Abhinav Arya; Prof. Arijit Das
November 2017
Naval Postgraduate School

Cybersecurity Analysis through Binary Forensics and Distributed Computing

Overview

Cybersecurity is of utmost importance to the U.S. Navy. With thousands of operational computers and networks, the Navy requires a reliable method of detecting cybersecurity and network vulnerabilities and/or attacks. Fortunately, there is a reliable and accurate method of unearthing these threats to military machines by analyzing system and network logs.

The Windows Event Log

Many Naval computers run on the Microsoft Windows operating system. Windows includes software that monitors network and security alerts and records them in a Windows Event Log. Event logs contain information critical to investigating the history and validating the security of Naval computers.

Microsoft updated the event log file with release of Windows 7 by obfuscating the file structure and encoding the data in a proprietary binary format (.evtx). These event logs can only be viewed through the Windows Event Viewer, a graphical user interface that cannot be parsed by automated systems. In addition, since the event logs are encoded in a binary format, traditional text processing methods cannot be applied, compounding the difficulty of automating event log processing.

Objective

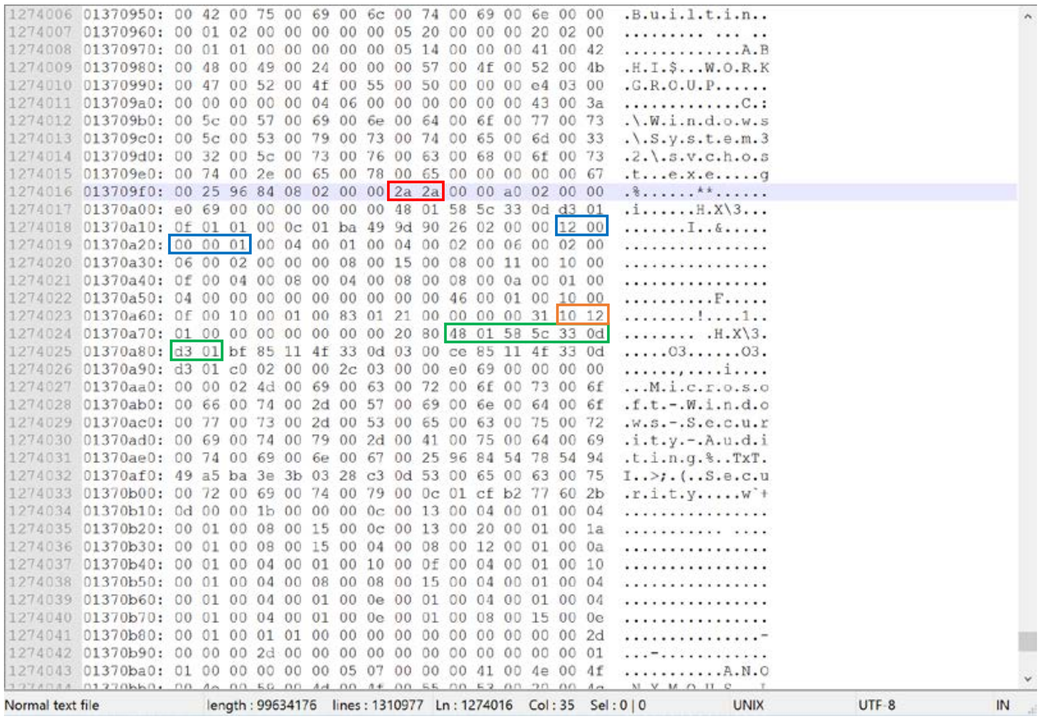
To streamline the process of updating Naval computers and servers to the latest Windows operating system, the Navy desires a software tool to decode the complex binary structure of windows event logs and extract relevant event information corresponding to critical security messages and alerts.

Binary Forensics

As previously mentioned, Windows Event Logs are encoded in a proprietary binary structure that prevents someone or some program from easily accessing the valuable information stored within them. Fortunately, research conducted by the Digital Forensics Research Conference (DFRWS) began to decrypt the binary structure of event logs.

According to Andreas Shuster's research paper presented at DFRWS, *Introducing the Microsoft Vista Log File Format*, the Windows Event Log is comprised of three sections: the file header, the chunk, and the event record. The chunk is merely a container for all the event records, which house the vital information regarding specific security alerts and events. Navigating to the individual event records is a relatively simple task. The start of every event record is marked by a sequence of two bytes: 0x2a 0x2a, which translates to ** in ASCII. From here, it becomes much more difficult to locate critical information.

The remainder of the event record is encapsulated in a Binary XML structure. It is similar to a regular XML document, except all the opening and closing tags have been converted to binary tokens. Furthermore, the content of the XML is not stored inside the tags, meaning that Windows separates the content (the data regularly sored within XML tags) from the structure (opening/closing tags). Microsoft elected to do this to reduce disk space usage, as the structure of the XML does not have to be replicated for every event in the event record. Therefore, all the information necessary for determining the characteristics of an event is stored separately from the binary XML, in a structure known as the Substitution Array.



```
01370950: 00 42 00 75 00 69 00 6c 00 74 00 69 00 6e 00 00  .B.u.i.l.t.i.n..
01370960: 00 01 02 00 00 00 00 00 05 20 00 00 00 20 02 00  ......... ... ..
01370970: 00 01 01 00 00 00 00 00 05 14 00 00 00 41 00 42  .............A.B
01370980: 00 48 00 49 00 24 00 00 00 57 00 4f 00 52 00 4b  .H.I.$...W.O.R.K
01370990: 00 47 00 52 00 4f 00 55 00 50 00 00 00 e4 03 00  .G.R.O.U.P......
013709a0: 00 00 00 00 00 04 06 00 00 00 00 00 00 43 00 3a  .............C.:
013709b0: 00 5c 00 57 00 69 00 6e 00 64 00 6f 00 77 00 73  .\.W.i.n.d.o.w.s
013709c0: 00 5c 00 53 00 79 00 73 00 74 00 65 00 6d 00 33  .\.S.y.s.t.e.m.3
013709d0: 00 32 00 5c 00 73 00 76 00 63 00 68 00 6f 00 73  .2.\.s.v.c.h.o.s
013709e0: 00 74 00 2e 00 65 00 78 00 65 00 00 00 00 00 67  .t...e.x.e.....g
013709f0: 00 25 96 84 08 02 00 00 2a 2a 00 00 a0 02 00 00  .%......**......
01370a00: e0 69 00 00 00 00 00 00 48 01 58 5c 33 0d d3 01  .i......H.X\3...
01370a10: 0f 01 01 00 0c 01 ba 49 9d 90 26 02 00 00 12 00  .......I..&.....
01370a20: 00 00 01 00 04 00 01 00 04 00 02 00 06 00 02 00  ................
01370a30: 06 00 02 00 00 00 08 00 15 00 08 00 11 00 10 00  ................
01370a40: 0f 00 04 00 08 00 04 00 08 00 08 00 0a 00 01 00  ................
01370a50: 04 00 00 00 00 00 00 00 00 00 46 00 01 00 10 00  ..........F.....
01370a60: 0f 00 10 00 01 00 83 01 21 00 00 00 00 31 10 12  ........!....1..
01370a70: 01 00 00 00 00 00 00 00 20 80 48 01 58 5c 33 0d  ........ .H.X\3.
01370a80: d3 01 bf 85 11 4f 33 0d 03 00 ce 85 11 4f 33 0d  .....O3......O3.
01370a90: d3 01 c0 02 00 00 2c 03 00 00 e0 69 00 00 00 00  ......,....i....
01370aa0: 00 00 02 4d 00 69 00 63 00 72 00 6f 00 73 00 6f  ...M.i.c.r.o.s.o
01370ab0: 00 66 00 74 00 2d 00 57 00 69 00 6e 00 64 00 6f  .f.t.-.W.i.n.d.o
01370ac0: 00 77 00 73 00 2d 00 53 00 65 00 63 00 75 00 72  .w.s.-.S.e.c.u.r
01370ad0: 00 69 00 74 00 79 00 2d 00 41 00 75 00 64 00 69  .i.t.y.-.A.u.d.i
01370ae0: 00 74 00 69 00 6e 00 67 00 25 96 84 54 78 54 94  .t.i.n.g.%..TxT.
01370af0: 49 a5 ba 3e 3b 03 28 c3 0d 53 00 65 00 63 00 75  I..>;.(..S.e.c.u
01370b00: 00 72 00 69 00 74 00 79 00 0c 01 cf b2 77 60 2b  .r.i.t.y.....w`+
01370b10: 0d 00 00 1b 00 00 00 0c 00 13 00 04 00 01 00 04  ................
01370b20: 00 01 00 08 00 15 00 0c 00 13 00 20 00 01 00 1a  ........... ....
01370b30: 00 01 00 08 00 15 00 04 00 08 00 12 00 01 00 0a  ................
01370b40: 00 01 00 04 00 01 00 10 00 0f 00 04 00 01 00 10  ................
01370b50: 00 01 00 04 00 08 00 08 00 15 00 04 00 01 00 04  ................
01370b60: 00 01 00 04 00 01 00 0e 00 01 00 04 00 01 00 04  ................
01370b70: 00 01 00 04 00 01 00 0e 00 01 00 08 00 15 00 0e  ................
01370b80: 00 01 00 01 01 00 00 00 00 00 00 00 00 00 00 2d  ...............-
01370b90: 00 00 00 2d 00 00 00 00 00 00 00 00 00 00 00 01  ...-............
01370ba0: 01 00 00 00 00 00 05 07 00 00 00 41 00 4e 00 4f  ...........A.N.O
```

Normal text file   length : 99634176   lines : 1310977   Ln : 1274016   Col : 35   Sel : 0 | 0   UNIX   UTF-8   IN

Figure 1: Hexadecimal representation of the binary data in .evtx event logs.

Figure 1 depicts the binary sequences that form an individual event record. As previously stated, the sequence 0x2a 0x2a indicates the beginning of an event record. For the purpose of extracting information, the Binary XML section can be skipped, as the Substitution Array holds the relevant data to analyze cybersecurity.

Although Shuster mentions that the sequence 0x14 0x00 0x00 0x00 is the sole indicator of the start of the Substitution Array, examination of several event records proves that the sequence 0x12 0x00 0x00 0x00 can also indicate the start of a Substitution Array, as demonstrated by Figure 1. In the cases where 0x14 specifies the start of the Substitution Array, the event ID is located at the 83rd and 84th byte after the header.

For 0x12 Substitution Arrays, the event ID comprises the 75th and 76th byte after the header, 0x10 0x12 in Figure 1. To convert this sequence to an identifiable decimal event ID, reverse the order of the bytes (because Windows stores data in a Little-Endian format) and perform a hexadecimal to decimal conversion. 0x10 0x12 therefore converts to the event ID 4624.

Event ID	Security Alert Message
4608	Windows is starting up
4624	An account was successfully logged on
4625	An account failed to log on
4649	A replay attack was detected
4688	A new process has been created
4724	An attempt was made to reset an account's password
4732	A member was added to a security-enabled local group
4740	A user account was locked out
4960	A Windows Firewall setting has changed

Table 1: Certain Event IDs correspond to critical security threats/alerts.

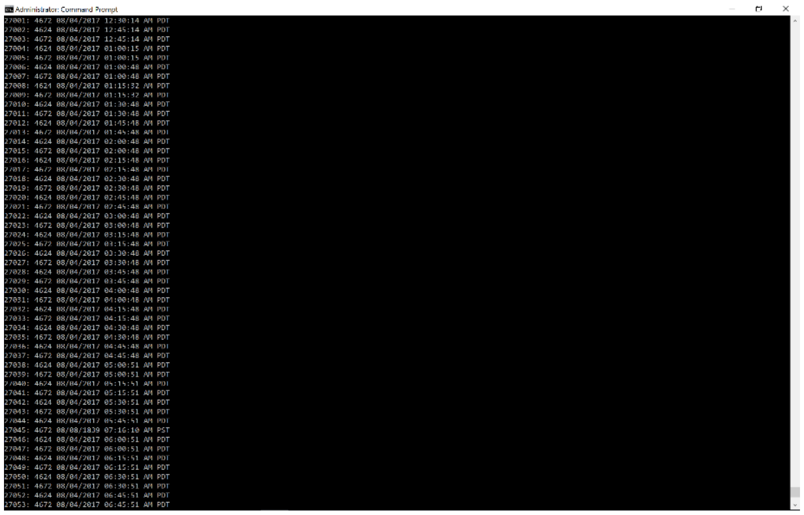
Table 1 maps select event IDs to their corresponding security message/alert. The event record depicted in Figure 1 is therefore an alert that an account successfully logged on to the computer, given by its 4624 event ID. For a full description of event IDs relating to security recorded in Windows Event Logs, please visit https://support.microsoft.com/en-us/help/977519/description-of-security-events-in-windows-7-and-in-windows-server-2008.

Obtaining the time that the security alert was raised is a slightly more challenging task since the 8 bytes that correspond to the Time Created are not always stored at the same location. Additionally, the time is stored as a 64-bit value representing the number of 100-nanosecond intervals since January 1, 1601, requiring a conversion to UNIX time for proper display. The bytes correlating to the time that the event record portrayed by Figure 1 was created are 0x48 0x01 0x58 0x5c 0x33 0x0d 0xd3 0x01, which represent 08/04/2017 08:07:16 AM PDT, meaning that a user logged on to the computer from which this event log was obtained at 8:07 AM on August 4, 2017.

By acquiring the event ID and the time the security alert was raised, the Navy can begin to pinpoint any potential cybersecurity threats harboring within Naval computers and servers. The Navy can monitor event logs for any dangerous events, such as those with event IDs of 4649 (replay attack detected) or 4724 (password reset attempt) etc.

Distributed Computing

The Navy desires a cross platform software tool that can analyze terabytes of event log data to help pinpoint potential cybersecurity threats to military machines. Following the decryption of the event log binary file format, a preliminary event log parser in Java was developed. This program, although capable of accurately extracting the security event data, processed the event logs sequentially and was therefore inefficient. Because the Navy possesses at least 1 gigabyte of event log data, a more efficient algorithm or programming design was needed.



```
27001: 4672 08/04/2017 12:30:14 AM PDT
27002: 4624 08/04/2017 12:45:14 AM PDT
27003: 4672 08/04/2017 12:45:14 AM PDT
27004: 4624 08/04/2017 01:00:15 AM PDT
27005: 4672 08/04/2017 01:00:15 AM PDT
27006: 4624 08/04/2017 01:00:48 AM PDT
27007: 4672 08/04/2017 01:00:48 AM PDT
27008: 4624 08/04/2017 01:15:32 AM PDT
27009: 4672 08/04/2017 01:15:32 AM PDT
27010: 4624 08/04/2017 01:30:48 AM PDT
27011: 4672 08/04/2017 01:30:48 AM PDT
27012: 4624 08/04/2017 01:45:48 AM PDT
27013: 4672 08/04/2017 01:45:48 AM PDT
27014: 4624 08/04/2017 02:00:48 AM PDT
27015: 4672 08/04/2017 02:00:48 AM PDT
27016: 4624 08/04/2017 02:15:48 AM PDT
27017: 4672 08/04/2017 02:15:48 AM PDT
27018: 4624 08/04/2017 02:30:48 AM PDT
27019: 4672 08/04/2017 02:30:48 AM PDT
27020: 4624 08/04/2017 02:45:48 AM PDT
27021: 4672 08/04/2017 02:45:48 AM PDT
27022: 4624 08/04/2017 03:00:48 AM PDT
27023: 4672 08/04/2017 03:00:48 AM PDT
27024: 4624 08/04/2017 03:15:48 AM PDT
27025: 4672 08/04/2017 03:15:48 AM PDT
27026: 4624 08/04/2017 03:30:48 AM PDT
27027: 4672 08/04/2017 03:30:48 AM PDT
27028: 4624 08/04/2017 03:45:48 AM PDT
27029: 4672 08/04/2017 03:45:48 AM PDT
27030: 4624 08/04/2017 04:00:48 AM PDT
27031: 4672 08/04/2017 04:00:48 AM PDT
27032: 4624 08/04/2017 04:15:48 AM PDT
27033: 4672 08/04/2017 04:15:48 AM PDT
27034: 4624 08/04/2017 04:30:48 AM PDT
27035: 4672 08/04/2017 04:30:48 AM PDT
27036: 4624 08/04/2017 04:45:48 AM PDT
27037: 4672 08/04/2017 04:45:48 AM PDT
27038: 4624 08/04/2017 05:00:51 AM PDT
27039: 4672 08/04/2017 05:00:51 AM PDT
27040: 4624 08/04/2017 05:15:51 AM PDT
27041: 4672 08/04/2017 05:15:51 AM PDT
27042: 4624 08/04/2017 05:30:51 AM PDT
27043: 4672 08/04/2017 05:30:51 AM PDT
27044: 4624 08/04/2017 05:45:51 AM PDT
27045: 4672 08/08/1839 07:16:10 AM PST
27046: 4624 08/04/2017 06:00:51 AM PDT
27047: 4672 08/04/2017 06:00:51 AM PDT
27048: 4624 08/04/2017 06:15:51 AM PDT
27049: 4672 08/04/2017 06:15:51 AM PDT
27050: 4624 08/04/2017 06:30:51 AM PDT
27051: 4672 08/04/2017 06:30:51 AM PDT
27052: 4624 08/04/2017 06:45:51 AM PDT
27053: 4672 08/04/2017 06:45:51 AM PDT
```

Figure 2: Output of the sequential EVTX parser

Figure 2 depicts the output of a sequential Java program that parsed a 20 MB security event log with over 30,000 events. Although this method satisfies the objective of extracting data from event logs, a more efficient solution can be implemented by leveraging distributed computing.

Hadoop is an open-source programming framework that supports distributed computing. The Hadoop Distributed File System (HDFS) enables computers in a cluster to interact with one another and share the load of storing and processing files.

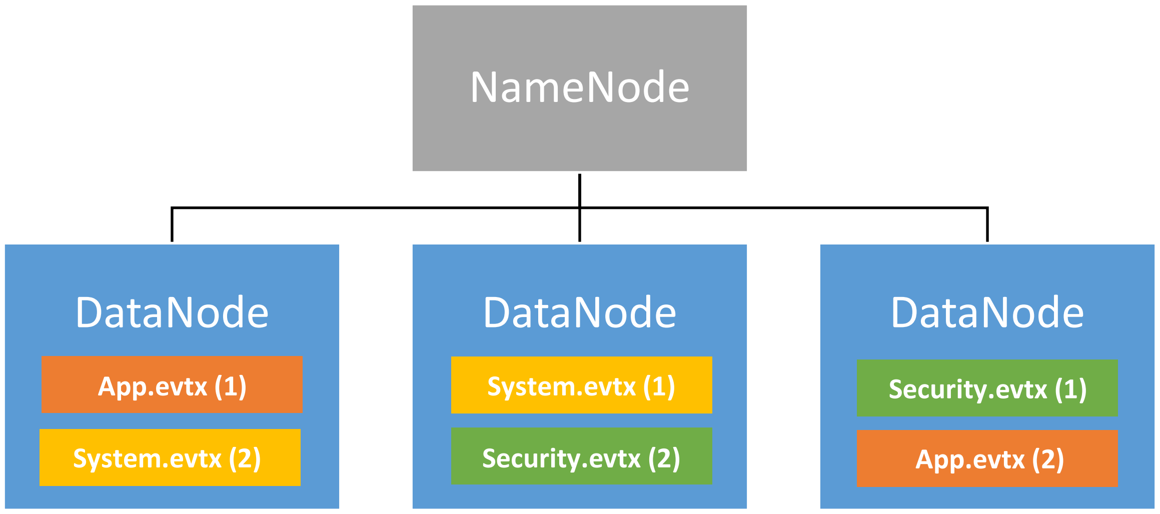


Figure 3: Graphical representation of the Hadoop Distributed File System (HDFS)

Figure 3 portrays the software architecture of HDFS. HDFS consists of a talker or communicator server (NameNode) and several storage servers (DataNodes). The NameNode communicates with all the

DataNodes, and directs the organization and replication of data within HDFS. When a file is uploaded to HDFS it immediately becomes immutable, that is in cannot be further edited. Since HDFS is equipped to handle Big Data, it typically divides large files into 64 MB chunks. Each chunk is replicated and distributed among the DataNodes, as directed by the NameNode. Figure 3 depicts 3 different event log files (orange, yellow, green) of size 20 MB. Since each file is less than 64 MB, it is not divided into blocks, but is rather directly replicated once, with each copy stored on stored on different servers. This organization offers two distinct advantages. (1) If a server crashes or loses memory, a copy of the file is always saved since every file is replicated multiple times and is stored on separate nodes. (2) This design enables distributed computing, as multiple nodes can process the same file at the same time. Instead of a single computer parsing an event log file, Hadoop enables several commodity servers to engage in high performance computing by parallel processing a single file.

HDFS is compatible with MapReduce, a programming paradigm developed by Google designed to manage Big Data. MapReduce efficiently operates over a distributed file system by utilizing multiple nodes to compute, operating on the simple principle of divide and conquer. As its name suggests, there are 2 primary phases in a MapReduce algorithm: (1) the Map phase, (2) the Reduce phase. In the Map Phase, the algorithm generates key-value pairs from the input. In the Reduce phase, the key-value pairs are aggregated into an output. The Map phase typically does all the preprocessing of the data, including filtering and sorting while the Reduce phase performs all the counting or summation operations.

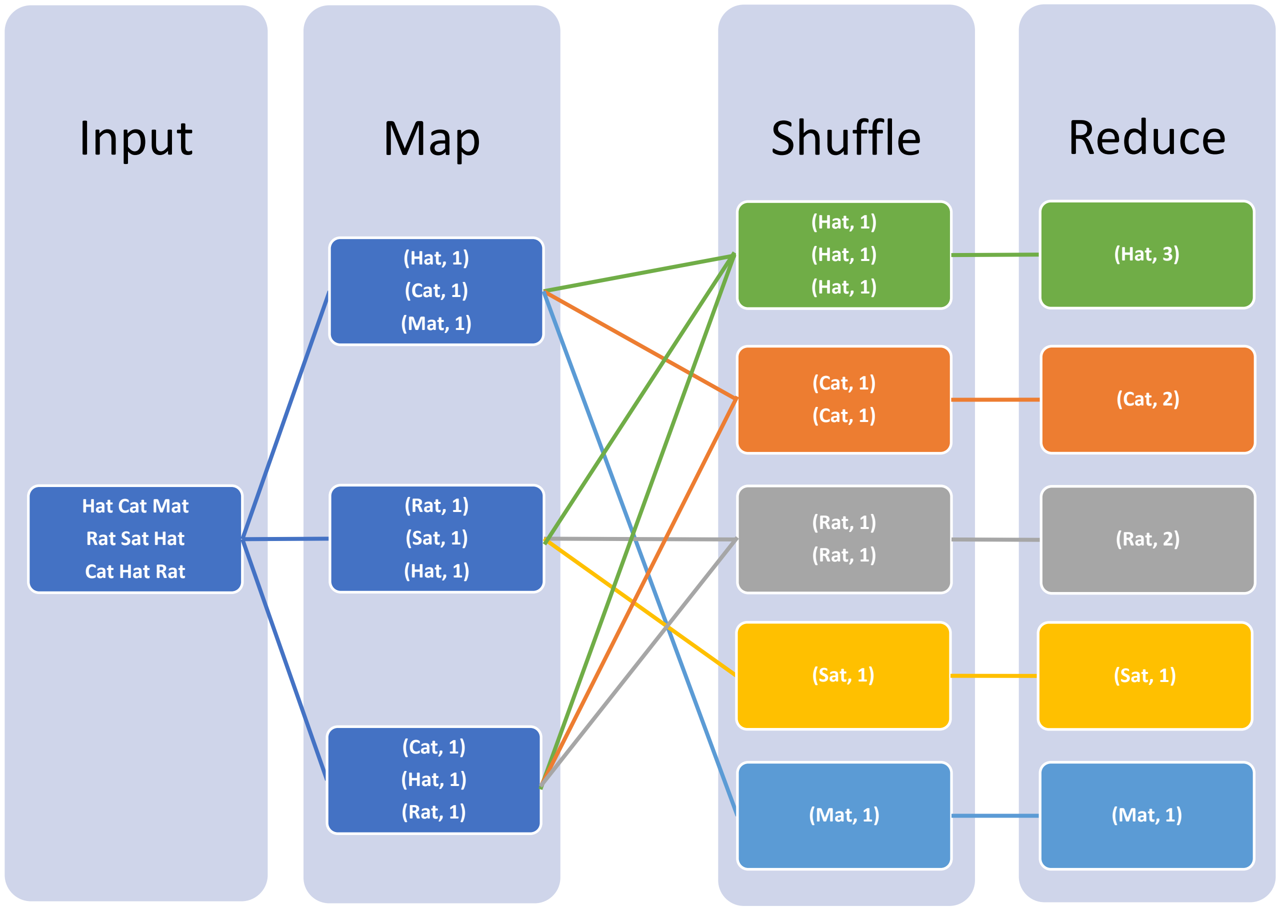
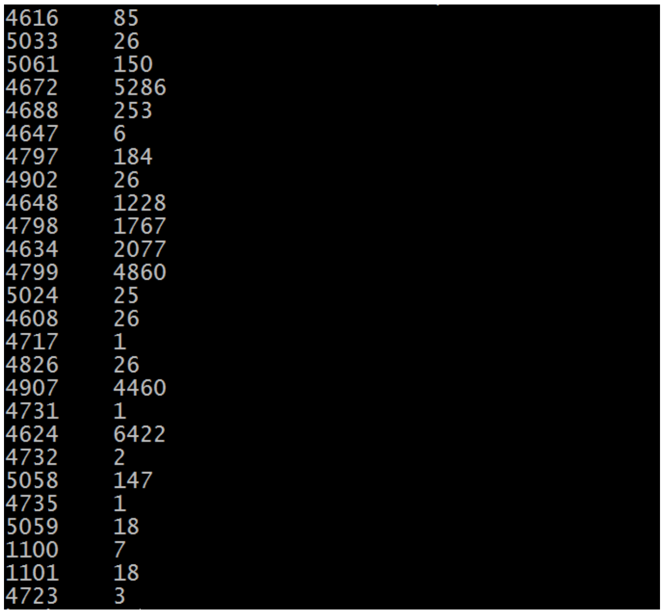


Figure 4: Graphical representation of a MapReduce algorithm

Figure 4 illustrates a MapReduce algorithm performing a wordcount job. It reads in the input text file and breaks it apart into key-value pairs, where each key is a word in the input file with a value of 1. Subsequently, the key value pairs are assigned to different computer nodes by their key in the shuffle phase. Finally, the algorithm aggregates the key-value pairs that have the same key into one pair, summing up the values to get a wordcount for each key.

The same principles of HDFS and MapReduce can be leveraged to parallel process event logs. Each event log is uploaded to HDFS, where the event logs are divided into separate servers and replicated. Then a MapReduce job is run on the event logs files. In the Map phase, the servers parse the event logs as aforementioned and generate key-value pairs of every Event ID and its corresponding Time. The Reducer aggregates the pairs by matching Event IDs and discards the Times and replacing the value with a wordcount of the Event IDs.



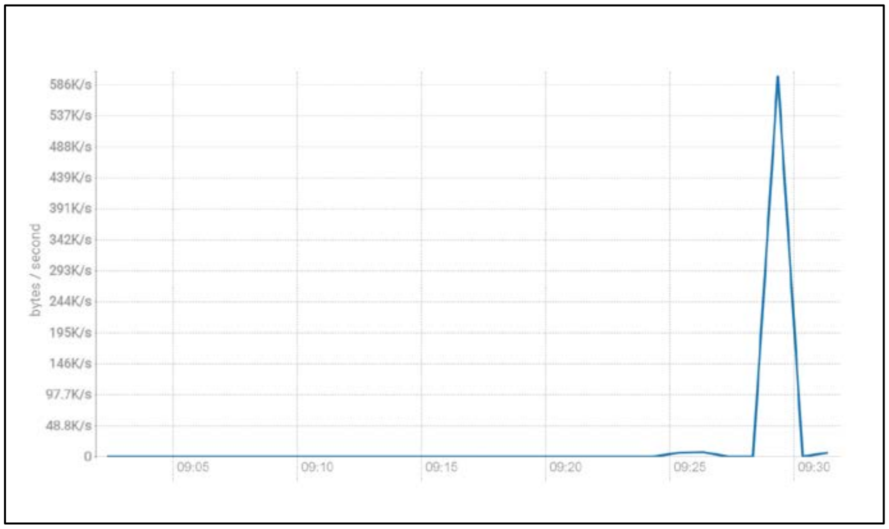
```
4616    85
5033    26
5061    150
4672    5286
4688    253
4647    6
4797    184
4902    26
4648    1228
4798    1767
4634    2077
4799    4860
5024    25
4608    26
4717    1
4826    26
4907    4460
4731    1
4624    6422
4732    2
5058    147
4735    1
5059    18
1100    7
1101    18
4723    3
```

Figure 5: MapReduce output: Event IDs and their corresponding frequency.

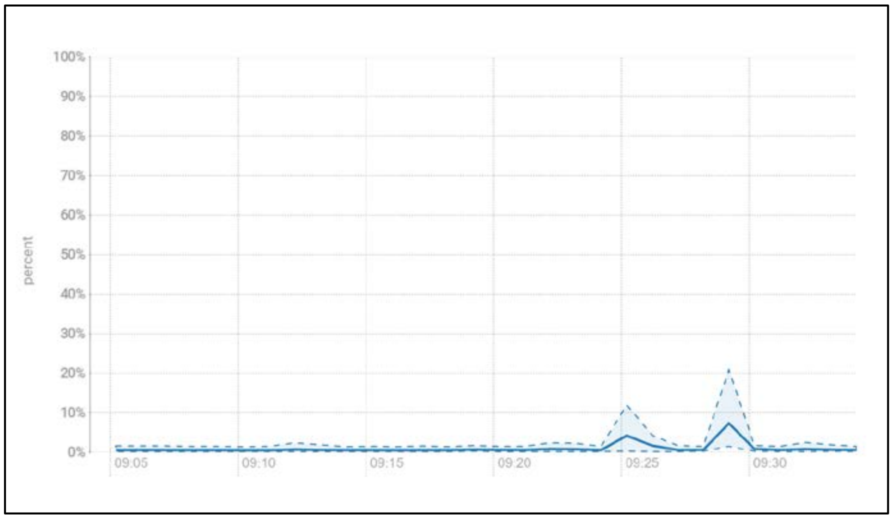
The advantage of leveraging Hadoop MapReduce to process event log data is twofold: (1) HDFS distributes the event logs to different servers, enabling parallel parsing of the binary data. (2) MapReduce can efficiently perform a wordcount operation on the Event IDs, providing insight into the frequency of certain events. As shown in Figure 5, the event ID 4624 has a corresponding value of 6422, signifying that the computer from which the event log was obtained was logged into 6422 times (reference Table 1). Similarly, 4688 has a value of 253, meaning that 253 programs or processes have been executed on this computer.

Conclusion

By utilizing binary forensics and distributed computing, a software application was created that decoded the complex binary structure of windows event logs and extracts relevant event information corresponding to critical security messages and alerts. Its output of Event IDs and corresponding number of incidents gives the Navy insight into the frequency of benign and malignant events on military computers. Since the software is compatible with .evtx files, it can process event logs generated by computers operating on Windows 7 or later, assisting in the upgrade of Naval computers to the latest operating systems. It will be made open-source, to help other individuals and corporations efficiently monitor their computers for cybersecurity threats.



Graph 1: Rate of bytes processed of two MapReduce EVTX parse jobs.



Graph 2: Cluster CPU usage of two MapReduce EVTX parse jobs.

As affirmed by Graph 1 and Graph 2, employing distributed computing and MapReduce has resulted in software that is scalable, able to process gigabytes to terabytes of event log data. A roughly 6000% increase in rate of bytes processed corresponded to an only 3.12% increase in cluster CPU usage, exemplifying the scalability of this software. This software will help pinpoint potential cybersecurity threats on military machines and safeguard against future attacks to the Naval network. Future research and software development can be conducted to extract more information from the event logs such as the application-specific event message and to further optimize the algorithm.

Acknowledgements

The authors would like to thank the ONR (Office of Naval Research) summer internship program at the Naval Postgraduate School, which enabled them to collaborate on this effort. The authors would also like to thank Captain Benjamin Brida for his insight into the Hadoop framework and the Java codebase/libraries for processing binary data, and Jim Zhou for setting up the Cloudera Hadoop cluster (on Redhat Linux) running on a Dell 1U rack mounted servers connected via an infiniband switch.

References

Apache Software Foundation. “MapReduce Tutorial.” *Apache Hadoop 2.9.0 – MapReduce Tutorial*, Apache, <https://hadoop.apache.org/docs/current/hadoop-mapreduce-client/hadoop-mapreduce-client-core/MapReduceTutorial.html>.

Hortonworks Inc. “Apache Hadoop HDFS.” *Apache Hadoop HDFS*, <https://hortonworks.com/apache/hdfs/>.

Hortonworks. “MapReduce.” *Apache Hadoop MapReduce*, <https://hortonworks.com/apache/mapreduce/>.

InfoQ. “Understanding HDFS Using Legos.” *YouTube*, YouTube, 18 Feb. 2015, <www.youtube.com/watch?v=4Gfl0WuONMY>.

Microsoft Corporation. “Description of Security Events in Windows 7 and in Windows Server 2008 R2.” *Support.microsoft.com*, 16 Feb. 2011, <support.microsoft.com/en-us/help/977519/description-of-security-events-in-windows-7-and-in-windows-server-2008>.

Schuster, Andreas. “Introducing the Microsoft Vista Event Log File Format.” *Digital Investigation*, vol. 4, 2007, pp. 65–72., doi:10.1016/j.diin.2007.06.015.