log entry contains an ID common to any potential merged event in accordance with merge properties;

beginning a new merged event, if the log entry is a beginning log entry of a merged event in accordance with the merge properties; and

ending an existing merged event, if the log entry is an ending log entry of an existing merged event in accordance with the merge properties; and

mapping each log entry containing an ID common to an existing merged event to that merged event in accordance with mapping properties for the merged event.