PATTERN DISCOVERY IN A NETWORK SECURITY SYSTEM

FIELD OF THE INVENTION

[0001] The present invention relates to a computer-based system for capturing security events from heterogeneous and homogenous sources, and discovering patterns in the captured security events.

BACKGROUND

[0002] Computer networks and systems have become indispensable tools for modern business. Today terabits of information on virtually every subject imaginable are stored in and accessed across such networks by users throughout the world. Much of this information is, to some degree, confidential and its protection is required. Not surprisingly then, intrusion detection systems (IDS) have been developed to help uncover attempts by unauthorized persons and/or devices to gain access to computer networks and the information stored therein. In addition, network devices such as routers and firewalls maintain activity logs that can be used to examine such attempts.

[0003] Intrusion detection may be regarded as the art of detecting inappropriate, incorrect or anomalous activity within or concerning a computer network or system. The most common approaches to intrusion detection are statistical anomaly detection and pattern-matching detection. IDS that operate on a host to detect malicious activity on that host are called host-based IDS (HIDS), which may exist in the form of host wrappers/personal firewalls or agent-based software, and those that operate on network data flows are called network-based IDS (NIDS). Host-based intrusion detection involves loading software on the system (the host) to be monitored and using log files and/or the host's auditing agents as sources of data. In contrast, a network-based intrusion detection system monitors the traffic on its network segment and uses that traffic as a data source. Packets captured by the network interface cards are considered to be of interest if they match a signature.

[0004] Regardless of the data source, there are two complementary approaches to detecting intrusions: knowledge-based approaches and behavior-based approaches. Almost all IDS tools in use today are knowledge-based. Knowledge-based intrusion detection techniques involve comparing the captured data to information regarding known techniques to exploit vulnerabilities. When a match is detected, an alarm is triggered. Behavior-based intrusion detection techniques, on the other hand, attempt to spot intrusions by observing deviations from normal or expected behaviors of the system or the users (models of which are extracted from reference information collected by various means). When a suspected deviation is observed, an alarm is generated.

[0005] Advantages of the knowledge-based approaches are that they have the potential for very low false alarm rates, and the contextual analysis proposed by the intrusion detection system is detailed, making it easier for a security officer using such an intrusion detection system to take preventive or corrective action. Drawbacks include the difficulty in gathering the required information on the known attacks and keeping it up to date with new vulnerabilities and environments.

[0006] Advantages of behavior-based approaches are that they can detect attempts to exploit new and unforeseen vulnerabilities. They are also less dependent on system specifics. However, the high false alarm rate is generally cited as a significant drawback of these techniques and because behaviors can change over time, the incidence of such false alarms can increase.

[0007] Regardless of whether a host-based or a network-based implementation is adopted and whether that implementation is knowledge-based or behavior-based, an intrusion detection system is only as useful as its ability to discriminate between normal system usage and true intrusions (accompanied by appropriate alerts). If intrusions can be detected and the appropriate personnel notified in a prompt fashion, measures can be taken to avoid compromises to the protected system. Otherwise such safeguarding cannot be provided. Accordingly, what is needed is a system that can provide accurate and timely intrusion detection and alert generation so as to effectively combat attempts to compromise a computer network or system.

SUMMARY OF THE INVENTION

[0008] Patterns can be discovered in security events collected by a network security system. In one embodiment, the present invention includes collecting and storing security events from a variety of monitor devices. In one embodiment, a subset of the stored security events is provided to a manager as an event stream. In one embodiment, the present invention further includes the manager discovering one or more previously unknown event patterns in the event stream.

BRIEF DESCRIPTION OF THE DRAWINGS

[0009] The present invention is illustrated by way of example, and not limitation, in the figures of the accompanying drawings in which:

[0010] **Figure 1** is a block diagram of a network security system according to one embodiment of the present invention;

[0011] **Figure 2** is a block diagram of a pattern discovery module according to one embodiment of the present invention;

[0012] **Figure 3** is a block diagram of a pattern identifier according to one embodiment of the present invention;

[0013] **Figure 4** is a flow chart illustrating a process for transaction tree building according to one embodiment of the present invention.

DETAILED DESCRIPTION

[0014] Although the present system will be discussed with reference to various illustrated examples, these examples should not be read to limit the broader spirit and scope of the present invention. For example, the examples presented herein describe distributed agents, managers and consoles, which are but one embodiment of the present invention. The general concepts and reach of the present invention are much broader and may extend to any computer-based or network-based security system. Also, examples of the messages that may be passed to and from the components of the system and the data schemas that may be used by components of the system are given in an attempt to further describe the present invention, but are not meant to be all-inclusive examples and should not be regarded as such.

[0015] Some portions of the detailed description that follows are presented in terms of algorithms and symbolic representations of operations on data within a computer memory. These algorithmic descriptions and representations are the means used by those skilled in the computer science 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 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 or magnetic signals capable of being stored, transferred, combined, compared and otherwise manipulated. It has proven 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. 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, it will be appreciated that throughout the description of the present invention, use of terms such as "processing", "computing", "calculating", "determining", "displaying" 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's registers and memories into other data similarly represented as physical quantities within the computer system memories or registers or other such information storage, transmission or display devices.

[0016] As indicated above, one embodiment of the present invention is instantiated in computer software, that is, computer readable instructions, which, when executed by one or more computer processors/systems, instruct the processors/systems to perform the designated actions. Such computer software may be resident in one or more computer readable media, such as hard drives, CD-ROMs, DVD-ROMs, read-only memory, read-write memory and so on. Such software may be distributed on one or more of these media, or may be made available for download across one or more computer networks (e.g., the Internet). Regardless of the format, the computer programming, rendering and processing techniques discussed herein are simply examples of the types of programming, rendering and processing techniques that may be used to implement aspects of the present invention. These examples should in no way limit the present invention, which is best understood with reference to the claims that follow this description.

[0017] Referring now to Figure 1, an example of a computer-based network security system 10 architected in accordance with an embodiment of the present invention is illustrated. System 10 includes agents 12, one or more managers 14 and one or more consoles 16 (which may include browser-based versions thereof). In some embodiments, agents, managers and/or consoles may be combined in a single platform or distributed in two, three or more platforms (such as in the illustrated example). The use of this multi-tier architecture supports scalability as a computer network or system grows.

[0018] Agents 12 are software programs that provide efficient, real-time (or near real-time) local event data capture and filtering from a variety of network security devices and/or applications. The primary sources of security events are common network security devices, such as firewalls, intrusion detection systems and operating system logs. Agents 12 can collect events from any source that produces event logs or messages and can operate at the native device, at consolidation points within the network, and/or through simple network management protocol (SNMP) traps.

[0019] Agents 12 are configurable through both manual and automated processes and via associated configuration files. Each agent 12 may include one or more software modules including a normalizing component, a time correction component, an aggregation component, a batching component, a resolver component, a transport component, and/or additional components. These components may be activated and/or deactivated through appropriate commands in the configuration file.

[0020] Managers 14 may be server-based components that further consolidate, filter and cross-correlate events received from the agents, employing a rules engine 18 and a centralized event database 20. One role of manager 14 is to capture and store all of the real-time and historic event data to construct (via database manager 22) a complete, enterprise-wide picture of security

activity. The manager 14 also provides centralized administration, notification (through one or more notifiers 24), and reporting, as well as a knowledge base 28 and case management workflow. The manager 14 may be deployed on any computer hardware platform and one embodiment utilizes a relational database management system such as an Oracle™ database to implement the event data store component. Communications between manager 14 and agents 12 may be bi-directional (e.g., to allow manager 14 to transmit commands to the platforms hosting agents 12) and encrypted. In some installations, managers 14 may act as concentrators for multiple agents 12 and can forward information to other managers (e.g., deployed at a corporate headquarters).

[0021] Consoles 16 are computer- (e.g., workstation-) based applications that allow security professionals to perform day-to-day administrative and operation tasks such as event monitoring, rules authoring, incident investigation and reporting. Access control lists allow multiple security professionals to use the same system and event database, with each having their own views, correlation rules, alerts, reports and knowledge base appropriate to their responsibilities. A single manager 14 can support multiple consoles 16.

[0022] In some embodiments, a browser-based version of the console 16 may be used to provide access to security events, knowledge base articles, reports, notifications and cases. That is, the manager 14 may include a web server component accessible via a web browser hosted on a personal or handheld computer (which takes the place of console 16) to provide some or all of the functionality of a console 16. Browser access is particularly useful for security professionals that are away from the consoles 16 and for part-time users. Communication between consoles 16 and manager 12 is bi-directional and may be encrypted.

[0023] Through the above-described architecture the present invention can support a centralized or decentralized environment. This is useful because an organization may want to implement a single instance of system 10 and use an access control list to partition users. Alternatively, the organization may choose to deploy separate systems 10 for each of a number of groups and consolidate the results at a "master" level. Such a deployment can also achieve a "follow-the-sun" arrangement where geographically dispersed peer groups collaborate with each other by passing primary oversight responsibility to the group currently working standard business hours. Systems 10 can also be deployed in a corporate hierarchy where business divisions work separately and support a rollup to a centralized management function.

[0024] The exemplary network security system illustrated in Figure 1 is described in further detail in U.S. Application 10/308,415, entitled "Real Time Monitoring and Analysis of Events from Multiple Security Devices", filed December 2, 2001, which is hereby incorporated fully by reference.

[0025] In one embodiment of the present invention, the network security system 10 also includes pattern recognition capabilities. In one embodiment, this is implemented on the manager 14 through a pattern discovery module 30, shown in Figure 1. As shown in Figure 1, in one embodiment, a set of security events stored in the event database 20 can be provided to the pattern discovery module 30 via the database manager 22 for pattern discovery.

[0026] In one embodiment, a user can initiate an event replay from the event database 20 using the console interface 16 by specifying a time period of previously stored security events to replay. Several embodiments for event replay are described in further detail in United States Patent Application 10/308,416 entitled "Method and Apparatus for Exercising and Debugging Correlations for Network Security System," filed December 2, 2002, which is hereby fully incorporated by reference.

[0027] In one embodiment, this replayed event stream is provided to the pattern discovery module. In another embodiment, the security events collected from the agents 12 are provided directly to the pattern discovery module 30, which takes a snapshot of the collected events from time to time. In one embodiment, the pattern discovery module 30 is configured to find – in other words discover, detect – patterns of interest in the event stream it is provided.

[0028] In one embodiment, the difference between the patterns detected by the pattern discovery module 30 and the patterns identified by the rules engine 18 is that the rules engine 18 finds known patterns in the collected security events, while the pattern discovery module finds previously unknown patterns in the security events. In other words, the rules engine 18 has correlation rules that are applied to the event stream, each rule looking for a specific pattern. However, in one embodiment, the pattern discovery module 30 does not rely on encoded domain knowledge, such as predefined rules or filters, but finds patterns previously unrecognized as such. In one embodiment, the patterns found by the pattern discovery module can be converted into correlation rules to be used by the rules engine 18 thereafter to correlate security events.

[0029] In one embodiment, the operation of the rules engine 18 and the pattern discovery module 30 also differ in that the rules engine locates specific instances of the previously known patterns it is looking for. However, in one embodiment, the pattern discovery module 30 is only configured to identify possible patterns. To locate the pattern in the event stream, a rule based on the newly detected pattern can be provided to the rules engine 18.

[0030] One embodiment of the pattern discovery module 30 is now described in more detail with reference to Figure 2. As discussed above, the input of the pattern discovery module 30 is an event stream (a collection of security events), provided, in one embodiment, from the event database 20 by the database manager 22. In one embodiment, the pattern discovery module 30 includes an event field filter 32 that retains only selected event fields for each event. For

example, event fields not useful for pattern discovery, such fields unique to each event (e.g., unique event identifiers) can be filtered out by the event field filter 32 to conserve memory and increase processing speed.

[0031] The event field filter is configured to retain at least the fields used by the transaction filters 34. In one embodiment, the event field filter 32 retains at least the event name field and the source and destination fields. In another embodiments, additional event fields are preserved, such as sensor vendor, product, category, and source and destination port. In one embodiment, there are multiple event field filters 32, one for each transaction filter 34. In such an embodiment, each event field filter 32 would retain at least those fields used by its respective transaction filter 34. The event field filter 32 can be configured to filter out or preserve any individual event field or any subset of event fields, including any combination of the event fields listed above.

[0032] In one embodiment, the security events (as represented by the preserved event fields) are used to generate transactions. In one embodiment, a transaction is defined as a group of items representing security events related by some transaction parameter. For example, all events received within some timeframe (e.g., 5 minutes, 30 minutes, one day, one week, one month ... etc.) can be one transaction. The event stream, which in one embodiment is all the events from some period of time, can be divided into any number of such time-based transactions.

[0033] Transactions can also be defined using event rates. In one embodiment, a transaction is defined by a burst of security events. In other words, a large transaction can be further divided into smaller transactions based on event rate. If some statistical analysis of the event stream or large transaction, such as a hidden Markov model, indicates that the event stream is bursty, each burst can be filtered into a transaction.

[0034] Transactions can also be defined by event fields. In one embodiment, events originating from the same source address (e.g., source IP address) are defined as transactions. Thus, each such transaction contains all events from one IP address within the timeframe of the event stream.

[0035] The transaction parameters can include both source and destination addresses. Thus, some transactions can be defined as all events within a timeframe having the same source and destination IP address (or IP port or both). Furthermore, some transactions can be defined as events exchanged between two participating source/target address (or port) pair. In such an embodiment, security events from address A to B and from B to A would be included in one transaction.

[0036] In one embodiment, transactions do not contain actual security events, but items representing security events. One item can represent many identical security events. In this

manner significant data can be saved. In one embodiment, each item indicates how many times each security event is being represented. In other embodiments, each security event is merely represented by a single un-numerated item in the transaction.

[0037] In one embodiment, the transaction definitions are not mutually exclusive. That is, all of the transaction definitions described above and further below, and any others not described, can be implemented on the same set of security events.

[0038] In one embodiment, the transactions are defined by a plurality of transaction filters 34. The ellipsis in Figure 2 indicates that the security events can be filtered into any number of transactions, and are not limited to the four transaction filters 34 shown in Figure 2. The transaction filters 34 can be configured to produce any possible transaction discussed above. For example, transaction filter 34a may produce time-based transactions, while transaction filter 34b may produce event field-based transactions.

[0039] In one embodiment, the transaction filters also convert the security events into items. As discussed above, an item is defined, in one embodiment, as a description of an event type. When a transaction filter 34 is filling a transaction, duplicate security events may be discarded, or an item count increased, as discussed above.

[0040] In one embodiment, transactions can also be "clustered" to generate new transactions. For example, the transaction filters 34 can be configured to cluster two overlapping transactions. Thus, e.g., if one transaction collects items from source A to destination B, and another transaction collects items from source B to destination C, then a third transaction can include items traversing from A to B to C. Other transactions can also overlap. Such clustered transactions can be useful for detecting transactions representing the spreading of worms, and other attacks where destination addresses become source addresses for an attack.

[0041] In one embodiment, the transactions produced by the transaction filters 34 are provided to the pattern identifier 36. The pattern identifier 36 analyzes the transactions to find the patterns in the event stream. One embodiment of the pattern identifier 36 is now described in more detail with reference to Figure 3.

[0042] In one embodiment, a pattern can be defined as a group of security events that occur together, i.e., are related, in some manner. For example, if security events A, B, and C frequently occur close together, then the event group ABC may be a pattern. On the other hand, if A, B, and C do not occur close together in time, but occur with similar frequency over time, the event group ABC can also be a pattern. An example of a known pattern is a series of unsuccessful logins followed by a successful login. Such a pattern occurs when a dictionary attack is used. A slow and low attack would have events that are not close in time, but related by event frequency.

[0043] To identify the event patterns in the security events provided to the pattern discovery module 30, the pattern identifier 36, in one embodiment, first builds a transaction tree from the transactions provided to it by the transaction filter 34 using tree builder 38. In one embodiment, the tree builder 38 first orders the items in each transaction by item support.

[0044] In one embodiment, this is done by counting the number of transactions in which each item occurs. This number is sometimes referred to as the "support" of an item. In one embodiment, items need a minimum number of support to stay in the transactions, otherwise they are deleted to reduce the size of the transactions.

[0045] In one embodiment, the tree builder 38 then organizes the transactions into a tree structure based on the support of each security event. One process for building the transaction tree is now described with reference to Figure 4. The process illustrated in Figure 4 operates iteratively one transaction at a time. In one embodiment, the transactions are organized by support from the highest to the lowest supported item.

[0046] In block 402 a transaction is selected. In one embodiment, the tree has an empty root node. All other nodes represent items. In block 404 the root node is selected, and in block 406 the next item in the selected transaction is selected, which, in the initial iteration is the first (i.e. highest support) item.

[0047] In block 408 a decision is made whether any children of the selected node (the root node in the initial iteration) represent the same item as the selected item. If the answer is yes, then this node becomes the selected node in block 410, and the support of the selected item is added to the current support of the node. However, if the answer is no, then a new child node of the selected node is created to represent the selected item in block 414, and this new node becomes the selected node in block 416.

[0048] Next, in block 412, a determination is made whether the selected transaction is fully processed, that is, whether it contains any more items. If the transaction is not complete (more items in transaction), then the processing of the selected transaction continues at block 406 with the selection of the next item in order of support. However, if the selected transaction has been fully processed (no additional items), then the next transaction to be processed is selected in block 402 and the processing continues according to the flow chart in Figure 4 until all transactions are consumed in the tree building process.

[0049] Again with reference to Figure 3, in one embodiment, the tree builder 38 provides the transaction tree to the tree parser 40, which extracts the patterns of interest from the transaction tree. In one embodiment, the tree parser 40 traverses the tree branches looking for significant drops in support. For example, if one branch of the tree – shown as Item Name(support) – is A(200), B(190), C(50), D(45), then there is a significant drop in support between item B and

item C. In this case, the tree parser 40 would identify the group AB as a pattern and the group CD as a pattern. The tree parser may also identify ABCD as a pattern as well. In one embodiment, ABCD being a pattern would not mean that the chronological order of these security events is preserved within the pattern.

[0050] In one embodiment, the drop in support is defined by a percentage of decrease. For example, the tree parser 40 defines a drop in support any time support drops by 30 percent or more from one item in a branch to the next. Other percentages can be used as well. In another embodiment, a drop in support is defined by support dropping by a certain number, such as 10.

[0051] In yet another embodiment, a drop in support is detected when the addition of the next item in the branch of the tree would cause the standard deviation of the supports thus far considered to go above a threshold number. For example, if the first three branch items' supports' standard deviation is below 0.03, but the addition of the fourth item would cause the standard deviation to be above 0.03, then the tree parser 40 observes a drop in support between the third and fourth items on the branch. The number 0.03 is only an example; other thresholds may be used. Furthermore, other statistical measures of variation can be substituted for the standard deviation, such as the variance.

[0052] In one embodiment, the tree parser 40, while traversing the transaction tree also looks for associational factors within the tree. For example, if a group of events occurs in more than one part of the transaction tree, the tree parser 40, in one embodiment, declares that group to be a pattern. In one embodiment, this is done by substituting a representation of a newly identified pattern for all its occurrences in the tree and performing pattern detection on the resultant tree.

[0053] In one embodiment, the pattern identifier 36 provides the discovered patterns to the pattern discovery module 30. In one embodiment, the pattern discovery module 30 displays the newly discovered patterns to the user via the console interface 16. The user can make use of the patterns in numerous ways depending on his or her purposes. For example, the user can investigate an event pattern to determine whether it represents a malicious attack or benign activity.

[0054] In one embodiment, the user interface provides the user a rule generation tool that enables the user to convert any of the patterns discovered directly into correlation rules to be used by the rules engine 18. Then, the rules engine 18 will detect and find specific instances of the pattern in all future (or replayed) event streams provided to it. In this manner, the newly discovered pattern can become a "previously known" pattern.

[0055] In one embodiment, the conversion can be provided using a one-click interface. For example, next to each discovered pattern, the console interface 16 can provide a button labeled

"Convert to Rule," which functions to implement the pattern as a correlation rule in the rules engine 18. The user may also be prompted to name the rule.

[0056] In the description above, the pattern discovery module 30 is described as taking security events as input and producing patterns in the security events as output. However, the pattern discovery module can also be configured to detect patterns within patterns. In such an embodiment, the pattern discovery module would treat each unique pattern as a unique element, and perform pattern discovery in a manner similar to finding patterns of events.

[0057] Thus, a network security system having pattern discovery capabilities has been described. In the foregoing description, various specific values and data structures were given names, such as "security event," "transactions," "transaction tree," and "item." Furthermore, various specific modules, such as the "pattern discovery module" and "pattern identifier" have been described. However, these names are merely to describe and illustrate various aspects of the present invention, and in no way limit the scope of the present invention. Furthermore, various modules, such as the tree builder 38 and tree parser 40 in Figure 3, can be implemented as software or hardware modules, or without dividing their functionalities into modules at all. The present invention is not limited to any modular architecture either in software or in hardware, whether described above or not.

Claims

1. A method for discovering an event pattern in an event stream, the event stream comprising a plurality of security events, the method comprising:
creating a plurality of transactions based on a transaction parameter, wherein each transaction represents a subset of the plurality of security events;
generating a transaction tree based on the plurality of transactions, wherein the transaction tree includes one root node and a plurality of non-root nodes, and wherein each non-root node represents a security event; and
parsing the transaction tree by observing drops in support from a first non-root node to a second non-root node, wherein a support of a non-root node represents a number of transactions that include the security event represented by the non-root node.

2. The method of claim 1, wherein the transaction parameter comprises an interval of time, and wherein each transaction represents events that were received during the interval of time.

3. The method of claim 1, wherein the transaction parameter comprises a value of an event field, and wherein each transaction represents events that include the value of the event field.

4. The method of claim 3, wherein the event field comprises a source address.

5. The method of claim 3, wherein the event field comprises a destination address.

6. The method of claim 1, wherein the transaction parameter comprises a point in time, and wherein each transaction represents events that were received proximate to the point in time.

7. The method of claim 1, wherein generating the transaction tree based on the plurality of transactions comprises generating the transaction tree based on a support of a security event within the plurality of transactions.

8. The method of claim 1, further comprising providing a user an option to convert the event pattern into an event correlation rule.

9. The method of claim 1, further comprising creating an event correlation rule using the event pattern.

10. The method of claim 1, wherein the plurality of events had been stored in an event database.

11. The method of claim 1, further comprising removing a field from each event in the plurality of security events.

12. The method of claim 11, wherein a value of the field is unique to each event.

13. A system for discovering an event pattern in an event stream, the event stream comprising a plurality of security events, the system comprising:
a transaction filter configured to create a plurality of transactions based on a transaction parameter, wherein each transaction represents a subset of the plurality of security events;
a tree builder configured to generate a transaction tree based on the plurality of transactions, wherein the transaction tree includes one root node and a plurality of non-root nodes, and wherein each non-root node represents a security event; and
a tree parser configured to parse the transaction tree by observing drops in support from a first non-root node to a second non-root node, wherein a support of a non-root node represents a number of transactions that include the security event represented by the non-root node.

14. The system of claim 13, wherein the transaction parameter comprises an interval of time, and wherein each transaction represents events that were received during the interval of time.

15. The system of claim 13, wherein the transaction parameter comprises a value of an event field, and wherein each transaction represents events that include the value of the event field.

16. The system of claim 13, wherein the transaction parameter comprises a point in time, and wherein each transaction represents events that were received proximate to the point in time.

17. The system of claim 13, wherein the tree builder is further configured to generate the transaction tree based on a support of a security event within the plurality of transactions.

18. The system of claim 13, further comprising a user interface to provide a user a tool to convert the event pattern into an event correlation rule.

19. The system of claim 13, further comprising an event field filter configured to remove a field from each event in the plurality of security events.

20. A machine-readable medium having stored thereon data representing instructions that, when executed by a processor of a pattern discovery system, cause the processor to perform operations comprising:
creating a plurality of transactions based on a transaction parameter, wherein each transaction represents a subset of a plurality of security events;
generating a transaction tree based on the plurality of transactions, wherein the transaction tree includes one root node and a plurality of non-root nodes, and wherein each non-root node represents a security event; and
parsing the transaction tree by observing drops in support from a first non-root node to a second non-root node, wherein a support of a non-root node represents a number of transactions that include the security event represented by the non-root node.

21. The machine-readable medium of claim 20, wherein generating the transaction tree based on the plurality of transactions comprises generating the transaction tree based on a support of a security event within the plurality of transactions.

22. The machine-readable medium of claim 20, wherein the operations further comprise providing a user an option to convert the event pattern into an event correlation rule.

23. The machine-readable medium of claim 20, wherein the operations further comprise creating an event correlation rule using the event pattern.

24. The machine-readable medium of claim 20, wherein the operations further comprise removing a field from each event in the plurality of security events.

25. A method for generating a rule, comprising:
displaying a pattern discovery tool configured to enable a user to select a subset of previously stored security events;
in response to the user selection:
creating a plurality of transactions based on a transaction parameter, wherein each transaction represents a subset of the selected security events;
generating a transaction tree based on the plurality of transactions, wherein the transaction tree includes one root node and a plurality of non-root nodes, and wherein each non-root node represents a security event; and

parsing the transaction tree by observing drops in support from a first non-root node to a second non-root node, wherein a support of a non-root node represents a number of transactions that include the security event represented by the non-root node;

displaying a rule generation tool configured to enable a user to perform an action; and

in response to the user action:

converting a selected newly identified pattern into a correlation rule.

26. The user interface of claim 25, wherein the user action comprises a single mouse-click.