Mining Security Events in a Distributed Agent Society

D. Dasgupta[*a], J. Rodríguez[a], S. Balachandran[a]
[a]Intelligent Security Systems Research Lab, University of Memphis, Memphis, TN, USA 38152

ABSTRACT

In distributed agent architecture, tasks are performed on multiple computers which are sometimes spread across different locations. While it is important to collect security critical sensory information from the agent society, it is equally important to analyze and report such security events in a precise and useful manner. Data mining techniques are found to be very efficient in the generation of security event profiles. This paper describes the implementation of such a security alert mining tool which generates profiles of security events collected from a large agent society. In particular, our previous work addressed the development of a security console to collect and display alert message (IDMEF) from a Cougaar (agent) society. These messages are then logged in an XML database for further off-line analysis. In our current work, stream mining algorithms are applied for sequencing and generating frequently occurring episodes, and then finding association rules among frequent candidate episodes. This alert miner could profile most prevalent patterns as indications of frequent attacks in a large agent society.

Keywords: Agent Society, frequent episodes, alert profiling, event mining, IDMEF, security console, xml feature extraction.

1. INTRODUCTION

In a large agent society, several intentional and unintentional activities can produce security alerts – Intrusion detection alerts. Agent societies are large systems formed by different types of elements such as computers, software elements, and communications elements. These elements can produce a large number of security alarms concerning malicious activity in the agent society. The analysis of the alerts coming from an agent society is a difficult task, due to aspects such as the different types of alerts, different sources of the alerts, different targets, and possibly redundant alerts – one intrusion detection alert can be detected by a different security agent.

A security event in an agent society is originated when a security agent detects the problem. The security sensors are deployed all around the agent society and are configured to look for a particular security issue [6, 15]. Agent societies are tolerant to security attacks, so the main processes can continue working; as a result the user can not realize that a problem is there. A security agent can report messages that belong to its field of action, so these elements have a partial point of view about the complete security scenario. Some times one security alarm in the society can produce numerous alerts. So for example a port scan activity can produce security messages in the computer under analysis as well as in the router that analyses the traffic in the segment of the network where the agent society is running.

The process of security alerts in an agent society is a difficult issue due to the following reasons:

- Different security issues can go on at the same time in the agent society. So for example, malicious activities can be performed by compromised agents, dangerous network activities can be performed by attackers; the users in the agent society can try to exploit vulnerability flows in the infrastructure and so on.
- In some agent security scenarios, the security alerts can occur in a short period of time. As found in the DARPA Intrusion Data Set [7], several thousands of security alerts can appear in just one second.
- The performance of the available tools used to process these security alerts is not always the best. So, for example, if processing a simple security alert takes a few milliseconds, the processing of thousand of alerts that arrive in a short period of time can take a long period of time.

[*] dasgupta@memphis.edu; phone 901 678-4147; fax 901 678-2480; http://issrl.cs.memphis.edu

As reported in [4,14] alarm correlation systems are tools performing operations such as (i) removing alarms that contain redundant information, (ii) removing alerts that contain low priority alarms in scenarios where alerts with high priority are present and finally (iii) substituting a set of alarms by some new information. The objective of such correlation system is to filter out as much information as possible as well to show the agent society operator the useful information in a friendly manner. This final information must point out the possible causes of the security threats going on, along with some useful recommendation as well.

We implemented the frequent episodes and association rule mining algorithms for building alert profiles which provided an overall picture of attack and anomaly patterns in a Cougaar Society. This implementation supports querying the Security Agents, retrieving and accumulating results in XML format, filtering any duplicates and visualizing them in a user friendly manner. Our goal in this work is to ultimately help the security administrator with a friendly tool that will aid in providing a good and proper recommendation for taking appropriate security actions.

In the following sections we highlight the tools, utilities and API for achieving our goal. Then we describe in detail the design of the mining algorithm involved in this work. Section 2 talks about the Cougaar Security Agent Architecture that was used for this implementation. Section 3 gives an overview of the Security Console Administrative tool [8] and the allied utilities. Section 4 deals with data mining functionality and related algorithms. In Section 5, we provide the experiment results with this mining and correlation process over the security alerts generated within the Cougaar Society. Finally we present the conclusions of our work.

2. COUGAAR – OVERVIEW

In very recent times, there have been tremendous breakthroughs in the large Agent-based society applications [1,6,12,21]. These scalable applications are used enormously for planning domains for large scale decision capabilities including defense areas. One of these applications is Cougaar™ [6] which, apart from being a logistic based Agent system, adheres to secure performance attainment. In such huge vulnerable domains, many intentional or unintentional activities occur that are detected by the security sensors generating a large number of alerts. Attacks and anomalies can also occur and be exhibited in multiple places across the society. The system is expected to perform well in highly chaotic environment. The administrator monitoring such large system must ensure proper steps for achieving survivability.

The Cougaar™ software was initially developed under DARPA (Defense Advanced Research Projects Agency) sponsorship under the Advanced Logistics Program (ALP) and is now available as open source. Cougaar is excellent software built on component-based, distributed agent architecture that is survivable and equipped with an automated logistics system. Cougaar societies (Figure 1) are made of Nodes (Java™ Virtual Machines), which contain Agents, which in turn contain components.

Cougaar is a hierarchical architecture where the smallest functional components are implemented as plugins. However, plugins are attached to an agent and a number of agents form a Cougaar Node (as shown in figure 1).

A Cougaar ***Agent*** is an autonomous software entity that has been given behaviors to model a particular organization, business process or algorithm. The agent architecture is based on the human cognitive model of planning. The agents can communicate with one another through a built-in asynchronous message-passing protocol. Cougaar agents cooperate with one another to solve a particular problem, storing the shared solution in a distributed fashion across the agents. Cougaar agents are composed of related functional modules, which are expected to dynamically and continuously rework the solution as the problem parameters, constraints, or execution environment change. All Agents in a given society run the same Cougaar core software baseline, written in Java, though different Agents can contain additional "jars" to give them particular behaviors, model particular entities, or embody particular capabilities. A Cougaar ***Community*** is a notional concept, referring to a group of Agents with some common functional purpose or organizational commonality. Thus a Cougaar society can be made of one or more logical communities, with some agents associated with more than one community and other Agents not associated with any community. Agents are the prime components of the Cougaar architecture. Agent mobility support in Cougaar is the infrastructure mechanism to dynamically transfer an agent from one node to another node, possibly to a different machine. An agent consists of two major components, a distributed blackboard (called Plan) and PlugIns. Each blackboard contains elements such as tasks, assets and plan elements. PlugIns are self-contained software components (compute engines) that can be loaded dynamically into agents. PlugIns interact with the agent infrastructure according to a set of rules and guidelines (as binders), and provide unique capabilities and behavior to complete given tasks. PlugIns bring functionality to the agent, while the society of agents (Nodes) provides structure and an order of operations.

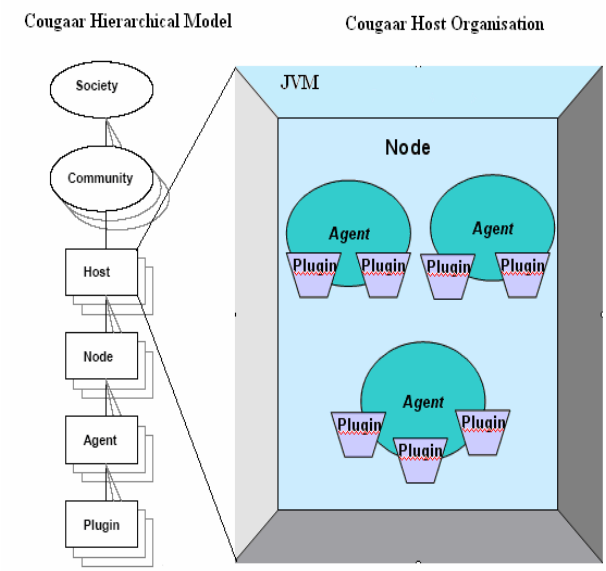

Figure 1 Cougaar™ Society Model

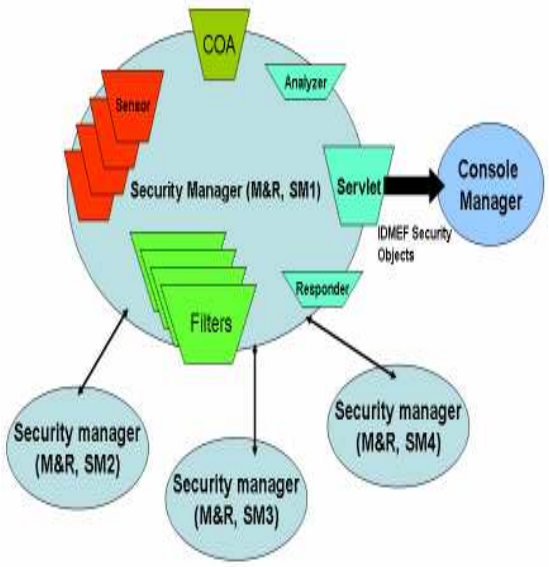

Figure 2. The Monitoring and Response Security Manager setup in the Cougaar Society

Cougaar supports a Web-based Servlet interface and also presents configuration options such as selecting an alternate HTTP port, enabling HTTPS, and enabling client-authentication. The communication among the agents is encrypted, making it secure.

2.1 Cougaar Monitoring and Response (M&R) Service

The Cougaar Agent framework is designed to repel various sorts of attacks by avoiding exposure of communications as much as possible and maintaining information integrity.

The Cougaar security reporting and analysis mechanism is done through Security M&R (Monitoring Response) managers [6,15], which are responsible for providing the management of the entire society's security (Figure 2). These interact with other M&R manager agents to monitor for attacks and report the potential attacks to the SC (Security Console) via the Console Manager. The interaction among managers is done in a hierarchical fashion which supports message aggregation. M&R managers at each level are responsible for their respective domains for detecting potential attacks and aggregating the reported events (generated by the registered sensors under them) as IDMEF Security objects [13] whenever they detect attacks, causing them to change their defense posture.

The Security Manager is equipped with the following specialized Plugins to perform specific operations.

- **Sensors**: Sensors generate security-related events. For instance, a sensor might detect login failures, message failures, or other policy violations. When a sensor is initialized, it registers itself with its M&R manager.
- **Analyzer:** These are components, mainly plugins that subscribe to security events coming from other components such as sensors or other analyzers. These analyzers report to the blackboard alerts that exceeded a threshold level of threat.
- **Servlet**: The IDMEF Security objects are sent to the Security Console Manager by this plugin through the HTTP connection.
- **COA:** Consults the rules list to determine the best course of action for a given attack.
- **Responder:** Perform the actions determined by the COA. The actions include sending an alarm to the security console, updating the security policy, and changing the defense posture and operating mode.
- **Filters:** Each Filter monitors specific events.

Console manager are agents administrated by the security administrator of the Cougaar community. These agents gather information and send it to the user. The data are security events generated by the sensor and analyzer. The Console manager can send queries that search for specific events as well.

3. SECURITY CONSOLE

The Security console (SC) [8] is an administrative tool that assists the user in looking for suspicious events and alerts collected while monitoring the activities of the large multi-agent system. SC is composed of the security console client and the security console manager. Typically we can have one security console manager per security enclave. There can be any number of SC clients. Figure 3 shows a simple society having four Nodes, one SC client and four-console managers each in a different enclave. The Security Console Manager (CM1, CM2, CM3, and CM4) is responsible for receiving and executing commands from the SC. The SC manager supports typical query requests and commands from the SC client for collecting specific alert messages. The SC manager also sends the results of these queries and commands to the SC client. Several SC clients can connect to the SC manager and each SC client is identified by the user id. The SC manager groups all the queries by user id.

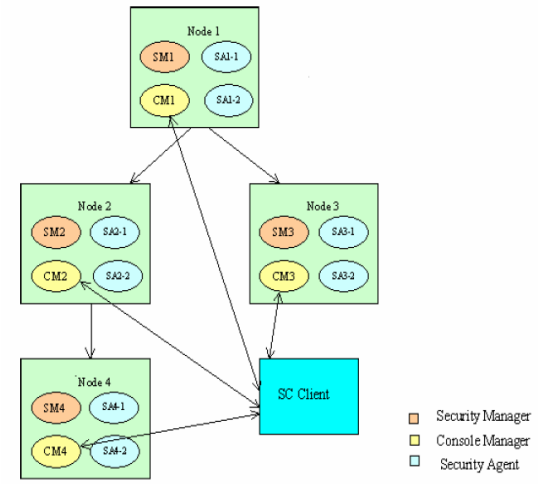

Figure 3 Example Architecture of Security Managers and Security Console Clients.

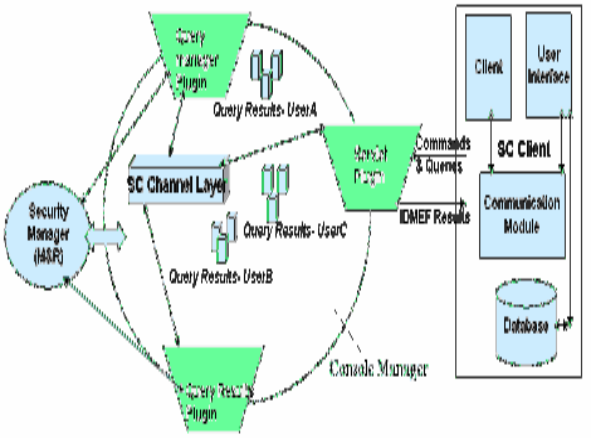

Figure 4. Console Manager Plugin Architecture.

The Monitoring and Response (M&R) Agents and Security Agents (SA1-1, SA1-2, SA2-1, SA2-2, SA4-1, SA3-1, SA3-2) are registered with specific Security managers in each security community. These M&R sensors report events and alerts represented as IDMEF objects to the Security Manager (SM1, SM2, SM3, SM4) within the designated Node. A servlet (via *https* protocol) mechanism is utilized to communicate with the Security Managers, M&R (monitoring and response) residing in the Cougaar society. The activities of the agent environment are monitored through general and specific queries. Polling and *keep-alive* mechanisms are used to get *Transient* and *Persistent* query results. The Cougaar aggregation scheme is involved in consolidating the query results. The summarized results are received in IDMEF with the information like creation time, analyzer, type of messages (heartbeat or alert), etc. These message elements become candidates for correlation mining. The queries can be on demand where the user receives security alerts only when desired. But there is another class of alerts - ***Security Alerts*** that are sent directly by the sensors (without any query), as and when they are generated in the society.

3.1 Security Console (SC) Manager

As shown in Figure 4, the Console Manager communicates with the SC Client on one side and Security manager(s) M&R on the other. The *SC Manager* has three java-based plugins along with the SC channel layer that abstracts the details of the communication protocol.

- **Query Results Plugin:** Sends the results of queries (Query Results UserA, Query Results UserB, so on) in the Manager whenever available. The queries are grouped by the user-id. The results of the queries are sent to the appropriate user. When an SC Client reconnects after a crash, this plugin sends the results of the queries at all the levels below the root level.
- **Query Results** (User A, User B, User C)**:** Query results for User Queries are grouped under the UserId (Unique Identification for the SC user, authenticated by the society). These are aggregated results present as IDMEF objects that the M&R sensor publishes in response to new urgent alerts in the M&R society.
- **Query Manager Plugin**: Receives and processes commands sent from the SC client through the channel layer and is responsible for publishing relays for new queries/expansions.

The queries are hierarchical, that is, there is a root query and then there are expansions and expansions of expansions and so on. The commands are java objects received originally in xml, which are converted to java objects.

- **Servlet Plugin:** Accepts connections from new security console clients and sends/receives information to/from connected SC clients. The information sent/received is done through the SC channel layer object in the blackboard. The servlet implements a keep-alive mechanism to simulate a push operation. This is useful, as the SC manager has to send the results when available.
- **SC Channel Layer**: All the information (commands and results) is sent/received through this blackboard object. It is used to abstract the details of the communication protocol.
- **Security Manager**: This Agent is responsible for managing all M&R components that are part of its security community. The two major capabilities of the manager are :
  - Maintains a distributed dictionary of sensor capabilities. The M&R manager collects registration information from sensors in its security community. This includes the list of sensors and the type of events they can generate. For instance, a sensor might generate IDMEF events to detect login failures for a particular sub-network. The M&R manager builds a dictionary of capabilities, aggregates the information, and sends it to its superior. When the manager sends the capabilities of a security community to its superior, it does not provide all the details of the dictionary. Instead, it builds an aggregated view of the capabilities.
  - Respond to requests by components that wish to query security-related events. These components include Console Manager, M&R filters, or those whose job is to correlate events.

The *SC Clien*t uses the HTTP protocol to communicate with the SC M&R manager via a servlet to send queries (and commands) and receive query results. It performs a low-level communication with the M&R Manager by sending query sentences and receiving IDMEF objects in XML format. It also converts XML IDMEF objects to Java IDMEF objects. The SC Client provides a GUI component that allows editing, storing, retrieving queries, analyzing results and recommending association rules. A central repository, an XML database is embedded to store IDMEF XML messages.

While the SC displays the results, it also archives them for further analysis. The next important task is filtering and fitting to format those events which are highly related and demonstrate some attack/anomaly patterns.

3.2 Security Events as IDMEF Messages

The IDMEF (Intrusion Detection Message Exchange Format) data model [13] is an object-oriented representation of the alert data sent to intrusion detection managers by intrusion detection analyzers. IDMEF provides a standard representation of this information that the intrusion detection analyzer reports. This model helps extensibility via aggregation and sub-classing while keeping the consistency of the model. This model currently defines two classes of objects: *alert* and *heartbeat* messages, which extend from a top-level IDMEF_Message class.

The Security Agents in the Cougaar Agent framework uses the Intrusion Detection Message Exchange Format (IDMEF) as the data formats for communicating with the Security M&R (Monitoring and Response) managers. This helps sharing information of interest between intrusion detection and response systems, and the management systems (like SC), which constantly interact with each other. As defined in [13], two classes of IDMEF messages are implemented in a Cougaar agent society: Heartbeats and Alerts.

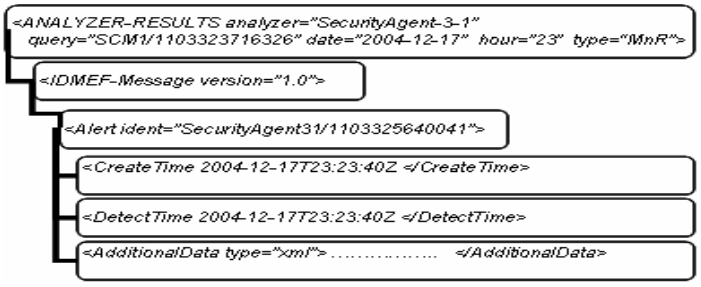

Figure 5 Heartbeat Message

A Heartbeat message reports that the elements which build up the network are alive. These messages can be scheduled, so for example, an analyzer (or other element in the agent society) running in the society must send a heartbeat message each certain time. A typical illustration of a heartbeat message coming from a Cougaar agent society is shown in figure 5.

The analyzer results tag shows information about the analyzer that produces the messages in the society, the query id that matches this heartbeat message, the date and hour of the heartbeat message and finally the type of alert message MnR – Heartbeat message. The alert ID tag shows information about the heartbeat ID – each heartbeat message has a unique ID. The creation and detection time tag shows the time that the alert was created and detected respectively. The additional data tag gives more information about the heartbeat message.

The second type of message is an alert message. These messages are produced by the analyzers in the agent society anytime an analyzer detects a malicious activity. A typical alert message can be seen in Figure 6. The analyzer result tag, alert ID tag, creation tag, detection tag, and additional data tag have information as described in heartbeat messages. The following are other tags in the alert message.

- The analyzer tag has information about the analyzer that detected the message. This tag includes information such as the analyzer's name, IP address, and process. Also the source of the attacks identified by the analyzer. This tag includes other tags with information about the node where the attacker process is running, the user who is running this process, and information about the process itself.
- The source tag shows information about the source of the attacks identified by the analyzer. This tag includes other tags with information about the node where the attacker process is running, the user who is running this process, and information about the process itself.
- The target shows information about the target of the attack identified by the analyzer. This tag includes other tags with information about the nodes under attack, users running in this node, and finally information about the process itself.
- The classification tag shows information about the type of alerts that the analyzer has identified and the URL where the society administrator can check for more information about the alert.
- The assessment tag shows information about the result of the previous evaluation done by the analyzer in order to estimate the status of the attack. The impact tag shows information about the completion and severity of the attack. The action tag describes the actions started by the analyzer in order to address the alert and finally the confidence tag shows the trustworthiness of the analyzer in the alert's evaluation.

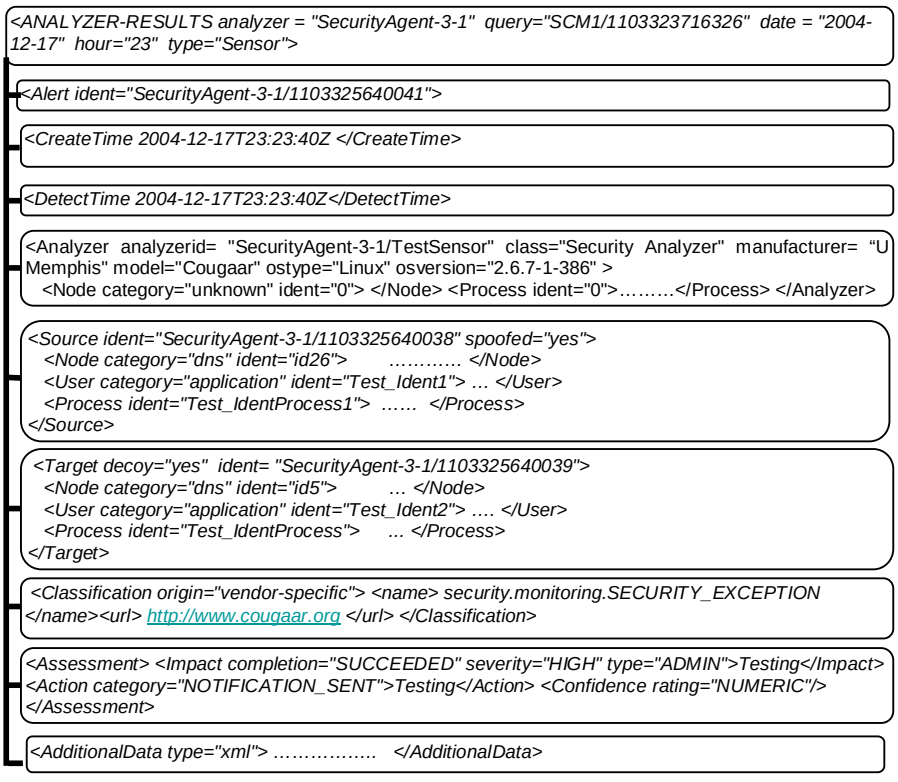

Figure 6 Alert Message

4. MINING SECURITY EVENTS

We have developed a set of data mining algorithms in order to filter out, select and construct association rules from security alerts coming from a Cougaar society. The raw data is preprocessed and summarized into xml records which contain features as described in section 3.2. Data mining algorithms [18] are applied to these xml records to compute frequent patterns which describe the correlation between this record data set.

An alarm correlation system is based in a group of alerts within a short period of time (time window) and interpreting them as a group. The input of the correlation system are the alerts coming from the agent society in a ordered sequence (A_1,t_1), (A_2,t_2), ..of alert occurrences, where A_n represents the security event arriving at the time t_n.

In this paper the goal of the alert correlation system is to discover rules in the form "if a certain security event happens within a short period of time of a given width, then other certain security alarms can also occur within the time window". These "security rules" are interesting for the following reasons:

- The rules are very easy to understand. A rule in the form "If an attacker attacks the process X in the machine XX and the aggressor also attack the process Y in the machine YY", is easy to read.
- The rules can be classified using some grade of frequency in the period of time – window size. The rules with higher frequency are more interesting than those with lower frequency. So frequency can be used like a filter. The security alerts with low frequency are filtered out.
- The rules can build up a "security scenario" about the security alerts happening in the agent society.

4.1 Illustration of Finding Episodes

Consider a window at a given starting position with a given width and a multi set of the alerts types which happen within the window [18]. Denote by r the set of all windows of the given width and denote by R the set of alerts types. The problem is to discover the collection f(r, min_fr) of frequent sets in r with respect to some frequency threshold min_fr. In this way we discover all the events that occur in all the windows with a probably of at least "min_fr". Figure 7 shows the patterns which can occur frequently.The alert type E is followed by the alert type F in (a). In (b) we can say that A and B also occur together but there is not restriction in the order of the events, that is, the events A and B can occur in any order.

In (c) if A and B occur then C will occur with some probability. These frequent occurrences are called patterns. These patterns can be drawn as acyclic graphs. An ordered episode such as in (a) is called a serial episode; an unordered episode such as in (b) is called parallel episode. The episode in (c) is a composite episode, non-serial non-parallel. Once those episodes are known, they can be used to obtain rules. If we know that episode E occurs in 70% of the windows and episode F occurs in the 50% of those windows where E occurs (Serial Episodes), then we can estimate that after seeing a window with E, there is a chance of about 71% that F follows in the same window.

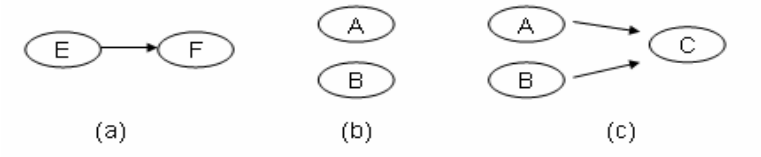

Figure 7 Episodes (a) Serial episodes (b) Parallel episodes (c) Composite episodes

Overall, the complete problem that we are considering is the following. Given an agent society running hundreds of security analyzers, mine all the security alerts coming from the society using frequency episode mining using as input parameters a window width and a minimum frequency threshold.
Given an alert message (IDEMF message) as described in [13], we have to define first what the events are. For the objectives of this study we had defined an event as an object with the following attributes: IP direction of the attacker, IP direction of the target, and finally classification of the attack. Alert messages coming from the agent society have a large number of attributes; different types of association rules can be formed. Our idea is to consider association rules in the form: "Once the attacker X, with IP, IP_1 has attacked the agent identified by the IP address IP_2; there is a probability of about 50% that the attacker Y, with IP address IP_3 will attack the agent with IP address, IP_4". Also the classification of the attack over the agent with target IP address will be shown in the rule.

4.2 Alert Miner

Figure 8 shows a diagram block of each component which builds up the mining component in the SC. As mentioned above, two types of security messages are coming from the agent society: heartbeats and alerts. We are interested mainly in mining alert messages so heartbeat messages are not considered for mining purposes. The components that build up the architecture are:

- **Agent society**: The sensor running in the Cougaar society produces security messages once the SC sends queries to the Security Manager SM. The SM gathers information from other SMs and agents running in the cougar society.
- **Query Manager**: The Query Manager is an SC component which controls the communication between the Cougaar society and the SC. This component sends the command from the SC to the society and receives the messages coming from the Cougaar society. In particular this component receives all the IDMEF coming from the society.
- **Data Base API**: We have developed an API that runs over the eXist DB API in order to provide specific functionalities required by the SC. This API is responsible for storing the IDMEF messages coming from the Query Manager component in the eXist DB and recovering the IDMEF messages from the eXist DB in order to run the frequent mining algorithm.
- **eXist XML DB:** The security alerts coming from the society are stored in the eXist DB in collections[10]. The DB has a root collection (called db). Each security message is stored in a collection called /db/date/Hour/QueryID/AnalizeID/message.xml. Where date represents the creation time of the security message in the society, the hour of creation of the security message in the society, Query ID represents the query ID of the query that was sent from the SC in order to retrieve the message, and finally the analyzer ID represents the name of the analyzer that sent the security message. XPATH [22] is used in order to query all this information from the security message. So the user can store thousands of security messages coming from the agent society in the eXist DB. Each security message builds up an event; the events are used to find out serial and parallel episodes.
- **Data Base Management Component**: Once the security messages are stored in the eXist DB, the user can proceed to select the message logically grouped, in order to perform statistical and mining analysis as well. This task is developed by the Data base management component; and XQuery [5,9] is used in order to perform different queries over the eXist DB.

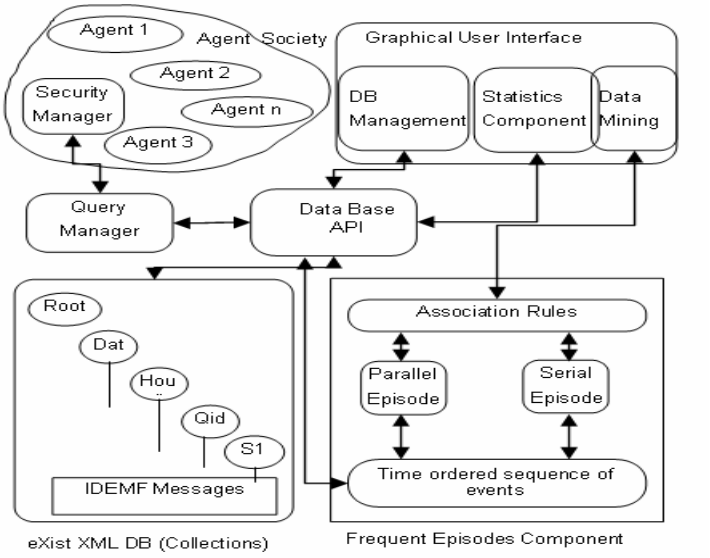

Figure 8 Data Mining Component and Data Flow

- **Statistical component:** This component provides a visualization tool build up by using JFreeChart Library [16] over the security messages selected by the user in the last step. The following are the chars statistics provided by the SC:

- *Distribution of security message in time* (In Hours and Seconds): This graph shows how the security messages are distributed over the time. Time is the horizontal axis and in the vertical axis the number of messages is depicted.
- *Analyzer activity in time* (In Hours and Seconds): This graph shows the number of security messages coming from each analyzer as well as the distribution of the messages in time.
- *Source activity in time* (In Hours and Seconds): This graph shows the number of messages coming from a particular IP attacker as well as the distribution of the messages in time.
- *Target activity in time* (In Hours and Seconds): This graph shows the number of messages that arrive to each IP target-victim and the distribution of the messages in time as well.

- **Data Mining Algorithm:** This component allows to the user to execute the frequent episode mining algorithm. Once the society administrator selects a set of messages, this module sends an XQuery to the Exist DB by using the Data Base API which gathers all the IDMEF messages which match the XQuery. The results are ordered in a time sequence of events. Each event is composed of the IP address of the attacker, IP address of the target and classification of the alert. These are time ordered sequence of events in the input for the parallel and serial episode mining algorithms. The parameter required in order to execute the mining algorithms are the following:
  - *Window Size* – The values here must be Numeric (>0). It represents the size of the sliding window in seconds. Typically this parameter can be between 1 and 60 seconds. It depends heavily on the amount of message coming from the agent society. For example if the messages coming from the society are about some hundred per seconds then 60 seconds in this parameter is probably fine. If the messages coming from the society are even higher, then 5 seconds is fine.
  - *Event Support (%)* – The values here must be numeric. It represents the minimum threshold for an event in order to be considered frequent in the mining algorithm. As the values are to be specified as a percentage.
  - *Episode Length* – The maximum length of Episodes desired is specified here as a positive integer number.
  - *Mining Type* – The type of mining desired could be either *serial* or *parallel.*
  - *Minimum Confidence* – The values must be Numeric (>0). It represents the minimum confidence in the Rules generated and provides the basis of selection from among the rules generated.

- **Mining Component – Association Rules:** Once the frequent episodes and their support are known, this component generates episode rules. These can be generated from frequent episodes in the same way that association rules are generated from frequent sets [2, 11, 18]. Formerly, the frequent episode mining has been executed; the next step is to get the association rules coming from the frequent episodes. Serial and parallel episodes with length greater than or equal to two are used in order to build up association rules which are classified according to two rules: Support and confidence. A typical rule is shown in Table 1.

Table I Security Rules

	Antecedent	***Consequent***
AN	*192.10.10.1*	*192.10.10.1*
TN	*131.84.10.2*	*141.84.10.1*
CS	*Security_Exception*	*Security_Exception*
SU	*50*	
CO	*75*	

AN: Attacker Node TN: Target Node CS: Classification S: Support CO: Confidence

Table II Statistical Results

Distribution of Security Alerts in Time (a)		**Distribution of Security Alerts by Target (b)**	
Time	***Alerts***	***Target IP***	***Messages***
2000-07-03T09:51	54	131.84.1.31	17150
2000-07-03T09:52	4	202.77.162.213	74
2000-07-03T10:08	13	172.16.115.20	43
2000-07-03T10:09	2	172.16.112.10	27
2000-07-03T10:12	9	172.16.112.50	25
2000-07-03T10:13	13	255.255.255.255	6
2000-07-03T10:14	2	172.16.112.105	4
2000-07-03T10:15	11	172.16.113.148	4
2000-07-03T10:16	4	172.16.112.100	4
2000-07-03T10:17	4	172.16.112.194	4
2000-07-03T10:33	25	172.16.115.87	4
2000-07-03T10:34	32	172.16.113.105	3
2000-07-03T10:35	2	172.16.113.50	3
2000-07-03T10:50	38	172.16.112.20	2
2000-07-03T11:27	17154	172.16.113.168	2
		Other IP	13
Total →	17368		17368

5. EXPERIMENT RESULTS

In order to analyze the result of our mining algorithms we have conducted experiments with the 2000 DARPA intrusion detection evaluation data set generated and managed by MIT Lincoln Labs [7] which simulates a distributed denial of service attack. We uploaded this data set in the eXist DB. Each message in the data set basically has the following information: Alert creation time, analyzer that identified the alert, source of the alert and target node of the attack. We filled out the other attributes of the IDMEF with dummy attributes (See figure 6).

Since each event in the mining algorithm is defined as a node source of the attack, node under attack and classification of the attack, each message in the data set provide the two first attributes. The attribute classification was filled out using a standard classification defined in Cougaar. The same procedure was followed for the attribute assessment in the IDMEF message.

5.1 Statistics of Collected events and alerts

Table 2 shows the results coming from the statistical component after preprocessing. Table 2 (a) shows the distribution at the time of the alerts. The table 2(b) shows a typical denial of service attack where the hacker sends thousand of message to the victim in just one second. As shown is Table 2 (b) the attacker sends all the packets to a particular node in the network – in this case, the machine with IP address 131.84.1.31 (first row in table 2b). This statistics module is very useful tool, since it allows to the security officer in the society to have a "fast view" of the security activity in the society. Particularly the security officer can send queries to eXist XML database about event activity in time sequences, nodes under attack, and security sensors with high activity. The security officer can send these queries over a certain period of time.

5.2 Frequency mining results

Figures 9 (a) and (b) show the result coming from the frequent mining episodes component. The panel in (a) shows the input and output values of the frequent mining episode algorithm. The left upper panel shows the sequence of event inputs to the mining algorithm. This panel also shows the starting and ending time of the alert sequence.

The right upper panel shows the input parameters of the frequent mining algorithm. In this case we have specified a window size of 1 second and event support of 0.05%, the confidence specified for the association rules is 75 % and the type of mining is parallel. The results of the mining algorithm are shown in the lower panels. The lower left panel shows the episodes length 1, 2 and 3 found by the frequent mining algorithm. As presented above, each episode is composed by one or more frequent events. The episodes are grouped in order to make the visualization better. The middle lower left panel shows the episodes with its respective support in a char fashion.

This GUI allows the administrator to easily identify which are the episodes with higher frequencies and the events that compose the frequent episode as well. The lower left panel shows in detail each of the events which compos the episode. In this case the parallel episode is composed by tree events. Each event shows the attributes that contain: IP Source, target and classification. The security officer has the option to save the frequent events.

The following is a typical output from the mining module:

This xml output shows an episode with support 2.2523 % composed of two events: Event 1 and Event 2. Event 1 and 2 tell us that the communication between the machines 172.16.112.194 and 202.77.162.213 is very frequent among other events. The panel in figure 9 (b) shows the association rules found starting from the frequent episodes. The panel shows 25 association rules. For each association rule, the boxes in the back show the confidence and the support in this order. The boxes in the "ground" represent the antecedent and consequent of the rules. Finally each consequent and antecedent has a label (partially showed) which identifies each event which builds up the association rule. The following xml output shows a typical association rule produced by the mining module.

```
<Episode Support="0.02523">
  <Event1>
    <Source ident="172.16.112.194" />
    <Target  ident="202.77.162.213" />
    <Classification ident="SECURITY_EXCEPTION" />
  </Event1>
 <Event2>
    <Source ident="202.77.162.213" />
    <Target  ident="172.16.115.20" />
    <Classification ident="SECURITY_EXCEPTION" />
  </Event2>
</Episode>
```

```
<RULE>
 <LHS>
    <EVENT   Classification="SECURITY_EXCEPTION"
Source       = "172.16.112.194"   Target = "202.77.162.213" />
 </LHS>
 <RHS>
   <EVENT Classification = "SECURITY_EXCEPTION"
Source = "202.77.162.213"  Target = "172.16.112.10" />
 </RHS>
 <CRITERE name = "Support" value = "0.02523" />
 <CRITERE name="Confidence" value="1.0" />
</RULE>
```

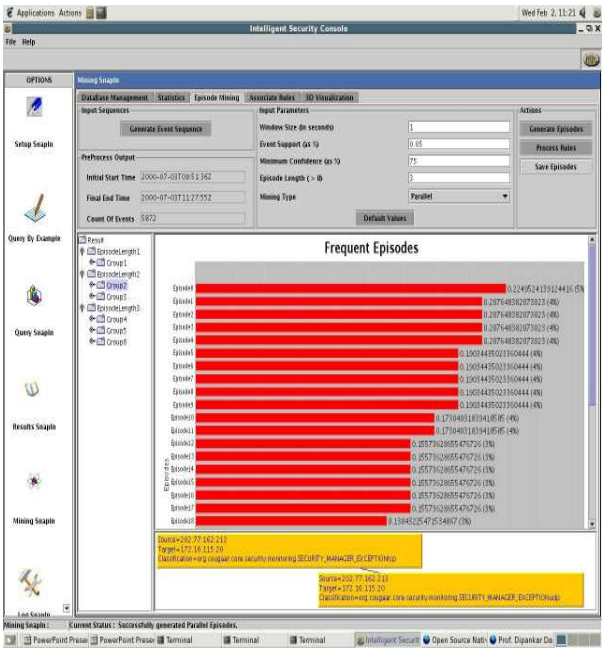

Figure 9a Snapshots of security event mining profiles - Frequent Episodes Algorithm Input - Output

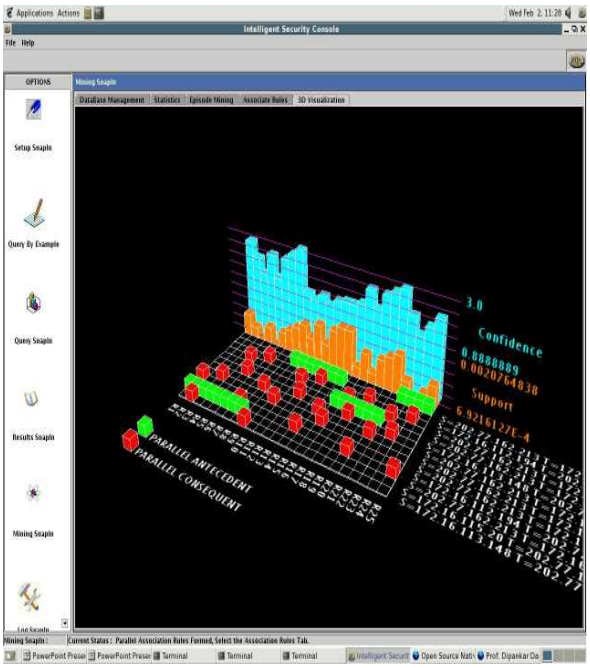

Figure 9b Snapshots of security event mining profiles - Association rules GUI

This association rule has a support of 2.523% and a confidence of 100%. Two events build up the association rule. The fist event LSH represents TCP/IP messages from 172.16.112.194 to 202.77.162.213 and the second event RHS represent TCP/IP messages from 202.77.162.213 to 172.16.112.194. So, this association rule point out that the network traffic between these machines is very high.

6. CONCLUSIONS

The application of stream Data Mining Algorithms for mining the Security events generated across the Cougaar agent society has been the focus of this work. In this work, frequency episode mining and association rule correlation algorithm have been implemented for finding and understanding the alert event patterns in the Agent Society. These correlations among highly related event attributes help to formulate rulesets which help in filtering out alerts that are of low priority or contain redundant information. The *confidence* and *support* parameters are tunable to get desired results. For instance, low support value indicates more event candidates for correlation while higher confidence value results in lesser but highly correlated event episodes. To test the effectiveness and efficiency, we conducted extensive experiments on the benchmark dataset of IDMEF alerts, Darpa2000 dataset from MIT Lincoln labs.

In this work, the system level tool, the Security Console, and the application specific repository, an XML Database were utilized for performing the complete analysis. These provide the ability to choose from some pre-defined set of queries as well as the capability to formulate new queries for security related critical events. The security event miner is developed with individual components working in a highly correlated manner. All the components are fused within this architecture to achieve different related functionality, mainly data analysis, data repository and efficient visualization.

Our development work with this correlation process mainly dealt with the process of analyzing security alerts in an agent society and addresses the core issues with security tools. One common issue was the proliferation of several different security issues at the same time. The major issue was the occurrence of several thousand security alerts in just one second in a scenario such as a Denial of Service Attack. This is a challenge to the Security Administrator as well for the smooth functioning of the mining process. Without detailed preprocessing, such events could severely hamper the algorithm's performance.

To achieve considerable efficiency and correctness, Security Console miner has been designed to filter out as much redundant and low priority alerts as possible and the final information points out the possible causes of the security threats in a user friendly manner with proper recommendation to the Security Administrator.

ACKNOWLEDGEMENTS

This work was supported by the Defense Advanced Research Projects Agency (# F30602-00-2-0514). The views and conclusions of this work no way reflect the opinions or positions of the Defense Advanced Research Projects Agency or the U.S. Government. We would like to thank other ISSRL members F. González, J. Gómez and K. Yallapu who also contributed in various stages of the project.

REFERENCES

1. *Agent Software.* Available at http://www.agentlink.org/ resources/agent-software.php.
2. Agrawal, R. Srikant, R. *Fast Algorithms for Mining Association Rules*. Proc. 20th Int. Conf. Very Large Data Bases, VLDB, 1994.
3. Allen, J. Christie, A. Fithen, W. McHugh, J. Pickel, J. Stoner, E. *State of the practice of Intrusion Detection technologies.* Carnegie Mellon University / Software Engineering Institute Technical Report CMU/SEI-99-TR-028, January 2000.
4. Almgren, M. Lindqvist, U. *Application-Integrated Data Collection for Security Monitoring*. Recent Advances in Intrusion Detection. Springer, Davis, California. October, 2001.
5. *An XML Query Language.* W3C Working Group. Open recommendation. Available at http://www.w3.org/TR/xquery.
6. Cougaar. *A Cognitive Agent Architecture*. Available at http://www.cougaar.org.
7. DARPA. *Intrusion detection data set 2000*. Available at http://www.ll.mit.edu/IST/ideval/data/2000/LLS_DDOS_1.0.html.
8. Dasgupta D, González F, Gómez J, Yallapu K. *An administrative tool monitoring a distributed agent society*. Open Cougaar Conference New York July 20 2004.
9. *Document Object Model (DOM) Level 2 Core Specification.* Version 1.0, W3C Recommendation, 13 November, 2000, http://www.w3.org/TR/DOM-Level-2-Core.
10. *eXist XML DB.* An xml data base. Available at http:// exist.sourceforge.net.
11. Helmer, G. Wong, J. Honavar, V. Miller, L. *Automated Discovery of Concise Predictive Rules for Intrusion Detection*. Department of Computer Science, Iowa State University, Department of Computer Science 2000.
12. Helmer, G. Wong, J. Honavar, V. Miller, L. *Lightweight Agents for Intrusion Detection.* Department of Computer Science, Iowa State University, Department of Computer Science 2000.
13. *Intrusion Detection Message Exchange Format. Internet Draft*. Available at http://www.ietf.org/internet-drafts/draft-ietf-idwg-idmef-xml-12.txt.
14. Jakobson, G. Weissman, D. *Alarm Correlation*. IEEE Network Magazine, Nov 1993.
15. *Feiertag R*. Rho J. Rosset S. Using security mechanisms in Cougaar. Cougaar Software Inc. February 2004, Available at http://securitycore.cougaar.org.
16. *JFreeChart Library*. Available at http://sourceforge.net/ projects/jfreechart.
17. Lippmann, R. Fried, D. Graf, I. Haines, J. Kendall, R. McClung, D. Weber, D. Webster, S. Wyschogrod. D, Cunningham, R. and Zissman, M. *Evaluating Intrusion Detection System*. Proceedings of the 2000 DARPA Information Survivability Conference and Exposition. (DISCEX), Vol 2, IEEE press, January 2000.
18. Mannila, H. Toivonen, H. Verkamo, A. *Discovery of Frequent Episodes in Event Sequences*. Department of Computer Science, University of HELSINKI. Series of Publications C. Report C-1997-15.
19. *The Apache XML Project*, http://xml.apache.org.
20. Wenke, L. Stolfo, S. Chan, K. Eskin, E. Fan, W. Miller, M. Hershkop S. Zhang, J. *Real Time Data Mining-based Intrusion Detection. Proceedings of DISCEX II.* June 2001.
21. Willmott, S. Bonnefoy, D. Constantinescu, I. Thompson, S. Charlton, P. Dale, J. Zhang T. *Agent Based Dynamic Service Synthesis in Large-Scale Open Environments: Experiences from the Agentcities Testbed*. Proceedings of AAMAS'04, July 19-23, 2004, New York, New York, USA.
22. *XML Path Language (XPath) 2.0.* W3C Working Draft 29 October, 2004. Available at http://www.w3.org/ TR/ 2004/ WD-xpath20-20041029.