Command-driven Decentralized Event Processing Approach for Monitoring Networked Android & Windows Devices

Sirojan Tharmakulasingam, Nirushan Rathakrishnan,Nirosh Jayaratnam, Jeyatharsini Jeyaganeshan, Gihan Dias
Department of Computer Science and Engineering, University of Moratuwa.
Kadubedda 10400, Sri Lanka.
{sirojan.11, nirushan.11, nirosh.11, jeyatharsini.11, gihan}@cse.mrt.ac.lk

ABSTRACT

We live in an era where we rely on devices for most of our activities. Organizations use huge amount of devices for their business operations and activities. Those devices are used by different types of personnel where there are no control over their proper usage. In order to ensure their proper usage, the devices should be monitored. Most of these devices used in organizations are connected via network. Monitoring of networked devices require certain amount of resources from devices and network bandwidth based on the transmission of data. Our research mainly focuses on monitoring the networked devices efficiently in terms of required resources and bandwidth. We use decentralized event processing approach in which partial event processing (command-driven, lightweight processing) happens at the devices and remaining processing (complex event processing) happens at the central node where events from all devices are collected. Major objective of command-driven lightweight processing on the devices is to truncate unwanted events for current context of monitoring in order to save the required bandwidth and resource utilization of devices. This paper presents our implemented system for monitoring Windows & Android devices based on this approach and achieved gain in resource utilization and bandwidth.

KEYWORDS

Event Monitoring, Complex Event Processing, Agents, Siddhi, Apache Thrift, Android Devices, Windows Devices

1 INTRODUCTION

Nowadays number of devices in use at organizations and business institutions are exponentially increasing. Each of those devices generates huge collection of events based on the intervention with operator of that device. If an organization wants to ensure the proper usage of their devices, they need to monitor devices by analysing events that are generated by those devices. Since it is not feasible to monitor all the devices separately, organizations need a centralized controller to monitor and control the devices.

The research experiences with monitoring systems over the years show that there is a danger of turning monitoring systems into databases. It seems that collecting and sending events to a central server in the system is often done without analysing whether the data is relevant to the current context of analysing or not. There are some previous researches based on some static pre-processing techniques which are not aware of current context of monitoring. As a consequence, the event processing system requires a huge amount of processing power and network bandwidth for transmission. It also loses its function of providing meaningful information to the current monitoring task. This means that the system needs to be designed in a way that the event data is systematically filtered, collected, checked, aggregated, and used according to the current demand. Experiences have shown that it is very important to develop such a system in a participatory way, meaning that the outlines, the procedures of the system should be agreed upon by international standards.

The major challenge in implementing such a system is heterogeneity in devices. Devices can be varied based on their hardware, architecture and operating systems. The proposed solution in this paper uses separate agents to run on heterogeneous devices. In general, most of the devices generate huge amount of events with high transaction rates. Therefore huge amount of resources such as memory, processor cycles, and high network bandwidth are needed for storing, processing, and exchanging information between devices and central node. To overcome this issue, agents are used in the proposed solution which are responsible for collecting event data from devices on which they are running and performing command-driven lightweight event processing. These agents should not impose load on the devices. In addition to that, agents should be bandwidth-efficient. Since we are using decentralized event processing approach, agents perform partial, command-driven, lightweight event processing on the devices in order to reduce the load on the devices as well as required bandwidth to send the processed event to central node for further complex processing. As the result of complex event processing at central node, unintentional activities, policy violations and potential threats can be detected.

Our event monitoring system Hydra has been implemented based on the aforementioned approach. Agent and Central Node are the two major components of our system. Agent runs on the device, collects data and event logs, and performs some lightweight processing based on the commands given by Central Node. Since the scope of this research is limited to Windows PCs and Android devices, we have developed two types of agents: Windows agent and Android agent. Central Node acts as a controller of agents and it is also capable of doing complex event processing in order to trigger real time alerts.

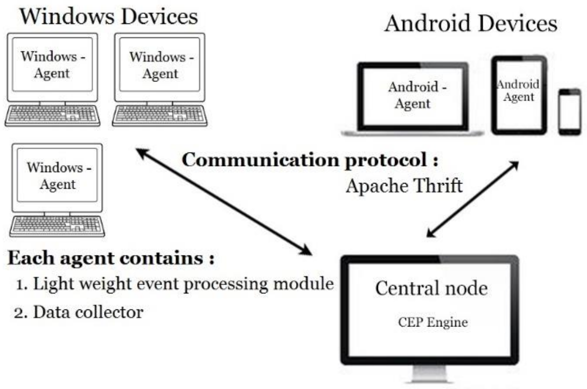

Figure 1. Our Solution

The rest of the paper is organized as follows. Section 2 discusses about some important data & events that can be collected for monitoring of Android Devices. While section 3 discusses about data collection & log collection from Windows devices and outlines some useful events that can be used for monitoring those devices, section 4 describes our command driven light weight processing approach that we used in our Android agent and Windows agent. Section 5 provides the outline of our central node implementation details. Section 6 contains the discussion about communication protocol that we used to establish communication between agents and central node. Section 7 summarizes the experimental results of the implemented system and presents some sample scenarios where our system can be used. The final section 8 provides an outlook to future directions of this research.

2 ANDROID DATA COLLECTION

Mobile devices usage is rising exponentially in today’s business context. Google, Apple, Research in Motion (RIM), and Microsoft are the major players in the mobile device market. A survey says that, Google Android ranked as the top smartphone platform with 82.8% market share in 2015 [1]. Research In Motion (RIM) is an exception as corporate customers can deploy a BlackBerry Enterprise Server and setup their devices to send mobile data to the central server so that central server will collect and do an analysis. Most of the organizations consider transitioning from RIM to new smartphone systems which in turns strengthen the requirement of a monitoring system for mobile devices.

As per the emerging requirement in monitoring the mobile devices, first requirement is to collect the important events and data from mobile devices which could be useful in monitoring. In our research, we limited our scope to Android devices since they cover a major portion in mobile device usage.

Application installation/removal, browser navigation, browser search, calendar event, call log, contact list, device accounts, device ID, GPS location, MMS, picture gallery, screen lock status, SMS and third-party application logs are the events collected for Android forensics [2].

In the latest Android versions (Jelly Bean or higher), third party applications are not allowed to access Android log files without root access. In addition to the above-mentioned data, we collected running processes list and their resource utilization such as CPU usage, RAM usage, and network usage. Along with that, we also focused on collecting the available sensor data. All these collected data will be sent through a lightweight processing module which is controlled by the commands given by the central node. After the partial processing, if the network connection is available then partially processed data will be sent to the central node. Else, it will be locally stored in a SQLite database. Locally stored data will eventually reach the central node when the connection becomes available. This approach prevents event losses if there is a network interruption between agent and central node.

3 WINDOWS DATA COLLECTION

In Windows, performance counters can be used to collect information about the performance of operating system, applications, services, and drivers. There are roughly one thousand performance counters that together reflect the current state of the system. Those performance counters can be accessed using Windows registry API. Since working directly with the registry is too complex, Microsoft provides a more abstract API called Performance Data Helper (PDH) which can be used to access performance counters. PDH is responsible to access the performance counters in the registry and the conversion of their raw values into appropriate numbers.

The registry collects values from performance counters using kernel and makes them accessible directly or using Performance Data Helper (PDH) library. A research team has built a system named WatchTower using PDH. WatchTower is a system that simplifies the collection of Windows performance data for monitoring and usage profiling of Windows machines. Their approach towards this large amount of data is to treat it as a dimensionality reduction problem, where each counter corresponds to a dimension [3]. The major problem of this approach is that only performance data is considered to build a monitoring tool and their dimensionality reduction technique is static. The dimensions are already predefined without the awareness of real-time monitoring task. Our system Hydra collects Windows event logs in addition to performance data such as running processes, and their CPU, memory, and network usage, total CPU, memory, and network usages using PDH library for precise monitoring. Collecting performance data using PDH library is reliable and less overhead. In contrast to their approach, our solution is dynamic (context-aware) which uses command-driven processing. Because of this dynamic nature, agents only send the relevant data to the central node. This increases the accuracy of monitoring tasks as well as improves the efficiency of the system in terms of resource utilization.

3.1 Windows Log Collection

Event logging is significant to detect errors, to find out the cause behind the error, and to prevent the error from recurring. The event logging service receives events from various sources and stores them in a single collection called an event log [4]. Monitoring and analysing of event logs should be automated to make system administrators' life easy since the number of Windows event logs has grown over the years [5].

Windows provides facility in Event Viewer to setup own Event Log Notification System for automation to export and to filter log entries and then to email or save it in a text file. It is inadequate for monitoring large size network because it only supports limited static functionalities. Since it is configured using static scripts, there is no awareness about current monitoring task in the process of log collection. This leads to a chance that irrelevant logs for the current analysis also get collected and it will be sent through the network. Because of this, bandwidth usage of system is high and human intervention is heavily required for detailed analysis. In contrast, our solution tries to minimize the required bandwidth and human intervention by having command-driven context-aware log collection and complex event processing techniques such as pattern matching respectively.

There are two identified alternatives to collect logs from Windows. Logs can be collected in binary format from unallocated space or using Event Logging API from allocated space. Polling log data at regular intervals from allocated space using programming interface immediately after logging of events is preferred over getting logs from unallocated space. Because getting logs earlier helps to predict some bad outcomes before they occur or at least immediately after their occurrence. So our agent uses Event Logging API to automate the process of collecting of events.

Centralized collection of log data from Windows PCs is important because processing event logs on local machine is not safe due to intensive or non-intensive failures of local machines. P. K.Sahoo, R. K. Chottray and S. Pattnaiak [6] have proposed a solution to centralize event logs. Their system retrieves Windows event logs, translates them to Syslog format, and sends to a central server. It stores in a database after processing them based on a set of rules that specified in the Winsyslog configuration. Syslog messages can be displayed by Windows GUI and reports are generated automatically based on data from database by "monitor ware console". Above solution requires more bandwidth as it sends all the logs without doing any processing in order to reduce its size. In their research, centrally collected logs are only used to generate some reports regarding the statistical information of the collected logs. However, those collected logs can be utilized to detect unintended activities and any kind of policy violations by doing further processing. In our solution, we process those collected logs partially on agents based on central node commands and then partially processed logs are transferred to central node for complex event processing in order to detect anomalies.

Stephan Grell and Olivier Nano [7] have implemented a system to monitor large scale internet services using central node with Complex Event Processing Engine(CEP) as it is able to do fast and real time in-memory processing of events (filtering, grouping and aggregating) as long as resource consumption are kept within limits. In our solution also, a CEP engine is used at central node for complex event processing and lightweight processing engines are used in distributed agents. We limit the resource consumption of distributed devices by switching central node commands based on resource availability of networked devices. For instance, once the remote device is running out of resources we skip processing on that device and do entire processing in the central node. In our solution, load is dynamically balanced by central node commands.

Even though events are processed on the distributed nodes in the solution of Stephan Grell and Olivier Nano [7], processing is done without knowing the current demand or context. But in our solution, central node sends command which consists the events to be considered and summarization level to process events in dynamic manner based on the current demand as well as restrictions. Our agents are capable of handling those commands and they can provide data as per those commands. This is the main value addition in our product.

3.2 Useful Events for Analysis in Windows Logs

Each event can be categorized under one of the five event types: error, warning, information, success audit, and failure audit. Events marked as errors and warning are more important than other categories for analysis purpose.

Spotting the Adversary with Windows Event Log Monitoring [8] recommends some important events to be collected that can be helpful in monitoring devices. It identifies some suspicious event IDs related with events of application whitelisting, application crashes, system or service failures, Windows update errors, Windows firewall, clearing event logs, software and service installation, account usage, kernel driver signing, group policy errors, Windows defender activities, mobile device activities, external media detection, printing services, pass the hash detection, and remote desktop logon detection. In our research, we make use of those events while collecting and detecting unusual events.

Russ Anthony [9] talks about some of important observations related with process creation events which can be useful for monitoring devices. Even though process creation seems to be not important due to the high frequency of its occurrence, it is important to identify the process names for long string of empty spaces, misspelled words, and non-standard path in order to detect suspicious processes. Other than that he talks about events related to privilege escalation which is also very useful since this might be an entry step for an attack or violation.

As stated in this section, these are some of the useful events considered in our solution, which are useful in detecting policy violations, intellectual property theft, misuse, and any attack simulations.

4 COMMAND-DRIVEN LIGHTWEIGHT PROCESSING

Processing events in a resource constrained environment is the major challenge which should be addressed here. Since the devices have limited resources and capabilities, a special care is needed in designing of event processing module. Because of this we have designed lightweight, bandwidth-aware and resource utilization-aware event processing modules for agents which can be controlled by commands given by central node. Because of the awareness about bandwidth and resource utilization, event processing plan is switched time to time in the devices based on the resource availability and demand. In order to have this awareness, central node continuously monitors resource utilization of connected devices and sends commands to agents based on that. For instance, event processing on some devices can be completely skipped especially when resource utilization is high in those devices and high-bandwidth is available to transmit data. This kind of dynamic adjustment is possible in our system since we use command-driven approach and distributed event processing between remote devices and central node based on load.

In our agent-based system, agents run on devices that need to be monitored. Agents consist of a data collector designed for a particular platform and a lightweight event processing engine. Other than the agents, we are having central node which is dedicated for complex event processing and event analysis. Since commands are dynamically sent to agents from central node, our lightweight processing modules in the agents need to act based on those commands and get back the results to central node. So our lightweight processing modules are designed in such a way that we can deploy dynamic commands. Based on the deployed commands, our agents collect specified events, perform processing at a specified level, and send those partially processed events to central node for further analysis. Since all the connected agents are fully controlled by central node commands, the output of the agents are very relevant to current monitoring task, and as a result of that, irrelevant resource consumptions can be avoided.

4.1 Command-driven Lightweight Processing on Android Agent

Command-driven pre-processing module of Android agent is written in Java. Since we collect a bunch of data from Android devices as mentioned in section 2 of this paper, there is a demand to reduce the size of data that needs to be transferred to the central node in order to reduce the required bandwidth. A lightweight command-driven module is included in the agent to achieve this demand.

The lightweight processing engine of Android agent is capable of filtering and aggregating data based on the given filtering parameters and aggregation time limit respectively. These parameters are given by central node commands based on the current demand of monitoring tasks. Based on these parameters the partial processing happens on android devices in order to reduce the size of data that need to be transferred, and then partially processed data is transferred to central node for further analysis.

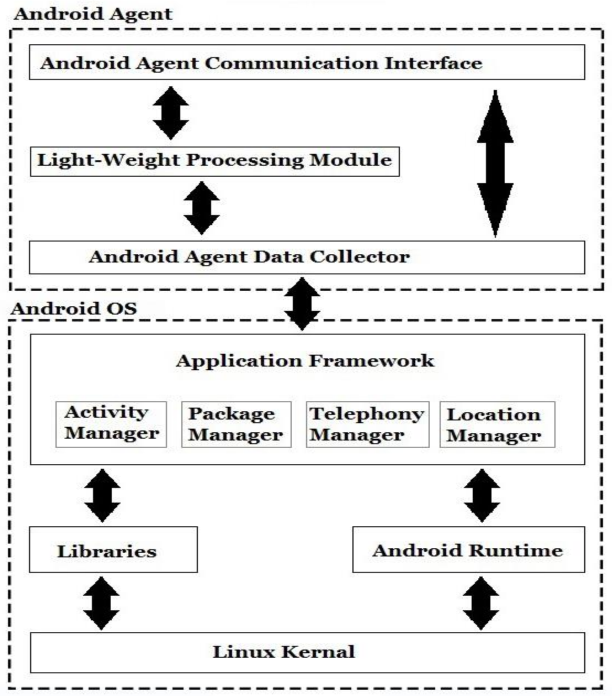

Figure 2. High level architecture of Android Agent

4.2 Command-driven Lightweight Processing on Windows Agent

Command-driven pre-processing module of Windows agent is written in C++. Since C++ is the native language for Windows platform, it is very easy to access the native APIs for event collection. It increases the performance of the agent and reduces load on the device.

The lightweight processing engine in Windows agent also has same features as Android agent in order to process the performance data collected from Windows. In addition to performance data, we also collect log data from Windows devices. In Android devices, we can’t collect logs of third party application due to the restriction in the latest Android versions. Pre-processing of log data is also a prominent task that needs to be done since the size of the log file is large. It will take huge bandwidth to transfer as raw data to the central node. Therefore, only the relevant events which specified in the previous section should be extracted out efficiently from the log files and transferred to the central node.

As per the discussion above, our lightweight processing module of Windows agent can do different levels of summarization. Summarization of event logs uses the information and attribute hierarchy of event logs. Event logs may contain information related to provider, object, subject, network, layer, filter, change, callout, application, access request, rules, errors, processes, logon type, impersonation level, account for which logon failed, failures, new logon and detailed authentication. All of these information have their own attributes. Based on the importance level, some of the information are dropped during high level summarization and some of the attributes are dropped during medium level summarization. Attributes or information to be dropped have been determined from previous researches. Attributes or information to be dropped is determined with the help of previous researches. Attributes or information which are highlighted in the analysis can’t be dropped even in high level summarization. Low informative details can be dropped during the processing on devices since they are less important for analysis. It saves required bandwidth by only sending the content-rich information to the central node.

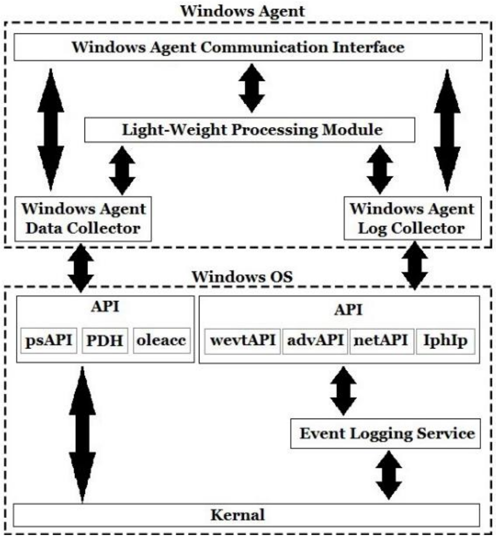
Windows Agent
Windows Agent Communication Interface
Light-Weight Processing Module
Windows Agent Data Collector
Windows Agent Log Collector
Windows OS
API
psAPI
PDH
oleacc
API
wevtAPI
advAPI
netAPI
IphIp
Event Logging Service
Kernal

Figure 3. High level architecture of Windows Agent

5 CENTRAL NODE

This is a central instance which includes a complex event processing engine, registration module for remote devices, and bandwidth plus resource aware command application module. It contains an embedded database where the rules for the complex event processing engine can be persisted. Each module is allocated for their own set of responsibilities. Device registration module keeps track of registered devices as well as the connections regardless of the mobility of those devices. Command application module continuously keeps track of the available bandwidth for each connection and resource utilization of remote devices. Based on those contexts, it switches the commands that are sent to remote agents. Based on received commands, agents crawl data from remote devices and partially process it and send it as an event stream to the central node.

The data stream which is pushed by agents from remote devices is directly fed into CEP Engine in the central node. We found Siddhi [10] and Esper [11] are the two CEP engines which provide required functionalities for Complex event processing. While Esper has restricted some features in commercial license Siddhi is fully open source application. In addition to that, Siddhi performs much better than Esper in terms of throughput [12]. Siddhi also provides CEP query support. We can send events using Apache Thrift [13], web services, Java message service, and emails. Because of these competitive advantages of Siddhi over Esper, we use Siddhi engine for complex event processing in our central node.

Central node is facilitated with user interface to write the CEP rules for the engine as well as to configure the event processing parameters for the agents in order to do the analysis with the objective of detecting policy violations, intellectual property theft, misuse, embezzlement, sabotage, and espionage. By writing custom rules and patterns, device monitoring can be conducted with reduced utilization of resources, which is the major objective of this research.

There are four major alternatives in the event processing commands that are chosen by central node based on the available bandwidth and resources.

		Available Resources in Remote Devices	
		Low	**High**
Available Bandwidth	**Low**	Skip processing at devices, Send predefined critical events only	Allow significant amount of processing at devices
	High	Skip processing at devices and send all events	Allow user defined commands

Table 1. Alternative Event Processing Plans & Conditions

6 COMMUNICATION PROTOCOL

Since this product is most concerned with performance and efficiency, native programing languages are used to develop the agents. Windows Agent is developed in C++ and Android Java is used to develop Android agent. The central node is developed using Java. Therefore, a standard cross platform communication protocol is required in order to establish the communication between the agents and central node for command and data transmissions.

Remote Procedure Calls (RPC) can be used to establish the communications between the agents and the central node. Apache Thrift [13] software framework is used to build RPC servers and clients that will help to communicate seamlessly across programming languages. This enables the server side to be written in Java, when one client is written in C++ to run on windows platform and other client is written in Android Java to run on Android platform. There are several alternatives for Apache thrift such as Protocol buffers [14], JSON-RPC [15] and Avro [16]. In contrast to the Apache Thrift, Protocol buffers doesn’t generate ready to use servers. JSON-RPC has a significant amount of communication overhead than Apache thrift. Error handling and extensibility support are also good in Apache thrift than Avro. These comprehensive advantages of Apache thrift make it fit for our monitoring system.

7 EXPERIMENTAL RESULTS

We deployed our agents in a lab and continuously profiling its resource utilization. The below graphs show the average utilization of resources at the remote nodes by agents.

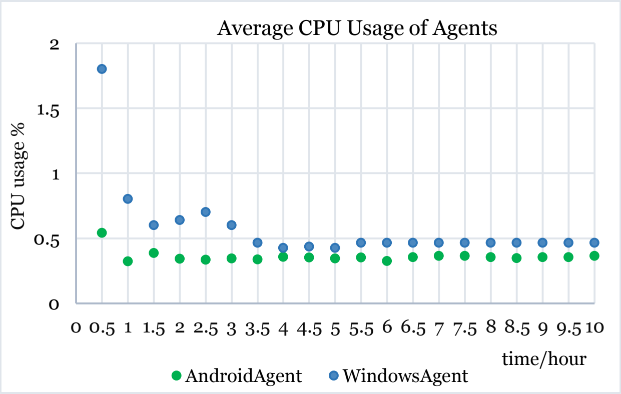

Figure 4. Average CPU usage of agents

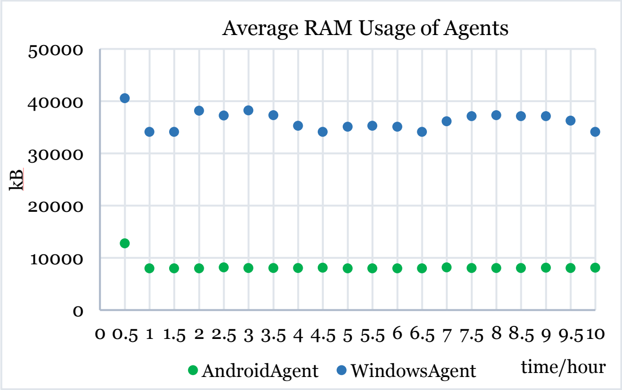

Figure 5. Average RAM usage of agents

CPU & RAM utilization by agents seems to be fair because those are very small fragments of available resources. RAM usage is a bit high for Windows agent since we process both log data and performance data where in Android agent we only consider performance and sensor data.

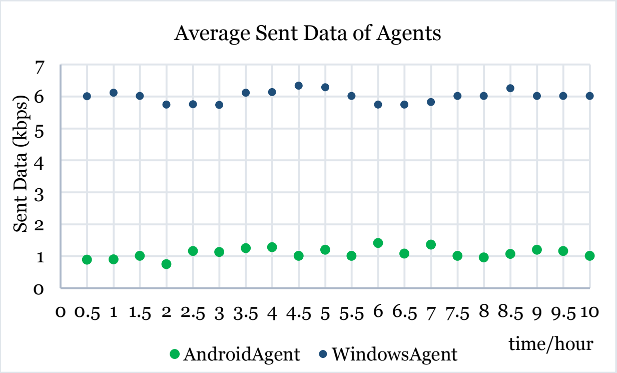

Figure 6. Average sent data from agents

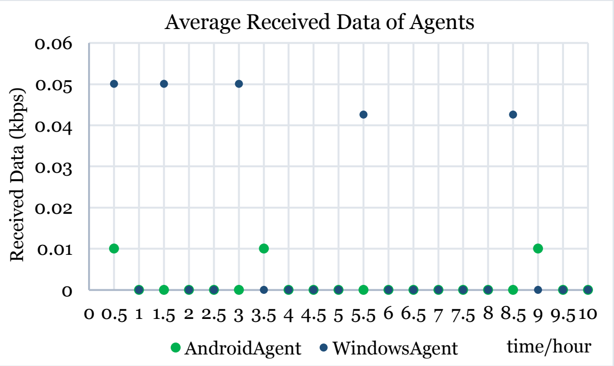

Figure 7. Average received data to agents

Send & received data also seems to be fair because it only requires very little amount of bandwidth compared to other existing systems. Send & received data is a bit high for Windows agent since we send the processed log data as mentioned above. The above graph shows the data reduction due to the partial processing on remote devices. If partial processing is not performed in remote devices, the required bandwidth becomes significantly high due to high data transfer. So, using command-driven decentralized event processing approach gives significant gain in resource utilization and bandwidth consumptions.

We deployed our Windows agent onto 10 Windows PCs at the university lab and monitored important events for 5 days. The graph below shows the statistics.

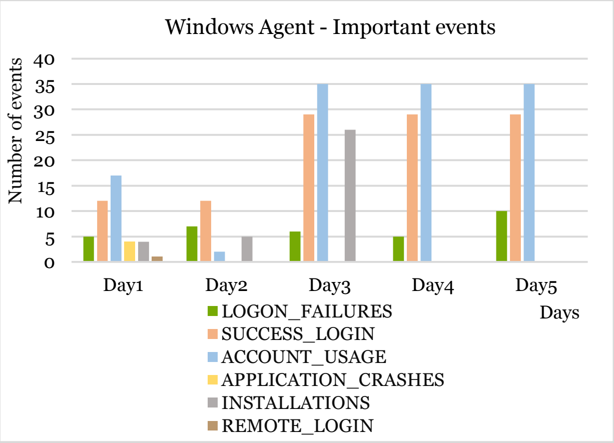

Figure 8. Important events detected

These are some important events which are detected by Windows agent during 5 days. Agents are capable of detecting these kind of important events based on central node commands and transfer them to central node for further analysis. When our system is asked to detect any policy violations or unintentional activities, it uses these important event collection to detect the anomalies.

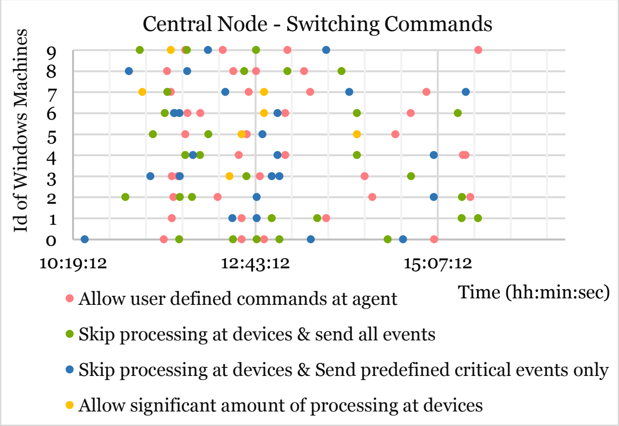

Figure 9. Statistics of Central Node commands switching

The graph above shows the statistics of different commands sent by the central node over time based on the available network bandwidth and resource availability in the remote devices. Since our system itself can dynamically adjust, it could be able to use the available resources efficiently in order to achieve its monitoring tasks.

7.1 Example Application Scenarios

Our system can be used in detection of any attack simulation. We simulated a Denial of Service (DoS Attack) which is an attempt to make a computer resource unavailable to its intended users. In our case, we have detected UDP flood attack using our system. UDP flood is based on sending the overwhelming number of UDP packets to random ports on a remote host. We have used LOIC Tool [17] to perform the UDP flood attack. The tool takes the IP of the target machine and performs the attack. We have mounted the attack on port 80 since firewalls cannot prevent that attack because they can't distinguish good traffic from DoS attack traffic. Our system can detect these kind of attacks by monitoring the network traffic pattern continuously via agents and alert if there is an anomaly detected.

Let's consider another scenario where an online exam is conducted in a university. Students are not permitted to access any lecture notes (via power point slides or pdf documents). In order to monitor any violations, we can simply write a rule in our central node such that if an agent notifies any foreground processes other than one web browser or more than one tab is used in that web browser then our system detects that as a violation of the specified rule and fires a real time alert.

Since complex event processing engine is provided with Event Processing Language (EPL) which is a declarative language for dealing with high frequency time-based event data, we could be able to write customized rules (Organization policies, suspicious event patterns) based on our requirements. Then our system will alert any violations on deployed rules.

8 FUTURE WORK

Future research on this event processing system includes development of agents for other platforms such as Linux, Mac, iOS and IoT (Internet of Things) devices as same as already developed for Windows and Android agents. Those agents should be compatible with the existing protocol. Since our communication protocol is Apache thrift and it supports cross platform communication, the extension of this solution to other platforms will not be a rough task to do.

Machine learning assisted rule generation module can be added in the central node. Since we are having light weight processing engine in the agents and complex event processing engine in the central node, it will be better to have such an automated rule generation module. This will be an additional step up in the journey of automated monitoring of devices in a distributed environment.

9 CONCLUSION

The system developed through this research serves as a prototype for monitoring system in a distributed environment. The major objective of this research is to develop agents which can survive in resource constraint environment and provide the relevant data based on the current context instructed by the central node. From the collected data we could be able to detect some policy violations, attack simulations, and misuse of resources. Since we collect data from native APIs of Windows and Android as much as possible, this research also serves as a guide for accessing the data through native APIs. Presence of complex event processing technology enhances the real time monitoring since it is a convenient technology to process events and discover complex patterns among multiple streams of event data through filtering, grouping, aggregating the event streams. In this Post-PC era, it is very much useful to have such automated monitoring systems to detect the unintended activities.

REFERENCES

[1] IDC Research, Smartphone OS Market Share, 2015 Q2 [Online]. Available: http://www.idc.com/prodserv/ smartphone-os-market-share.jsp.

[2] J. Grover, “Android forensics: Automated data collection and reporting from a mobile device,” Digital Investigation, vol. 10, pp. S12–S20, 2013.

[3] M. Knop, J. Schopf, and P. Dinda, “Windows Performance Monitoring and Data Reduction using WatchTower,” Proceeding 11th IEEE Symp. High-Performance Distrib. Comput., pp. 1–14, 2002.

[4] Microsoft Corporation, Windows API Index [Online]. Available: https://msdn.microsoft.com/en-us/ library/ windows/desktop/ ff818516(v=vs.85).aspx.

[5] M. D. Mullinix, “An Analysis of Microsoft Event Logs”, December 2013.

[6] P. K.Sahoo, R. K. Chottray, and S. Pattnaiak, “Research Issues on Windows Event Log,” Int. J. Comput. Appl., vol. 41, no. 19, pp. 40–48, 2012.

[7] S. Grell and O. Nano, “Experimenting with complex event processing for large scale Internet services monitoring,” Complex Event Processing for the future, 2008.

[8] Network Components and Applications Division, National Security Agency, United States of America, ‘Spotting the Adversary with Windows Event Log Monitoring’. [Online]. Available: https://cryptome.org/ 2014/01/nsa-windows-event.pdf.

[9] Russ Anthony, “Detecting Security Incidents Using Windows Workstation Event Logs,” SANS Institute, June. 2013.

[10] Sriskandarajah Suhothayan, Isuru Loku Narangoda, Subash Chaturanga. “Siddhi-CEP - high performance complex event processing engine,” 2011.

[11] A. Mathew, “Benchmarking of Complex Event Processing Engine – Esper,” 2014.

[12] Sriskandarajah Suhothayan, Isuru Loku Narangoda, Subash Chaturanga. “Siddhi: A Second Look at Complex Event Processing Architectures,” November 2011 ACM 978-1-4503-1123-6/11/1.

[13] Randy Abernethy. The Programmer’s Guide to Apache Thrift. MEAP12.Manning Publications, 2015.

[14] Google Inc, Protocol Buffers: What Are Protocol Buffers? GOOGLE. Google Developers [Online]. April 2, 2012. Available: https://developers.google.com/protocol-buffers/.

[15] JSON-RPC Working Group. (2013) JSON – RPC: Specifications. JSON-RPC Google Group [Online] Available: http://www.jsonrpc.org/specification.

[16] Jim Scott, “Avro – More Than Just A Serialization Framework”, Chicago Hadoop Users Group, April 2012. [Online]. Available: https://vimeo.com/ 40776630.

[17] Verma, Deepanker. "LOIC (Low Orbit Ion Cannon) - DOS Attacking Tool - Infosec Resources". InfoSec Resources. N.p., 2011. Web. 23 Jan. 2016. Available: http://resources.infosecinstitute.com/loic-dos-attacking-tool/