AI-Powered Ransomware Detection Framework

Subash Poudyal, Dipankar Dasgupta
Department of Computer Science
The University of Memphis
Memphis, TN, USA
{spoudyal, ddasgupt}@memphis.edu

***Abstract*—Ransomware attacks are taking advantage of the ongoing pandemics and attacking the vulnerable systems in business, health sector, education, insurance, bank, and government sectors. Various approaches have been proposed to combat ransomware, but the dynamic nature of malware writers often bypasses the security checkpoints. There are commercial tools available in the market for ransomware analysis and detection, but their performance is questionable. This paper aims at proposing an AI-based ransomware detection framework and designing a detection tool (AIRaD) using a combination of both static and dynamic malware analysis techniques. Dynamic binary instrumentation is done using PIN tool, function call trace is analyzed leveraging Cuckoo sandbox and Ghidra. Features extracted at DLL, function call, and assembly level are processed with NLP, association rule mining techniques and fed to different machine learning classifiers. Support vector machine and Adaboost with J48 algorithms achieved the highest accuracy of 99.54% with 0.005 false-positive rates for a multi-level combined term frequency approach.**

***Index Terms*—Ransomware detection, Reverse Engineering, Artificial Intelligence, Dynamic Binary Instrumentation, AI Tool, NLP, FP-Growth**

I. Introduction

Ransomware-as-a-service become a trend due to economic benefits and anonymity. Bad actors working under the dark web has been developing sophisticated ransomware attack techniques by adopting social engineering techniques and exploiting vulnerabilities. The social engineering attack often comes to exploit user's trust and the phishing email, lottery campaigns are more common to make the people fall into the trap of adversaries. According to the survey done by Sophos [16], 45% of ransomware attacks are via malicious links or attachments in emails. 21% of the attacks are from a remote attack on the server and the remaining through misconfigured systems, USB devices, and so on.

The phishing attacks come in the form of COVID-19 themed lures and exploit people's concerns over the pandemic and safety of their family members. Attackers try to increase the anxiety level of internet users. Some of the lures include: information about vaccines, masks, and hand sanitizer; insurance plans to cover COVID-19 illness, government assistance forms for economic relief, and critical updates to consumer applications [6]. Sometimes the victim becomes prey to double extortion. The user files are encrypted for which they are asked to pay the ransom. But these encrypted data may also be held with third parties which often threaten to release those hostage data if the ransom is not paid in time. On May 2020, Maze ransomware attacked Conduent, an IT services enterprise [7]. The Maze ransomware is the latest trend in the ransomware attack which steals data before encryption and threatens to release in the wild if no ransom is paid. The attackers use various exploit kits, some of them are Fallout EK and Spelevo EK that use flash player vulnerabilities [9] Besides, they also use brute force attack on remote desktop protocol. In June 2020, the same Maze ransomware breached systems of the U.S. subsidiary of ST Engineering Aerospace [7]. This ransomware first appeared in November 2019 and uses notoriously smart ways to avoid reverse engineering analysis by the defenders. On July 23rd, 2020, Garmin, a multi-national technology company suffered from WastedLocker ransomware attack disrupting various customer services, apps, and possible data breach [4]. The customers were unable to login, record, or analyze their health and fitness data. WastedLocker first appeared in the wild in April and often leverage the payload through SocGholish and Cobalt Strike tools [19]. The user files are encrypted via AES symmetric keys which are then encrypted using an RSA-4096 public key.

The first ransomware, AIDS Disk, was first spotted in 1989. The initial version of ransomware used symmetric encryption. Archievus ransomware seen in 2006 was the first to use asymmetric encryption. The most common ransomware families are determined based on different reports from security and cyber defense companies which include Kaspersky, SonicWall, WeLiveSecurity, and others [1, 2, 13, 14]. Locky, TeslaCrypt, Cerber, GandCrab, Locker, WannaCryptor, TorrentLocker, Locker, WannaCry, Stop, CryptoJoker, Dharma, and CrypoWall are the most common ones among the others. From 2006 to 2015 various version of both symmetric and asymmetric ransomware appeared. However, table I shows the timeline of ransomware for the last 5 years only.

In this paper, we leveraged the state-of-the-art hybrid malware analysis techniques to perform multi-level analysis at DLL, function call and assembly level. Various AI techniques were used to study the relation at multi-level, train, test and validate our learning model. We propose a AI powered ransomware detection framework and also design a detection tool referred to as AIRaD.

The remainder of this paper is organized as follows. Section II presents the related work. In section III we discuss the proposed AI-powered ransomware detection framework. Section IV discusses about the experiment. Section V deals

978-1-7281-2547-3/20/$31.00 ©2020 IEEE

TABLE I: Timeline of ransomware families [15, 12, 17]

2016	Ransom32, 7ev3n, CryptXXX.
2017	Spora, DynA-Crypt, Samas, Wannacry, NotPetya, Locky, Scarab.
2018	Ransomcloud, GrandCrab, SamSam, KillDisk, NewMada.
2019	LockerGoga, vxCrypter, eCh0raix, GermanWiper, MegaCortex, REvil, GandCrab.
2020	WannaCryptor, Cerber, Crysis, Sodinokibi, Stop, Phobos, Dharma, WastedLocker....

with association rule mining. Section VI is about ransomware specific behavior chains. Section VII describes the ransomware analysis tool. The paper ends with a conclusion in section VIII.

II. Related work

Researchers are continuously working to analyze and detect ransomware. Some of the notable works are discussed here. Takeuchi et al. [30] have proposed ransomware detection using API call sequence and support vector machines as classifiers with 97.48% accuracy. Hampton et al. [25] studied the behavior of ransomware in the Window system and identified API calls specific to it. The frequency of API usage among ransomware and normal binaries are useful for identifying ransomware without comparing the code signature. They claim that their approach can allow a better understanding of the ransomware strain in terms of API calls. Bae et al. [22] have used the Intel PIN tool to generate windows API call sequences and used the N-gram approach and TF-IDF. They reported accuracies for ransomware, malware, and benign application using different machine learning classifiers. Continella et al. [24] proposed ShieldFS which analyzes low level I/O file system requests. If a write operation is suspicious it reverts the current process file operations. PayBreak [27] tried to recover the data corrupted by ransomware by extracting the encryption key. This approach only works for symmetric encryption or hybrid encryption. Here, no accuracy evaluation is done. Moreover, they have not explained how they would handle the false positives as encryption behavior can be found in normal user profiles too. Kharaz et al. [26] proposed a ransomware detection framework called UNVEIL which tracks encryption behavior of ransomware. They tracked IO operations calculating entropy scores. The authors reported the detection of 13,000 malware samples across different malware families. However, they did not verify the accuracy of their system. Canzanese et al. [23] have analyzed system call traces utilizing the N-gram language model and TF-IDF to detect malicious processes. They have claimed that their proposed system would alarm the user if some unintended behaviors are observed which includes activities like host modifications. However, they have not explained how they will deal with a false alarm. Moreover, their detection approach is not resilient to the obfuscated behavior of the malware. Trung et al. [31] have used methods such as N-gram, doc2vec, TF-IDF to convert the API(Application programming interface) sequences to numeric vectors. These vectors are supplied to the machine learning classifiers. Andronio et al. [21] proposed the HELDROID framework for detecting ransomware. The framework works by searching for the requisite ransomware elements within mobile applications. It can detect whether an app is trying to encrypt or lock the device without permission from the user. The results of testing HELDROID on APKs comprising ransomware, goodware, scareware, and malware showed almost zero false positives.

Unlike other's work, our approach leverages intelligent ransomware analysis components with multi-level deep inspection to uniquely identify ransomware among others.

III. AI-powered ransomware detection framework

The overall back end architecture for AI-powered ransomware detection framework is shown in Figure 1. A brief discussion of each phase follows below.

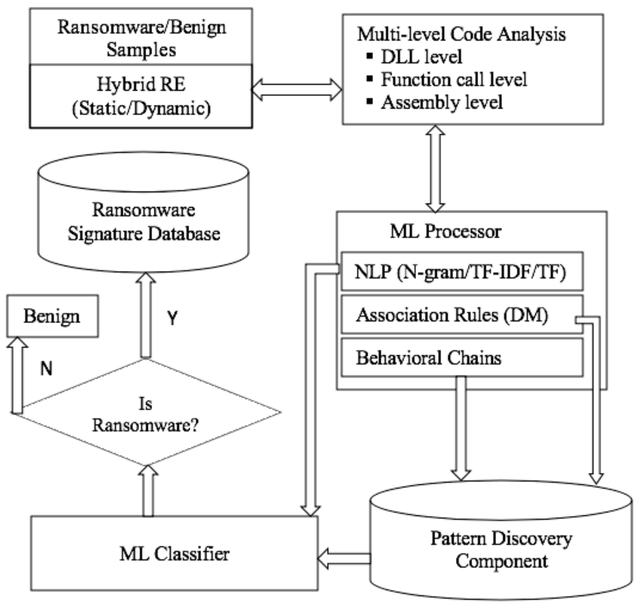


Fig. 1: AI-powered ransomware detection framework. Some terms used are NLP: Natural Language Processing DM: Data Mining RE: Reverse Engineering.

A. *Hybrid Reverse Engineering*

First, the ransomware and benign samples will be reverse engineered using a hybrid approach. Hybrid reverse engineering involves both static and dynamic analysis of ransomware and benign binaries. This static and dynamic analysis comes with the pros and cons. Static analysis is observing and extracting some features of a binary placed or stored in a hard drive. In contrast, dynamic analysis involves extracting behavior and features by running the binary in the memory. Static analysis is necessary to capture the initial properties of the binary. Further analysis is done using Object dump, a Linux based tool and PE parser [10], an open-source tool which helps to reveal properties like import and export of functions by the binaries. It is assumed to achieve high code coverage than via dynamic analysis because some missing parameters misguide the dynamic analysis. Static analysis generally tracks the code from start to end though the dynamic behavior is not captured. Dynamic analysis is done using a virtualized environment such as a Cuckoo sandbox [3] and the dynamic binary instrumentation tool, PIN [11]. The modern version of ransomware families is often difficult to analyze due to their anti-analysis techniques. Thus, we see the significance of hybrid analysis and so is our approach. This phase also includes the initial pre-processing of the received raw outputs obtained via the adopted reverse engineering approaches.

1) Dynamic Binary Instrumentation: Instrumentation techniques have been used by programmers to diagnose program crashes, analyze errors and to write trace information. Instrumentation comes in two types: code instrumentation and binary instrumentation. Since the malware code is not available, we consider only binary instrumentation. Binary instrumentation is often referred to as Dynamic binary instrumentation or DBI in short as the instrumentation is done using dynamic analysis of program traces. In our approach, we use the PIN tool [11] for DBI. PIN makes tracking of every instruction executed possible by taking complete control over the run-time execution of the binary.

2) Cuckoo sandbox: Cuckoo sandbox is an advanced open source automated malware analysis tool which allows execution and analysis of malware samples in a safe environment [3]. It makes use of the virtualization technique to run malware samples. We choose Cuckoo sandbox among the others because of its modular design supporting multiple environments and allows flexibility in result analysis. If highly sophisticated ransomware does not run in a virtualized environment then its function call trace is studied via reverse engineering provided by NSA's Ghidra tool [5].

B. Multi-level Code Analysis

Multi-level code analysis is a unique approach since the code is inspected at three levels i.e DLL, function call, and assembly level using a hybrid reverse engineering approach. We extract the binary behavior and properties specific to these three levels.

1) DLL Level: DLLs are dynamic link libraries which are subroutines to perform actions such as file system manipulation, navigation, process creation, communication, and so on. They are loaded into the memory whenever required and freed from memory whenever not needed making our system lightweight. Thus, it makes effective use of available memory and resources by dynamic linking capability. DLLs have functions that they export and make it available to other programs. During a program run, all necessary DLLs are loaded into the memory, but the referenced function call is accessed only when needed by locating the memory address where the function code resides. There are certain DLLs that are called more often because the function calls implemented by them are significant to carry out actions as per malware behavior.

2) Function call Level: A function call is a piece of code that has lines of instructions that make an impact to the system or user. These are essential code blocks that carry out various functionalities and have less overlapping than DLL and assembly level analysis. Analysis at this level helps to identify function calls that are more unique to malware's behavior. Categorization of functions based on functionality such as file operations, system information gathering, file enumeration, encryption key generation, and encryption, etc. are some of the key behaviors specific to ransomware that we analyze at this level.

3) Assembly Level: Assembly instruction is a low-level machine instruction, which is also called machine code. It can be directly executed by a computer's central processing unit (CPU). Each assembly instruction causes a CPU to perform a specific task, like *add, subtract, jump, xor*, and so on. Each function call or system call is implemented via assembly instructions. Assembly instructions are also analyzed based on categorized groupings. Some of the categories are Data transfer, Logical, Control transfer, Flag control, and so on.

C. ML Processor

Machine Learning (ML) Processor consists of various machine learning components that assist the multi-level code analysis.

Natural Language Processing (NLP) language models have proved useful in recommendation systems, text classification, speech recognition, and so on. Through NLP component various popular NLP techniques such as N-gram, Term Frequency-Inverse Document Frequency (TF-IDF) and Term Frequency (TF) are leveraged to generate feature database which later on is fed to the ML classifier.

Association rule is a rule-based data mining (DM) approach to discover notable relations and patterns among variables in a given data. FP-growth algorithm is preferred as it is more efficient than others including Apriori. Apriori takes more execution time for repeated scanning to mine frequent items but FP-growth scans the database only twice for constructing frequent pattern tree. Through Association rules component FP-growth algorithm is leveraged to discover notable relations and patterns at multi-level.

Through the Behavior chain component, ransomware specific chains are discovered showing the relation at three levels.

D. Pattern Discovery Component

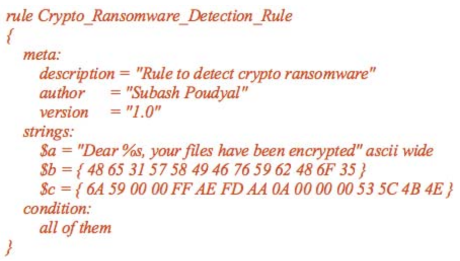
Both Association rules and Behavior chain component contribute to pattern discovery. The pattern database consists of all the discovered patterns which can be either association rule chains or behavioral chains. This database is considered as a feature database for the ML classifier.

E. ML Classifier

ML classifier component leverages the various supervised machine learning classifiers to train, test, and validate the model and give a decision whether a binary in consideration is a ransomware or a benign application. We use Logistic Regression (LR), Support vector machines (SVM), Random Forest (RF), J48, and Adaboost with RF and J48.

F. Ransomware Signature Database

If the binary is ransomware, then its signature will be stored in the malware signature database and binary is deleted. This decision will be based on threshold accuracy of 95%. If the binary is not ransomware then it is labelled as benign. The unique N-gram sequences with N-gram TF-IDF probability scores 1 is more certain to be seen. Similarly, association rules pattern with score value 1 are more certain to be seen. So, is the case with custom behavioral chains. These patterns' corresponding opcode locations at the PE file help to set up conditions for the Yara rules [20]. Figure 2 shows a sample Yara rule. This rule says that if all the string's pattern specified in variables a, b, and c are observed in a binary then it is a probable crypto ransomware.



```
rule Crypto_Ransomware_Detection_Rule
{
   meta:
      description = "Rule to detect crypto ransomware"
      author      = "Subash Poudyal"
      version     = "1.0"
   strings:
      $a = "Dear %s, your files have been encrypted" ascii wide
      $b = { 48 65 31 57 58 49 46 76 59 62 48 6F 35 }
      $c = { 6A 59 00 00 FF AE FD AA 0A 00 00 00 53 5C 4B 4E }
   condition:
      all of them
}
```

Fig. 2: Sample Yara rule to detect crypto ransomware

IV. Experiments

In this section, we discuss about data collection, evaluation measures and results.

A. Dataset

The dataset consisted of the binaries in a portable executable (PE) file format. Most of the malware attacks occur by leveraging the PE file format. The bad actors often target the windows operating system (OS) which uses PE file format, and this OS is widely used all over the world. As per security report from Fireeye [8] 70% of the malware attacks are launched via executable which has PE file format. 550 samples of ransomware were collected from Virus Total [18] and 540 normal samples from the Windows 10 OS and open-source software.

B. Experimental Protocols and Evaluation Measures

We experimented with coding in python, bash script, and the system with configuration Intel(R) Core(TM) i7-5500U CPU @ 2.40 GHz 2.39 GHz, 8.00 GB RAM, and 1 TB disk space. For running malware samples 6 virtual environments was set up with five i7 processor machines, one with 32 GB RAM and four with 8 GB RAM.

For evaluation of the experiments performed we used widely used performance metrics of True Positive Rate (TPR), False Positive Rate (FPR), Precision, Recall, F-measure, and accuracy. For association rule mining, minimum support threshold is set to 2 and confidence threshold to 0.8, but only association rules with a score of 1 are shown in section V.

C. Experimental Results

The experiments done with combined multi-level features with term frequency show promising accuracy with low false-positive rates as seen in Table II. SVM and Adaboost with J48 reported the highest accuracy of 99.54% and lowest false positive rate of 0.005. J48 achieved the second highest accuracy of 99.26% and a false positive rate of 0.007. The improved accuracy seen here than our previous approaches [28, 29] is due to the hybrid reverse engineering techniques used which builds a unique feature set.

V. Association Rule Mining

Association rules show the probability of relationship between items. It is generally a rule-based data mining approach that helps to discover correlations among data items. In our approach, we have used the FP-Growth algorithm, which is an array-based, uses depth-first search, and requires only two database scans, making it more efficient and scalable. A list of DLLs, function calls, and corresponding assembly instructions obtained from the advance reverse engineering process is provided as an input to the FP-Growth algorithm. The generated FP-Growth association rules are used to create ransomware detection signatures.

A. Association rules at DLL level

Table III shows a portion of the association rules at the DLL level. These DLLs are obtained from the import table of ransomware portable executable files. The first-row rule shows the use of ADVAPI32, OLE32, SHELL32, and WS2_32 DLL implies KERNEL32 DLL. This rule contains DLLs associated with ransomware specific behavior including encryption as described in section VI. The eighth rule shows the use of ADVAPI32, KERNEL32, OLE32, SHLWAPI, WININET, WS2_32 implies SHELL32. This rule contains DLLs specific to ransomware behavior including deletion of shadow copy.

B. Association rules at Function call level

Table IV shows a portion of the association rules at the function call level. These function calls are obtained via both static and dynamic analysis of ransomware executables. The first row rule shows the use of GetCurrentThreadId, GetTickCount, RtlUnwind, WideCharToMultiByte, lstrlenW implies ExitProcess. This rule contains function calls that are specific to the anti-analysis behavior of ransomware.

TABLE II: Machine learning algorithms' evaluation for multi-level combined approach with term frequencies

Machine learning Classifier	TPR	FPR	Precision	Recall	F-measure	Accuracy (%)
Logistic Regression	0.992	0.008	0.992	0.992	0.992	99.17
Support Vector Machine	0.995	0.005	0.995	0.995	0.995	99.54
Random Forest(RF)	0.990	0.010	0.990	0.990	0.990	98.99
J48	0.993	0.007	0.993	0.993	0.993	99.26
Adaboost with RF	0.987	0.013	0.987	0.987	0.987	98.71
Adaboost with J48	0.995	0.005	0.995	0.995	0.995	99.54

C. Association rules at Assembly level

Table V shows a portion of the association rules at the assembly level. These function calls are obtained via dynamic binary instrumentation. These rules are specific to different function calls defined for various chains as shown in section VI.

The combination of association rules at three levels proves more effective to create unique ransomware detection signatures based on functionality chains discussed in section VI. The experimental results also show promising accuracy with low false-positive rates.

TABLE III: Association rule mining at DLL level with score: 1.0

Association rules at DLL level
[ADVAPI32, OLE32, SHELL32, WS2_32] → [KERNEL32]
[ADVAPI32, OLE32, SHELL32, WS2_32] → [ADVAPI32]
[ADVAPI32, OLE32, SHELL32, WININET] → [KERNEL32]
[ADVAPI32, OLE32, SHELL32, WININET, WS2_32] → [KERNEL32]
[KERNEL32, OLE32, SHELL32, WININET, WS2_32] → [ADVAPI32]
[ADVAPI32, KERNEL32, OLE32, SHELL32, WS2_32] → [WININET]
[ADVAPI32, KERNEL32, OLE32, WININET, WS2_32] → [SHELL32]
[ADVAPI32, KERNEL32, OLE32, SHLWAPI, WININET, WS2_32] → [SHELL32]
[ADVAPI32, KERNEL32, OLE32, SHELL32, SHLWAPI, WININET] → [WS2_32]
[ADVAPI32, KERNEL32, OLE32, SHELL32, WININET, WS2_32] → [SHLWAPI]

TABLE IV: Association rule mining at function call level with score: 1.0

Association rules at function level
[GetCurrentThreadId, GetTickCount, RtlUnwind, WideCharToMultiByte, lstrlenW] → [ExitProcess]
[CloseHandle, GetModuleHandleA, GetTickCount, ReadFile, RtlUnwind, WideCharToMultiByte] → [GetProcAddress]
[CloseHandle, GetCurrentProcessId, GetModuleHandleA, ReadFile, RtlUnwind, WideCharToMultiByte] → [ExitProcess]
[ExitProcess, GetCurrentThreadId, GetTickCount, SetFilePointer, Sleep, WideCharToMultiByte] → [GetModuleHandleA]
[ExitProcess, GetCurrentThreadId, GetModuleHandleA, RtlUnwind, SetFilePointer, WideCharToMultiByte] → [WriteFile] 1.0)
[CloseHandle, ExitProcess, GetCurrentProcessId, GetModuleHandleA, RtlUnwind, WideCharToMultiByte] → [ReadFile]
[GetCurrentProcess, InterlockedIncrement, IsDebuggerPresent, RaiseException, RtlUnwind, SetUnhandledExceptionFilter, Sleep, UnhandledExceptionFilter, VirtualProtect, WideCharToMultiByte] → [HeapCreate]
[EnterCriticalSection, ExitProcess, GetLastError, GetProcAddress, LeaveCriticalSection, WriteFile] → [HeapReAlloc]

VI. Behavioral Chaining

Ransomware specific behavioral chains are basically multi-level chains which are constructed by studying the behavior

TABLE V: Association rule mining at assembly level with score: 1.0

Association rules at Assembly level
[add, and, cmp, data16, jmp, lea, sbb, shl] → [xor]
[add, cmp, data16, jm, lea, sbb, shl, sub] → [xor]
[add, cmp, data16, jmp, or, rcr, sbb] → ror]
[add, cmp, data16, jmp, or, rcr, shl] → sbb]
[add, cmp, data16, jmp, rcr, sbb, shl] → [xor]
[add, cmp, fdivr, jmp, les, or, repz, sbb, sub] → [bound, ror]
[add, cmp, fdivr, jmp, les, or, repz, shl, sub] → [sbb]
[add, cmp, fdivr, jmp, les, repz, sbb, shl, sub] → [xor]
[add, call, ficom, les, lock, or, rcr, sbb, xchg] → [cmp]
[call, cmp, ficom, les, lock, or, rcr, sbb, xchg] → [add]

of different ransomware families. Both static and dynamic analysis of ransomware binaries reveals the different chains which are seen in a wide range of ransomware families. Table VI shows the chains commonly seen in ransomware binaries.

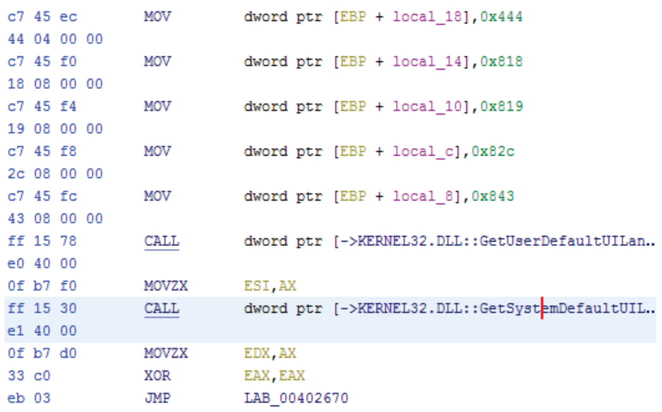
TABLE VI: Chaining ransomware behavior

Chain A	System services with initial setup
Chain B	Module enumeration
Chain C	Anti-analysis
Chain D	Access elevation
Chain E	Snapshot
Chain F	Parameter setup
Chain G	System profiling
Chain H	Encryption setup
Chain I	File encryption
Chain J	Ransom note
Chain K	Network enumeration
Chain L	Deletion
Chain M	Error handling
Chain N	CC communication

Chain A deals wih system services and an initial setup. It uses *GetStartupInfoW* which gets information related to the window station, desktop, and appearance of the main window. *GetStdHandle* gets a handle to the specified IO device. These handles are used by Windows applications to read and write to the console. These are also used by *ReadFile* and *WriteFile* functions. *GetEnvironmentStringsW* makes the environment variables available for the current process running in an infected computer. *FreeEnvironmentStringsW* frees all environment settings. This function is generally used only once. Malware writers do not want to interfere with their work. They may use *SetEnvironmentVariable* to set certain variables to fulfill their malicious behavior. *IsProcessorFeaturePresent* determines whether the specified processor feature is supported by the system in use.

Chain B deals with module enumeration. *GetModuleFileName* loads the malware executable and *GetModuleHandle* gets the handle to the custom malware DLL with obfuscated functions. Obfuscation behavior can be captured at the assembly level.

Chain C deals with the anti-analysis behavior of the ransomware. *GetTickCount* gives the time in milliseconds that have passed since the system was started. *GetSystemInfo* and *GetNativeSystemInfo* use *dwNumberOfProcessors* method to check the number of processors running in a system.



```
c7 45 ec        MOV      dword ptr [EBP + local_18],0x444
44 04 00 00
c7 45 f0        MOV      dword ptr [EBP + local_14],0x818
18 08 00 00
c7 45 f4        MOV      dword ptr [EBP + local_10],0x819
19 08 00 00
c7 45 f8        MOV      dword ptr [EBP + local_c],0x82c
2c 08 00 00
c7 45 fc        MOV      dword ptr [EBP + local_8],0x843
43 08 00 00
ff 15 78        CALL     dword ptr [->KERNEL32.DLL::GetUserDefaultUILan..
e0 40 00
0f b7 f0        MOVZX    ESI,AX
ff 15 30        CALL     dword ptr [->KERNEL32.DLL::GetSystemDefaultUIL..
e1 40 00
0f b7 d0        MOVZX    EDX,AX
33 c0           XOR      EAX,EAX
eb 03           JMP      LAB_00402670
```

Fig. 3: Disassembled code to check default user language

If the system has only one processor then the malware writers label it as an analysis environment and may not execute at all. *GetUserDefaultUILanguage* gets the language identifier for the current user while *GetSystemDefaultUILanguage* gets for the operating system. This function chain is often used to reveal the user language so that the malware writers can decide whether to execute further or not based on the country and spoken language preferences.Figure 3 shows its usage.

Chain D deals with access elevation. *OpenProcessToken* opens the token associated with a given process while *GetTokenInformation* is used to obtain the token id, session id, or security identifier of the process's owner. This obtained token is duplicated and applied to a new thread created in suspended mode using *SetThreadToken*. It also contains function calls to bypass user access control by elevating the privilege to the admin level.

In **Chain E**, *CreateToolhelp32Snapshot* is used to create a snapshot of processes, heaps, threads, and modules. Malware often uses this function as part of the code that iterates through processes or threads. This snapshot function is called during different functionality blocks such as while loading DLLs, loading application processes, loading antivirus processes, and so on. *Process32FirstW* gets information about the first process seen in a system snapshot. This is used to enumerate processes from a previous call to *CreateToolhelp32Snapshot*.

Chain F deals with parameter setup. The command-line string via *GetCommandLineA* serves as one parameter value to be passed to *GetCommandLineW* function which later removes malware itself and deletes shadow copies via the command prompt window.

Chain G is concerned with profiling system identifiers. Some of the often profiled system identifiers are keyboard layout, Windows version used, domain used, CPU identifier, and so on. *RegOpenKeyExW* opens the specified registry key for system profiling. The parameter *lpSubKey* specifies the name of the registry key to be open. The access right for the registry key object is *KEY_EXECUTE* (0x20019), which is equivalent to *KEY_READ*.

Chain H is concerned with encryption setup. The *CryptAcquireContextW* function is used to acquire a handle to a key container implemented by either cryptographic service provider (CSP) or Next-generation CSP. The *szProvider* parameter specifies this information. Example:

szProvider="Microsoft Enhanced Cryptographic Provider v1.0"

The *CryptGenKey* generates a public/private key pair. The handle to the key is returned in parameter *phKey*. It has the *Algid* parameter which specifies the type of encryption algorithm being used. For example, *Algid=0xa400* represents *CALG_RSA_KEYX* as the "RSA public key exchange algorithm". The *CryptExportKey* function exports a cryptographic key pair from a CSP in a secure manner. At the receiver end *CryptImportKey* function should be used to receive the key pair into a recipient's CSP. *CryptDestroyKey* destroys the encryption handle but not the keys. *CryptReleaseContext* releases the handle of a cryptographic service provider and a key container.

Chain I deals with file encryption. At first ransomware iteratively finds next files in a given folder to encrypt using *FindNextFileW* then writes the filename.some_unique_extension as a new filename to the buffer using *wsprintfw*. The local file names are often compared if they are not among these files: autorun.inf, ntuser.dat, iconcache.db, bootsect.bak, boot.ini, ntuser.dat.log, thumbs.db, ransom_note.html, ransom_note.txt so that it won't interfere with the normal functioning of the system and also should not encrypt the ransom message. *lstrcmpiW* is used to make comparisons with the discovered filename with the hardcoded list of filenames. This list or approach slightly differs among various ransomware families. The *CryptAcquireContextW* handle is called to get the *CryptoAPI* function i.e *CryptGenRandom* ready to use. Here, *CryptGenRandom* is used to generate a random key to be used by symmetric encryption algorithms. *CryptEncrypt* does the real encryption of text or strings. *CryptDestroyKey* only destroys the encryption handle but not the keys. *CryptReleaseContext* releases the handle of a cryptographic service provider and a key container.

CreateFileW creates a new file or opens an already existing file to overwrite its content. *ReadFile* reads the just opened file using its handle from the position specified by the file pointer. *WriteFile* writes given data of buffer pointer to the specified file. Finally, *MoveFileW* function moves file to the same or different location but with a different filename extension i.e .some_extension is attached to the current filename. Again, this differs among ransomware families. Some ransomware families overwrite the filename with some random strings being generated using the *CryptGenRandom* function.

Chain J deals with creating a ransom note. *Wsprintfw* function writes some file name ransom_message.txt to buffer then creates a new file of that name and returns the handle using *CreateFileW*. *LstrlenW* gets a length of the text to be written while *WriteFile* is used to write the ransom note to the specified file. Finally, *CloseHandle* closes the file handle given by the *CreateFileW* function.

Chain K has functions associated with network enumeration. The functions *WNetOpenEnumW, WNetEnumResourceW, and WNetCloseEnum* are used in a chain for lateral movement across the network to infect more victim's machines.

Chain L deals with self-delete. *GetModuleFileNameW* gets the malware executable location while the function *wsprintfW* writes the previously obtained command line parameter to buffer. The *ShellExecuteW* function via *lpFile* parameter value as *cmd.exe* executes the given command. The DLL, function call, and assembly used for this chain are shown in Table VII.

TABLE VII: Chain L: Self deletion

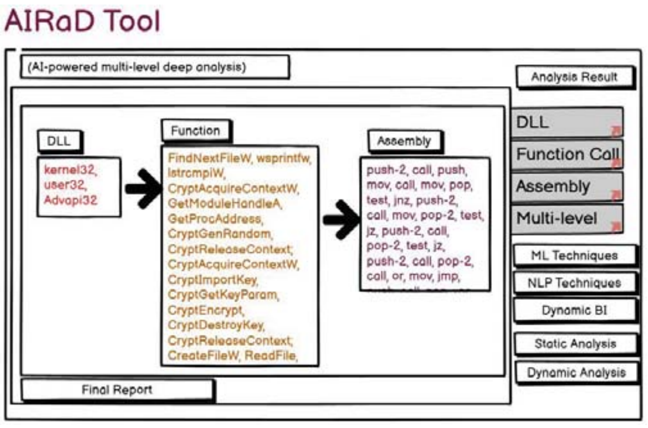
DLL	Function	Assembly
kernel32, user32, shell32	GetModuleFileNameW, wsprintfW, ShellExecuteW	push, mov, push-5, call, push-4, mov, call, mov, test, jz, push-3, call, test, jz, push-3, call, add, push-6, call, push, call, int

Chain M deals with error handling. *GetLastError* gets the last error code for the calling thread of the given process. *GetCurrentThreadId* gets the identifier value for the thread whose error code for execution of a certain function is to be considered. *SetLastError* sets the error code for the calling thread of a given process. For example zero error code means error success and the operation was successfully completed. This sequence comes more often to get the status of functions being executed.

Chain N deals with command and control(CC) server communication. This function chain differs among different ransomware families as some use hard-coded URL, some use domain generation algorithm, and the way to get the victim's IP address also differs. Here, the most seen common sequence is illustrated. *InternetOpenW* function opens the browser application, *InternetConnectW* opens a File Transfer Protocol (FTP) or HTTP session for a given site. Malware may use *ipv4bot.whatismyipaddress.com* to find the victim's IP address Or they could find via command prompt. In the meantime, it connects to CC server via *HttpOpenRequestW* using the handle of *InternetConnectW* function. *HttpAddRequestHeadersW* specifies the CC server. *InternetReadFile* reads the data from a handle opened by the *InternetOpenUrl*, *FtpOpenFile*, or *HttpOpenRequest* function. Finally, *InternetCloseHandle* closes the internet handle.

VII. AIRaD TOOL

Having defined the ransomware detection approach using AI techniques we got motivated to design a ransomware analysis tool referred to as AIRaD (AI powered Ransomware Detection) tool.



kernel32, user32, Advapi32

FindNextFileW, wsprintfw, lstrcmpiW, CryptAcquireContextW, GetModuleHandleA, GetProcAddress, CryptGenRandom, CryptReleaseContext; CryptAcquireContextW, CryptImportKey, CryptGetKeyParam, CryptEncrypt, CryptDestroyKey, CryptReleaseContext; CreateFileW, ReadFile,

push-2, call, push, mov, call, mov, pop, test, jnz, push-2, call, mov, pop-2, test, jz, push-2, call, pop-2, test, jz, push-2, call, pop-2, call, or, mov, jmp,

Fig. 4: Snapshot view of AIRaD tool

This tool is in the process of development and we plan to make it an open-source tool. Figure 4 shows one of the output interfaces of our tool. It shows the multi-level mapping for file encryption activity. The main box shows DLL, function call, and assembly components with an arrow pointing from DLL to the assembly level. This association among these three levels is significant to recognize ransomware specific behavior and create unique signatures. The upper part of rightmost column shows buttons to choose either one level of analysis or multi-level analysis. Choosing DLL option shows all the DLLs specific to the selected sample. Similarly, for function call and assembly. The multi-level option shows DLL, function call and assembly level mappings of various ransomware behavioral chains. The bottom portion of it shows buttons for machine learning techniques, NLP techniques, Dynamic binary instrumentation, and Static and dynamic analysis approaches used in our tool. A user will be able to select which machine learning algorithm he/she wants to evaluate with. Similarly, with NLP techniques and dynamic binary instrumentation. This tool leverages the techniques defined in our proposed framework and can also be considered an explanatory AI tool as it identifies the distinguishing behavioral chains which help to create a unique dataset for machine learning models.

VIII. CONCLUSION

In this paper, we proposed an AI-powered ransomware detection framework and designed a ransomware analysis tool using the techniques of reverse engineering, static and dynamic analysis, and machine learning. Leveraging dynamic binary instrumentation tool PIN, Cuckoo sandbox environment, and Ghidra framework we generated distinguishing feature dataset. Association rule mining and Ghidra's disassembly contributed to the existing analysis by other approaches to build unique behavioral multi-level chains specific to ransomware. Collectively this contributes to creating unique Yara rules which can be used by researchers and the security community. The major contribution of our work can be summarized as follows:

- We designed an automated AI-powered hybrid reverse engineering framework that leverages the multi-level features extracted by static and dynamic analysis techniques including various machine learning methods and achieved high accuracy and low false-positive rate.
- We designed and developed the AIRaD (AI powered Ransomware Detection) tool focusing on simplicity, usability and analysis power.
- The behavioral chain of ransomware is a unique approach that creates distinct ransomware detection signatures. Similarly, association rule mining revealed distinguishing sequences at multi-level.

In the future, we plan to distinguish ransomware family-specific behavior and make the ransomware detection tool fully functional.

References

[1] 2020 q1 threat report welivesecurity. https://www.welivesecurity.com/wp-content/-uploads/2020/04/ESET$_{T}hreat_{R}eport_{Q}12020.pdf. \quad Accessed - onJuly20, 2020.$

[2] 2020 sonicwall cyber threat report. https://www.sonicwall.com/resources/2020-cyber-threat-report-pdf/. Accessed on July 20, 2020.

[3] Cuckoo automated malware analysis. https://cuckoosandbox.org . Accessed on May 22, 2020.

[4] Garmin begins recovery from ransomware attack. https://www.bbc.com/news/technology-53553576 . Accessed on July 28, 2020.

[5] Ghidra. https://www.nsa.gov/resources/everyone/ghidra/ . Accessed on June 22, 2020.

[6] How to adapt to the new threat environment. https://home.kpmg/xx/en/home/insights/2020/05/rise-of-ransomware-during-covid-19.html. Accessed on July 10, 2020.

[7] Incident of the week: Maze ransomware targets conduent and aerospace entity in unrelated incidents. https://www.cshub.com/attacks/articles/incident-of-the-week-maze-ransomware-targets-conduent-and-aerospace-entity-in-unrelated-incidents . Accessed on July 20, 2019.

[8] M-trends 2020- fireeye. https://content.fireeye.com/m-trends/rpt-m-trends-2020. Accessed on July 20, 2020.

[9] Maze: the ransomware that introduced an extra twist. https://blog.malwarebytes.com/threat-spotlight/2020/05/maze-the-ransomware-that-introduced-an-extra-twist/ . Accessed on September 25, 2020.

[10] Pe-parse tool. https://github.com/trailofbits/pe-parse. Accessed on May 05, 2020.

[11] Pin tool. https://software.intel.com/content/www/us/en/develop/articles/pin-a-dynamic-binary-instrumentation-tool.html . Accessed on July 22, 2019.

[12] Ransomware. https://www.knowbe4.com/ransomware. Accessed on October 24, 2019.

[13] Ransomware 2018-2020 - kaspersky lab. https://media.kasperskycontenthub.com/wp-content/uploads/sites/100/2020/05/12075747/KSN-article$_{R}ansomware - in - 2018 - 2020 - 1.pdf. AccessedonJuly20, 2020.$

[14] Ransomware is now the biggest cybersecurity threat. https://www.zdnet.com/article/ransomware-is-now-the-top-cybersecurity-threat-warns-kaspersky/. Accessed on July 20, 2020.

[15] A record year for enterprise threats. https://documents.trendmicro.com/assets/rpt/rpt-2016-annual-security-roundup-a-record-year-for-enterprise-threats.pdf. Accessed on May 20, 2018.

[16] Sophos 2020 threat report. https://www.sophos.com/en-us/medialibrary/Gated-Assets/white-papers/sophos-the-state-of-ransomware-2020-wp.pdf. Accessed on July 20, 2019.

[17] Sophos 2020 threat report. https://www.sophos.com/en-us/medialibrary/pdfs/technical-papers/sophoslabs-uncut-2020-threat-report.pdf. Accessed on July 20, 2019.

[18] Virus total. https://www.virustotal.com/gui/home/upload . Accessed on June 22, 2020.

[19] Wastedlocker ransomware: Abusing ads and ntfs file attributes. https://labs.sentinelone.com/wastedlocker-ransomware-abusing-ads-and-ntfs-file-attributes/ . Accessed on July 28, 2020.

[20] Yara rules. http://virustotal.github.io/yara/. Accessed on July 10, 2020.

[21] N. Andronio, S. Zanero, and F. Maggi. Heldroid: Dissecting and detecting mobile ransomware. In *International Symposium on Recent Advances in Intrusion Detection*, pages 382–404. Springer, 2015.

[22] S. I. Bae, G. B. Lee, and E. G. Im. Ransomware detection using machine learning algorithms. *Concurrency and Computation: Practice and Experience*, page e5422, 2019.

[23] R. Canzanese, S. Mancoridis, and M. Kam. System call-based detection of malicious processes. In *2015 IEEE International Conference on Software Quality, Reliability and Security*, pages 119–124. IEEE, 2015.

[24] A. Continella, A. Guagnelli, G. Zingaro, G. De Pasquale, A. Barenghi, S. Zanero, and F. Maggi. Shieldfs: a self-healing, ransomware-aware filesystem. In *Proceedings of the 32nd Annual Conference on Computer Security Applications*, pages 336–347, 2016.

[25] N. Hampton, Z. Baig, and S. Zeadally. Ransomware behavioural analysis on windows platforms. *Journal of information security and applications*, 40:44–51, 2018.

[26] A. Kharaz, S. Arshad, C. Mulliner, W. Robertson, and E. Kirda. {UNVEIL}: A large-scale, automated approach to detecting ransomware. In *25th {USENIX} Security Symposium ({USENIX} Security 16)*, pages 757–772, 2016.

[27] E. Kolodenker, W. Koch, G. Stringhini, and M. Egele. Paybreak: Defense against cryptographic ransomware. In *Proceedings of the 2017 ACM on Asia Conference on Computer and Communications Security*, pages 599–611, 2017.

[28] S. Poudyal, D. Dasgupta, Z. Akhtar, and K. Gupta. A multi-level ransomware detection framework using natural language processing and machine learning. In *14th International Conference on Malicious and Unwanted Software "MALCON 2019" (MALCON 2019)*, Nantucket, USA, 2019. IEEE.

[29] S. Poudyal, K. P. Subedi, and D. Dasgupta. A framework for analyzing ransomware using machine learning. In *2018 IEEE Symposium Series on Computational Intelligence (SSCI)*, pages 1692–1699. IEEE, 2018.

[30] Y. Takeuchi, K. Sakai, and S. Fukumoto. Detecting ransomware using support vector machines. In *Proceedings of the 47th International Conference on Parallel Processing Companion*, pages 1–6, 2018.

[31] T. K. Tran and H. Sato. Nlp-based approaches for malware classification from api sequences. In *2017 21st Asia Pacific Symposium on Intelligent and Evolutionary Systems (IES)*, pages 101–105. IEEE, 2017.