IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received August 2, 2021, accepted August 25, 2021, date of publication August 31, 2021, date of current version September 10, 2021.

Digital Object Identifier 10.1109/ACCESS.2021.3109260

Analysis of Crypto-Ransomware Using ML-Based Multi-Level Profiling

SUBASH POUDYAL, (Member, IEEE), AND DIPANKAR DASGUPTA, (Fellow, IEEE)
Center for Information Assurance, Department of Computer Science, The University of Memphis, Memphis, TN 38152, USA
Corresponding author: Subash Poudyal (spoudyal@memphis.edu)

ABSTRACT Crypto-ransomware is the most prevalent form of modern malware, has affected various industries, demanding a significant amount of ransom. Mainly, small businesses, healthcare, education, and government sectors have been under continuous attacks by these adversaries. Various static and dynamic analysis techniques exist, but these methods become less efficient as the malware writers continuously trick the defenders. Numerous research of ransomware with AI techniques often lack the behavioral analysis and its correlation mapping. In this work, we developed an AI-powered hybrid approach overcoming the recent challenges to detect ransomware. Specifically, we proposed a deep inspection approach for multi-level profiling of crypto-ransomware, which captures the distinct features at Dynamic link library, function call, and assembly levels. We showed how the code segments correlate at these levels for studied samples. Our hybrid multi-level analysis approach includes advanced static and dynamic methods and a novel strategy of analyzing behavioral chains with AI techniques. Moreover, association rule mining, natural language processing techniques, and machine learning classifiers are integrated for building ransomware validation and detection model. We experimented with crypto-ransomware samples (collected from VirusTotal). One of the machine learning algorithms achieved the highest accuracy of 99.72% and a false positive rate of 0.003 with two class datasets. The result exhibited that multi-level profiling can better detect ransomware samples with higher accuracy. The multi-level feature sequence can be extracted from most of the applications running in the different operating systems; therefore, we believe that our method can detect ransomware for devices on multiple platforms. We designed a prototype, AIRaD (AI-based Ransomware Detection) tool, which will allow researchers and the defenders to visualize the analysis with proper interpretation.

INDEX TERMS Artificial intelligence, hybrid malware analysis, machine learning, multi-level profiling, ransomware detection, reverse engineering.

I. INTRODUCTION

The economic benefits and anonymity has fostered the cyber-criminals to perform continuous ransomware attacks in various sectors.

Recent years have seen attacks being spread as a ransomware-as-a-service model. These attacks are often delivered via phishing campaigns where a user is masqueraded with a seemingly genuine email with malicious links or attachments. The victim often becomes prey to social engineering attacks and lured to click the link or download the malicious attachment. According to the survey done by Sophos [10], 45% of ransomware attacks are via malicious links or attachments in emails. Also, 21% of the attacks are from a remote attack on the server, and the remaining through misconfigured systems and USB devices. Recently, phishing attacks are coming in the form of COVID-19 themed lures and exploit people's concerns over the pandemic and safety of their family members. The bad guys 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 [5]. Other forms of attacks include exploiting vulnerabilities in user systems or in the application they are using. Unpatched systems provide room for remote code execution and privilege escalation. The MS17-010 [6], SMB vulnerability, and EternalBlue exploit made several WannCry ransomware attacks worldwide in 2017 [12].

However, the same vulnerability and exploit is even used today by recent ransomware families, including Ryuk, SamSam, and Satan [9]. The attack exploits systems that

The associate editor coordinating the review of this manuscript and approving it for publication was Tyson Brooks.

This work is licensed under a Creative Commons Attribution-NonCommercial-NoDerivatives 4.0 License.
For more information, see https://creativecommons.org/licenses/by-nc-nd/4.0/

TABLE 1. Recent ransomware families with some properties.

Family	Propagation strategy	Date appeared	Cryptographic technique	C&C Server
Cerber	Email spam, RIG and magnitude exploit kit	2015	RC4 and 2048-RSA	IP range
TeslaCrypt	Angler browser exploit kit	2015	AES-256	Tor anonymity network
Shade	Malicious websites, exploit kits, infected email attachments	2015	256-AES	Tor anonymity network
Locky	Spam campaigns, Neutrino exploit kit, Nuclear exploit kit, RIG exploit kit	2016	RSA and AES	Using DGA algorithm
Dharma	Unprotected RDP port, Spam campaigns	2016	256-AES and 1024-RSA	Random IPs/locations
Stop	Malicious email attachments/advertisements, torrent websites	2018	256-AES and 1024-RSA	Listed address
GandCrab	Spam emails, exploit kits	2018	AES and RSA	Tor anonymity network/DGA
Ryuk	TrickBot and RDP	2018	AES and RSA	Internet-facing Mikrotik router
Anatova	Spear phishing in private p2p network	2019	RSA and Salsa20	Listed address
Maze	Malspam campaign, RDP attacks	2019	ChaCha20 and RSA	Tor network
AgeLocker	Age encryption tool of Google	2020	X25519 (an ECDH curve), ChaChar20-Poly1305, HMAC-SHA256	Tor network
Snake	Malspam campaign, RDP attacks	2020	AES-256 and RSA-2048	Listed address
WastedLocker	Fake browser update	2020	AES-256 and RSA-4096	Tor network
Conti	Phishing emails, Server Message Block	2020	AES-256 and hard-coded public key	Listed address range
Babuk	Spear phishing	2021	ChaCha and Elliptic Curves	NA
Darkside	Spear phishing, unpatched vulnerability	2021	Salsa20 and RSA-1024	Listed address

are not security patched are prey to continuous attacks disrupting regular businesses/services with financial losses and reputation.

Table 1 shows different recent crypto-ransomware families with properties such as propagation strategy, date when it first appeared, cryptographic techniques used, and command and control (C&C) server used.

On May 7, 2021, Colonial Pipeline said that a ransomware attack forced the company to proactively close down operations and freeze IT systems after becoming the victim of a cyberattack [1]. More than 100GB of data got stolen in just 2 hours. Salsa20 and RSA-1024 encryption were used to encrypt user files. Ransomware often comes in two forms: crypto-ransomware and locker ransomware. Crypto-ransomware is more prevalent these days, which encrypts user's data and holds until the ransom is paid while locker ransomware locks the user's system making the system unusable. The locker ransomware often replaces the whole screen with a warning picture with instructions to pay to get the system to an expected stage. Hybrid encryption is more common, which includes the combination of symmetric and asymmetric encryption.

Ransomware attack vectors are dynamic and use sophisticated encryption techniques. They come in various obfuscated and persistent forms making the analysis and detection work more complex. Various anti-ransomware tools and methods have been proposed [16], [17], [19]–[22], [30] and claim to have reasonable detection rates. However, they still fall short at detecting zero-day ransomware attacks, explaining the why question of the claimed better performance, and the methods often work for given malware family only. Moreover, malware with obfuscated code is often bypassed or falsely detected in those approaches. To overcome the limitations of the current prominent methods, we have proposed an automatic hybrid analysis tool with an explainable component showing the relations at multi-level with behavioral chains. We analyze the extracted features at DLL, function call, and assembly level using hybrid analysis, association rule mining, and behavioral chain analysis. The prototype AIRaD tool stands on the proposed architecture and gives the user the flexibility for ease of use with detailed analysis. Furthermore, our method detects zero-day ransomware binaries among a broad range of malware families.

The remainder of this paper is organized as follows. Section II presents the related works on detecting ransomware. In section III we discuss the hybrid reverse engineering techniques applied at multiple levels. Section IV discusses about the different machine learning techniques used. Section V describes the details of the AI-based ransomware detection framework which includes information about behavioral chains, dataset and experiments, and designing a prototype system, AIRaD tool. Future direction and limitations are mentioned in section VI. The paper concludes with a summary in section VII.

II. LITERATURE REVIEW

Most of the ransomware detection-related works are based on either static or dynamic analysis, and a few on hybrid analysis consists of both. Most of the work is based on API features or function calls since these are essential in implementing malware functionalities. Here we discuss some of those notable works.

Alam *et al.* [14] have proposed ransomware prevention via performance counters referred to as RAPPER. They have

used neural networks and Fast Fourier Transformation with hardware performance counters (HPCs) as event traces. Perf, a well-known tool in Linux, is used to monitor the HPCs and observe the performance counters and the system behavior. The authors of this work claim this behavior as a good advantage for getting the feature set for their machine learning model. The accuracy of their approach is based on given ransomware samples. Their work could have shown the efficiency for scaling the learning model and the approach to resilience to obfuscation behavior of ransomware samples.

With an objective to overcome the limitations of supervised learning algorithms, Sharmeen *et al.* [27] have proposed a semi-supervised framework to learn unique ransomware behavioral patterns using deep learning techniques. They claim that their model is scalable to accommodate new variants of malware executable. The ransomware samples were run in an isolated environment using Cuckoo sandbox with varying run time from four to nine minutes. Their approach ignored the samples that could not run in the given experimental settings, making the model less robust. It misses the obfuscated nature of malicious binaries and impacts the detection of obfuscated ransomware samples.

Arabo *et al.* [15] have done process behavior analysis to determine whether a given sample is ransomware or not. The study starts with which APIs are called and how much system resources are used? The dynamic run result is analyzed to get the feature statistics fed into various supervised and unsupervised machine learning models. The analysis is also done using file extension, API calls made, and disk usage. This particular analysis aims to give users a quick alert if the binary under consideration is a possible ransomware executable. However, their approach has low accuracy and lacks false-positive analysis.

Faghihi *et al.* [18] have presented a data-centric detection and mitigation against smartphone crypto-ransomware using dynamic analysis and hash-based techniques. They analyze the user's data along with its entropy values to make detection decisions. API calls related to file operations are intercepted by function hooking techniques. Then entropy and structure of data are monitored to detect and neutralize the ransomware. Their approach claims a high accuracy with a low false-positive rate but shows a low resilience to obfuscation behavior of ransomware executable.

A block cipher algorithm is detected to prevent ransomware infection in work proposed by Kim *et al.* [23]. The sequence and frequency characteristics are obtained from the opcode of binary files. They have considered Alf and Vegard's RISC (AVR) processor microcontroller for their experiment, where they use a convolutional neural network. Their approach is restricted to static analysis and will not capture the run-time behavior of the malware. The authors have claimed a high accuracy of their system. They could have explained how their approach could handle the anti-analysis behavior of the ransomware samples.

Takeuchi *et al.* [30] have proposed ransomware detection using deep inspection of API call sequence and support vector machines as classifiers with 97.48% accuracy and missing rate of 1.64%. They have included the number of occurrences of the given n-gram of API in their vector model. Their proposed feature vector model improved the detection performance with fewer false positives. The model is not tested with other malware families and lacks the explainable component of their chosen machine learning algorithm.

Hampton *et al.* [19] studied ransomware behavior in the Window system and identified API calls specific to it. The frequency of API usage among ransomware and normal binaries is useful for identifying ransomware without comparing the code signature. They claim that their approach can better understand the ransomware strain in terms of API calls. Handling of obfuscated binaries that uses obscure API calls is missing in this research.

Bae *et al.* [16] 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 six different machine learning classifiers. They have reported the highest accuracy of 98.65% and compared with others works. The test environment runs with an execution time limit of five minutes which malware writers often fool. Steps should have been taken to handle various anti-malware analysis techniques deployed by the adversaries. Their approach could have classified a given malware family instead of a whole set of malware.

Hwang *et al.* [20] have proposed a two-stage mixed ransomware detection model using the Markov model and Random Forest. They leveraged the Windows API call sequence pattern to build a Markov model, then used the Random Forest machine learning model to train the remaining data. The overall achieved accuracy was 97.3% with 4.8% FPR. They performed a dynamic analysis in a sandbox environment and grouped collected APIs into different categories. Their approach could have addressed the handling of obfuscated binaries. What if binaries detect the sandbox environment and do not execute at all? Would the accuracy be improved after adopting a hybrid approach? Answers to such queries are also expected.

Ki *et al.* [22] have assigned alphabet letters to API functions and apply a DNA sequence alignment algorithm to extract common API call sequence patterns of malicious processes. They claim that the sequence analysis is a better approach as the malware authors can insert dummy and redundant function calls to make the frequency analysis method ineffective. The experimental result showed an accuracy of 99.8%. However, their approach could not handle the detection of obfuscated binaries.

Khan *et al.* [21] have proposed a digital DNA sequencing engine for ransomware detection using Naive Bayes, Decision stump, and AdaBoost algorithms for classification. The features are obtained from the pre-processed data using Multi-Objective Grey Wolf Optimization (MOGWO) and Binary Cuckoo Search (BCS) algorithms. Then the digital DNA sequence is generated for the selected features using the design constraints of DNA sequence and k-mer

frequency vector. The experiments show the highest accuracy of 87.9% with AdaBoost. This approach has not mentioned the handling of obfuscated binaries. Moreover, more information about the initial feature, dataset, and methods could have been given.

Shaukat *et al.* [28] have proposed the RansomWall tool, which combines the features obtained through static and dynamic analysis. They monitor file operations dedicated to encryption purposes. They have claimed to detect zero-day ransomware samples. Their approach achieved an accuracy of 98.25% with the Gradient tree boosting algorithm. They have listed some of the suspicious Windows cryptographic function calls, but extensive analysis of those is missing.

A two-level(DLL and assembly) frequency analysis was done along with the design of a machine learning framework in one of our previous work. [26]. The framework has the flexibility to choose the best machine learning classifier for the detection task. Cosine similarity was used to measure the similarity among different ransomware families. Cosine similarity identifies the most discrimination features between benign and ransomware samples. Experiments were done with assembly code feature and DLL features separately and in a combination of both. The combined features at two-level achieved better performance with an accuracy of 97.95%. The downside of this approach was the use of only static analysis techniques, which do not handle the obfuscation behavior of malware. Also, the function call features were not included in the experiments, which would have enhanced the detection rates.

Our other previous work [25] deals with multi-level(DLL, function call, assembly) analysis using machine learning and Natural Language Processing (NLP) techniques. The n-gram probabilities, term-frequency, and inverse document frequency (TF-IDF) were used to generate the features at a multi-level. The logistic regression classifier achieved the detection rate of 98.59% for generated TF-IDFs trigrams at a combined multi-level. Though this approach improved the performance of the previous one, it still fell short in handling the obfuscated binaries due to the dependency on static analysis. The shortcomings of our recent approaches motivated us to adopt the hybrid approach of ransomware analysis in [24]. We reverse engineer samples using both static and dynamic analysis techniques leveraging different AI techniques. This approach was able to achieve 99.54% accuracy with 0.005 false-positive rates.

This current version of our research includes the extended and enhanced work of [24] with an improvised framework, more insights and experiments, and a focus on behavioral chains. Unlike other's work, our recent approach leverages hybrid multi-level deep inspection showing the relation of three levels, its contribution to unique signatures, and behavioral chains. Our approach can handle the obfuscated behavior of ransomware binaries as escaping analysis at one level is not enough as we consider till low-level analysis at assembly. Most related works lacks the correlations of ransomware behaviors which could have addressed the explainable AI. Moreover, our prototype tool will detect zero-day ransomware binaries and show the explainable components that contributed to the excellent accuracy of machine learning classifiers.

III. HYBRID REVERSE ENGINEERING AT MULTIPLE LEVELS

Advance reverse engineering techniques are applied at multiple levels, i.e., Dll, function call, and assembly. Details about the methods and multiple levels are discussed in subsections given below.

A. HYBRID REVERSE ENGINEERING

Reverse engineering (RE) is a backward process of re-creating things. It deals with disassembling and dealing with each piece of disassembled components to find the behavior of the original one. In malware analysis, the actual code is rarely known, and there comes the need for RE to see the actual conduct of the malware to detect and defend against its attack.

Ransomware analysis is done either using static, dynamic, or hybrid approaches. Static analysis is the basic one that involves inspection of the binary stored in a hard disk or drive. This analysis reveals information such as compile-time, binary format, imports, exports and strings used. However, the run-time behavior of the ransomware cannot be captured. This shortcoming is achieved using dynamic analysis where a malware sample is run in the memory, and its behavior is revealed to the analyst. A hybrid analysis is the combination of both of these approaches [29], [31].

First, the ransomware and benign samples are reverse engineered using a hybrid approach. Ransomware writers often use various notoriously clever techniques to bypass analysis techniques to evade the defenders. Though, some virtualization tools and environments claim they can overcome the anti-analysis techniques used by cybercriminals. But, this is not as easy as claimed. This is also one of the reasons why we adopted a hybrid approach. Another reason is the ability to capture distinct features and a reasonable detection rate which we will see in the coming sections of this paper. Those binaries which do not run or exit in the middle will be analyzed using a PE parser [7], an open-source tool that helps to reveal properties like DLL used, import and export of functions by the binaries. Dynamic analysis is done using a virtualized environment with Cuckoo sandbox [2], advanced disassembler, and debugger tool from National Security Agency Research Directorate [4], and the dynamic binary instrumentation tool, PIN [8].

Below is a description of the tools and techniques we used for hybrid reverse engineering.

1) DYNAMIC BINARY INSTRUMENTATION

Programmers have used instrumentation techniques to diagnose program crashes, analyze errors, and 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 [8] for DBI. PIN makes tracking of every instruction executed 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 that allows execution and analysis of malware samples in a safe environment [2]. It makes use of the virtualization technique to run malware samples. 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 [4].

3) GHIDRA

Ghidra is a reverse engineering framework developed by the National Security Agency Research Directorate [4]. This tool allows malware researchers to analyze binaries on a variety of platforms. It will enable various features, including disassembly and decompilation. Ghidra's disassembly features help to analyze the behavior of ransomware samples better. Its use made defining the multi-level chains of ransomware possible.

B. FEATURE EXTRACTION AT MULTIPLE LEVELS

The raw output obtained from the hybrid reverse engineering techniques is pre-processed and fed to the feature extraction component. Features are extracted at three levels: DLL, function call, and assembly. Below is the description of the extractor at each level.

1) DLL LEVEL

DLL extractor at the DLL level works via an automated script that parses through the raw reverse engineered output. It gives the list of all DLLs being called by various function calls. DLLs are dynamic link libraries that are subroutines to perform actions such as file system manipulation, navigation, process creation and communication. They are loaded into the memory whenever required and freed from memory whenever our system is lightweight. Thus, it makes effective use of available memory and resources by dynamic linking capability. DLLs have functions that they export and make available to other programs. During a program run, all necessary DLLs are loaded into the memory. Still, the referenced function call is accessed only when needed by locating the memory address where the function code resides. Specific DLLs 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

Function call extractor at function call level is implemented through an automated script that parses through the raw reverse engineered output. It gives the list of all function calls being used by the program in execution, as seen in the execution trace. A function call is a piece of code that has lines of instructions that impact 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 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, are some of the critical behaviors specific to ransomware that we analyze at this level.

3) ASSEMBLY LEVEL

Dynamic binary instrumentation tool PIN is leveraged to get the assembly instructions being used by the program. 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, Arithmetic operation, and Flag control.

IV. USE OF MACHINE LEARNING

Traditional malware detection techniques are primarily based on signatures that adversaries can easily evade. Machine learning systems have better detection capability as they can automate the work of creating signatures. Moreover, they can better detect the previously unseen malware.

ML classifier component leverages the various supervised and unsupervised machine learning classifiers to train, test, and validate the model and decide whether a binary in consideration is ransomware or a benign application. The threshold accuracy is set to 95%. If the binary is classified as ransomware, then we generate an alert and delete the sample. Whereas if the binary is classified as not ransomware, then it is labeled as a benign application. We use Logistic Regression (LR), Support vector machines (SVM), Random Forest (RF), J48, Adaboost with RF/J48 and Neural network.

Through the natural language processing (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 a feature database which later on is fed to the ML classifier.

Data mining techniques like frequency analysis and association rule mining are used to find the hidden features or patterns that can distinguish a given family sample from others. Our approach uses the FP-Growth algorithm, which is 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 advanced 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. Table 2 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 V-A. Table 3 shows a portion of the association rules at the function call level. 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. Similarly, Table 4 shows a portion of the association rules at the assembly level.

TABLE 2. 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]
[ADVAPI32, CRTDLL, GDI32, KERNEL32, OLE32, OLEAUT32, USER32] → [WININET]
[ADVAPI32, CRTDLL, GDI32, KERNEL32, OLE32, OLEAUT32, WININET] → [USER32]
[ADVAPI32, CRTDLL, GDI32, OLE32, OLEAUT32, USER32, WININET] → [KERNEL32]

V. AI-BASED RANSOMWARE DETECTION FRAMEWORK

The computational blocks and flow diagram for hybrid multi-level profiling (HMLP) for ransomware detection are shown in Figure 1. Both ransomware and benign binaries are analyzed using hybrid reverse engineering techniques, and feature extraction at multi-level is done as discussed in section III. The use of machine learning is discussed in section IV. Behavioral chaining is the core component of this framework and is discussed in detail below.

A. BEHAVIORAL CHAINING

A chain is a continuous sequence of components that achieve a specific functionality or an activity. Behavioral chains are collections of functionality chains at different levels or can be linked across multi-level. Functionalities are derived from different ransomware families, which are analyzed. The algorithmic steps to find a behavioral chain are illustrated in algorithm 1.

TABLE 3. 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])
[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]
[DeleteCriticalSection, GetCurrentProcessId, GetLastError, GetTickCount, SetLastError, TlsAlloc, TlsFree] → [TlsSetValue]

TABLE 4. 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]
[add, call, cmp, ficom, les, or, rcr, sbb, xchg] → [lock]
[add, call, cmp, ficom, lock, or, rcr, sbb, xchg] → [mul]
[add, call, ficom, les, lock, or, rcr, sbb, xchg] → [cmp]

According to the algorithmic steps, for each ransomware family, we performed reverse engineering to extract features to build behavioral chain profiles, also dynamic analysis is done running a binary in an isolated virtual environment. Dynamic analysis is real execution of a binary to study the actual behavior and functionality. If some binaries do not run in a virtualized environment, we study the behavior using static analysis techniques.

Figure 2 shows the dynamic run trace of crypto-ransomware via dynamic analysis in a virtualized environment. It shows the function call sequence associated with encryption behavior. Figure 7 shows the assembly-level code to check default user language. Ransomware samples often check the user language to decide whether to execute or not.

We can predict the functionality through disassembly and run trace information; moreover, a deep manual inspection of these results helps create functionality chains at DLL, function call, and assembly level. A self-deletion functionality of malware has multi-level chains as shown in table 5. A major chain can have minor sub-chains. For example, a self-delete chain contains shadow copy delete as sub-chains under the major chain. We ignore the functionality which is commonly

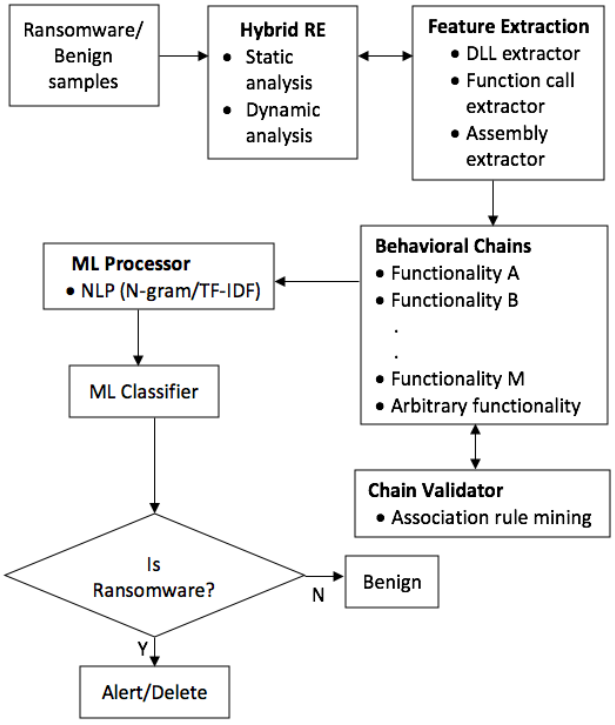

FIGURE 1. Hybrid multi-level profiling based ransomware detection framework.

```
[0056.562] CryptAcquireContextW (in: phProv=0x356ce13, szContainer=0x0, szProvider=0x0, dwPro
dwFlags=0xf0000000 | out: phProv=0x356ce13*=0x3512f1) returned 1
[0056.562] VirtualAlloc (lpAddress=0x0, dwSize=0x9, flAllocationType=0x3000, flProtect=0x40) retu
0x3400000
[0056.562] GetModuleHandleA (lpModuleName="Advapi32.dll") returned 0x23760001
[0056.562] GetProcAddress (hModule=0x23760001, lpProcName="CryptGenRandom") returned 0x66
[0056.563] CryptGenRandom (in: hProv=0x3512f1, dwLen=0x8, pbBuffer=0x35cabd | out: pbBuffer=
returned 1
[0056.563] CryptReleaseContext (hProv=0x3512f1, dwFlags=0x0) returned 1
[0056.563] VirtualFree (lpAddress=0x3400000, dwSize=0x0, dwFreeType=0x6200) returned 1
[0056.563] CryptAcquireContextW (in: phProv=0x354adcf, szContainer=0x0, szProvider="Microsoft
Cryptographic Provider v1.0", dwProvType=0x1, dwFlags=0xf0000000 | out: phProv=0x354adcf*=0x
returned 1
[0056.563] CryptImportKey (in: hProv=0x3512f1, pbData=0x2e40000, dwDataLen=0x223, hPubKey=
dwFlags=0x0, phKey=0x326ce00 | out: phKey=0x326ce00 *=0x3b6230) returned 1
[0056.563] CryptGetKeyParam (in: hKey=0x3b6230, dwParam=0x8, pbData=0x325ced6, pdwDataLen
dwFlags=0x0 | out: pbData=0x325ced6*=0x800, pdwDataLen=0x532bef3*=0x3) returned 1
[0056.563] CryptEncrypt (in: hKey=0x3b6230, hHash=0x0, Final=1, dwFlags=0x0, pbData=0x32e110
pdwDataLen=0x365cf36*=0xd6, dwBufLen=0x200 | out: pbData=0x32e1100*, pdwDataLen=0x324ef
returned 1
```

FIGURE 2. Dynamic run trace of crypto ransomware showing functions called for encryption behavior.

seen in benign applications such as error handling chains. For all other unidentified functionality, we keep it as an arbitrary chain. The next stage is automatic chaining for all given sample binaries. We performed reverse engineering of each sample using our HMLP algorithm to extract the raw features at three levels.

We then leverage the FP-growth association rule mining algorithm. The association rules with support 0.8 and confidence score one are checked if found from the first stage. If found, they will be part of the final feature dataset. All non-matching chains with previously mentioned association rules scores become part of arbitrary chains in the final feature dataset for the given sample. An illustration of a behavioral chain is shown in Figure 3. Here, the acronym Fcall refers to function calls, and Assm refers to assembly instruction.

Algorithm 1 HMLP Behavioral Chaining

Input: PE binaries
Output: Behavioral chains

procedure HMLP_behavioral_chaining(PE binaries)
  Step 1: Manual_chaining_crypto_ransomware
  for one sample from each ransomware families in [Locky, TeslaCrypt, GandCrab, CryptoWall, Cerber, Petya] **do**
    Hybrid RE of binaries
    Extract features at multi-level
    Inspect/analyze run time execution trace
    Identify and define major functionality chains
    Add multi-level ingredients to major chains
    Identify and define minor functionality chains for each major chains
    Add multi-level ingredients to each minor sub chains
    For all remaining chains keep as arbitrary chains
  end for
  Step 2: Automatic_chaining_all_samples
  for each sample in dataset **do**
    Hybrid RE of binaries
    Extract features at multi-level
    FP-growth rule mining at multi-level
    For each chain/rule with confidence 0.8 and score 1
    Check if a rule matches with defined functionality chain from Step 1
    Add chain as a distinct chain to feature dataset if matched
    Add chain as an arbitrary chain to feature dataset if not matched
  end for
  return behavioral chains
end procedure

1) RELATIONSHIP BETWEEN BEHAVIOR CHAIN AND ASSOCIATION RULES

Association rules discover patterns in a given feature set, while behavioral chains are classification of those. We defined behavioral chains and sub-chains, which can be considered a semi-expert system. While many association rules are generated with varying lengths, properly labeling them in the behavioral chain is very important. In particular, behavior chains provide explainability of the association rules.

Figure 4 shows the behavioral chain for system profiling, particularly for the identification of default user and system language, which is obtained via the relationship with the association rules. This 3-layered hierarchy is concerned with profiling system identifiers. Some of the other often profiled system identifiers are keyboard layout, windows version used, the domain used, and CPU identifier. Figure 5 shows how association rule is mapped to the behavioral chains

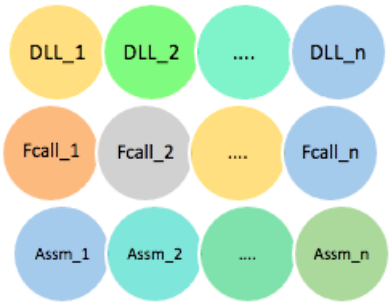

FIGURE 3. A sample arbitrary structure of a behavioral chain.

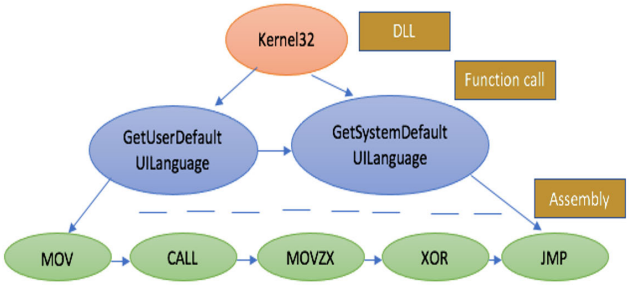

FIGURE 4. Behavioral chain for system profiling with default user and system language.

utilizing our algorithm described earlier which then is used to create detection signature(YARA rules [13]) as shown in Figure 6. The detection signature specifies that it needs all condition from ''a'', three among ''b'', six among ''c'' OR ''d'' and file size(Kilobytes).

2) CHAIN VALIDATOR

Chain validator component aids in the automatic validation of the behavioral chains. We use association rule mining for this. 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 trees. We consider association rules with minimum support threshold two and confidence threshold of 0.8 and check whether it matches the defined chain ingredients. Only the matching chains form the functionality chains A-M. All non-matching chains with previously described support and confidence scores are part of arbitrary functionality chains. We use a novel approach to calculate the ransomware profiling chain ratio. Below, DCR represents the DLL chain ratio, FCR represents a function call chain ratio, ACR represents the assembly chain ratio, and RPCR represents the ransomware profiling chain ratio.

$$DCR = \frac{No.\ ofDLLchainsseen}{Totalno.\ ofDLLchains}$$

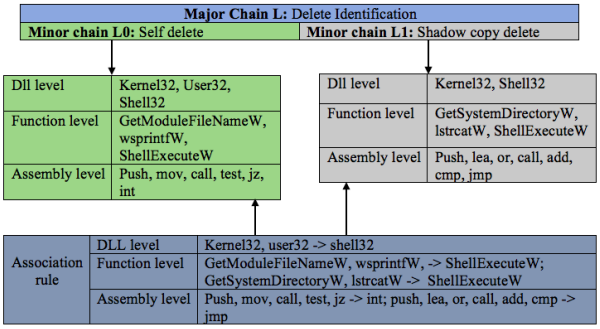

FIGURE 5. Major and minor chain for delete identification behavior mapped with association rules.

```
rule Crypto_Ransomware_Detection_Rule
{
        meta:
            description = "Rule to detect crypto-ransomware"
            author = "Subash Poudyal"
            filetype = "Win32 EXE"
            version = "1.0"
        strings:
            $a1 = "kernel32.dll" fullword ascii
            $a2 = "user32.dll" fullword ascii
            $a3 = "shell32.dll" fullword ascii

            $b1 = "GetModuleFileNameW" fullword ascii
            $b2 = "wsprintfW" fullword ascii
            $b3 = "ShellExecuteW" fullword ascii
            $b4 = "GetSystemDirectoryW" fullword ascii
            $b5 = "lstrcatW" fullword ascii

            $c1 = "push" fullword ascii
            $c2 = "mov" fullword ascii
            $c3 = "call" fullword ascii
            $c4 = "test" fullword ascii
            $c5 = "jz" fullword ascii
            $c6 = "int" fullword ascii
            $c7 = "lea" fullword ascii
            $c8 = "or" fullword ascii
            $c9 = "add" fullword ascii
            $c10 = "cmp" fullword ascii
            $c11 = "jmp" fullword ascii

            $d1 = "Dear %s, your files have been encrypted" ascii wide

        condition:
            ($a*) and 3 of ($b*) and 6 of ($c*) or $d1 and filesize < 32768
}
```

FIGURE 6. Crypto-ransomware signature based on behavior chain.

$$FCR = \frac{No.\ offunctioncallchainsseen}{Totalno.\ offunctioncallchains}$$

$$ACR = \frac{No.\ ofassemblychainsseen}{Totalno.\ ofassemblychains}$$

$$MPCR = \frac{DCR + FCR + ACR}{Totalnumberoflevels}$$

Our HMLP detection approach chains are created at DLL, function call, and assembly level for given ransomware functionality. These chains are made leveraging the feature extraction component that relies on hybrid RE. We first manually inspect thousands of lines of function and activity trace of prominent ransomware families and reach on consensus to define the major chains seen in most ransomware families.

For all other remaining samples, these chains are discovered and validated automatically using the chain validator component, which uses association rule mining.

Each main functionality chain can have sub-chains under them. If there is some overlapping in the main chain, then it is kept under the sub-chain with some variations. If functionality cannot be well defined, then it is kept under an arbitrary functionality chain. Based on crypto-ransomware behavior, we have identified chains from A to M, which incorporate most crypto-ransomware families.

The multi-level analysis is crucial for creating unique ransomware signatures.

This analysis is based on static and dynamic analysis, observing the function call trace, assembly code, and disassembled code blocks.

A brief explanation of each chain follows below.

Chain A deals with the initial setup.

Chain A0 uses *GetStartupInfoW* which gets information related to window station, desktop and appearance of the main window. *HeapSetInformation* enables features to use heap as a data structure. *HeapCreate* and *GetProcessHeap* are used by the HeapHandle parameter. *GetModuleHandleW* gets a handle for a given module, either an executable or a DLL file. This handle is referenced later to request the necessary function calls. *GetProcAddress* gets the memory address of an exported DLL function. The first parameter is the handle to the DLL and the second parameter is the function name exported from that DLL. For example, *CryptGenRandom* is exported from Advapi32 DLL. *FlsAlloc* allocates a fiber local storage index. Any fiber in the process can subsequently use this index to store and retrieve values that are local to the fiber [3]. It setups with three sub-functions which are *FlsGetValue, FlsSetValue* and *FlsFree*.

Chain A1 deals with console-setup. *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 system in use supports the specified processor feature.

Chain A2 deals with system services with error handling. *GetLastError* gets the last error code for the calling thread of the given process. Threads do not overwrite each other's error code. *GetCurrentThreadId* gets the identifier value for the thread whose error code for execution of a particular 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 A3 deals with module enumeration. *GetModuleFileName* loads the malware executable and *GetModuleHandle* gets the handle to the custom malware DLL with obfuscated functions. The obfuscation behavior can be captured at the assembly level.

Chain B deals with time tracking and anti-analysis behavior of the ransomware. *GetTickCount* gives the time in milliseconds that have passed since the system was started.

mov ebx, 0xdbba0
call dword ptr [kernel32.dll::GetTickCount]
cmp eax, ebx

The 0xdbba0 value i.e 900000 milliseconds or 15 minutes is compared with the time obtained via *GetTickCount*. If the user system is active for less than 15 minutes, the malware will not execute, giving us a false impression of a benign application. Malware writers try to evade malware analysis being done in a virtual environment. Generally, such an analysis environment runs for less than 15 minutes or even less than that.

GetSystemInfo and *GetNativeSystemInfo* use *dwNumberOfProcessors* method to check the number of processors running in a system. If the system has only one processor then the malware writers label it as an analysis environment and may not execute at all. *GetSystemInfo* often invokes native low level function call *NtQuerySystemInformation* to get the number of cores used in a system.

Chain C deals with DLL and function loading. *LoadLibraryW* loads the specified DLL into the address space of the calling process. *GetModuleHandleW* gets the handle object for that particular DLL. *GetProcAddress* receives the address of an exported function from that specified DLL. This chain of function calls is the action triggering points of the code section while executing a binary.

Chain D deals with access elevation.

In **Chain D0**, *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*.

Chain D1 is used to bypass user access control by elevating privilege to an admin level.

Chain D2 contains functions that destroy previously created handles for access and privilege escalation.

Chain E is used by ransomware for process enumeration. *CreateToolhelp32Snapshot* is used to create a snapshot of processes, heaps, threads, and modules. Malware often uses this function as part of code that iterates through processes or threads. This snapshot function is called during different functionality blocks such as loading DLLs, loading application processes, and loading anti-virus processes. *Process32FirstW* gets information about the first process seen in a system snapshot. This is used to enumerate processes from a previous call to *CreateToolhelp32Snapshot*. Malware often enumerates through processes to find a process to inject into. Ransomware does not want applications

running as it might affect their encryption operation as applications often lock the files that are open or being used. *lstrcmpiW* compares all observed processes with the hard-listed ones to kill out; a similar comparison is made for observed anti-virus software to kill them. Some of the killed-out processes are mydesktopqos.exe, sqlbrowser.exe, sqlservr.exe, msftesql.exe, mysqld.exe, excel.exe, msaccess.exe, outlook.exe, winword.exe, and wordpad.exe. The malware process is often run under explorer.exe.

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 deals with system profiling.

Chain G0 is concerned with profiling system identifiers. Some of the often profiled system identifiers are keyboard layout, windows version used, the domain used, and CPU identifier, etc. *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 registry key object is *KEY_EXECUTE* (0×20019) which is equivalent to *KEY_READ* and Combines the *STANDARD_RIGHTS_READ*, *KEY_QUERY_VALUE*, *KEY_ENUMERATE_SUB_KEYS*, and *KEY_NOTIFY* values.

When *lpSubKey* = "Keyboard Layout Preload" it is inferred that malware is trying to know about keyboard layout. Similarly, other registry key values that are revealed include:

- *lpSubKey* = "Control Panel International"
- *lpSubKey* = "Keyboard Layout Preload"
- *lpSubKey* = "SOFTWARE Microsoft Windows NT CurrentVersion"

RegQueryValueExW receives the type and data for the specified open registry key while the *RegCloseKey* closes all open handles of the registry keys.

Chain G1 deals with user interface language reveal. *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 7 shows its usage.

Chain G2 deals with checking malware footprint if it is already there in an infected system. If this footprint is already available, then the malware will not execute; else, a new footprint is created. Generally, a locked file is created with a hidden attribute at C drive as a footprint. The 8 characters hex as a footprint is obtained from the volume serial number via *GetVolumeInformationW* function. A similar hidden footprint is left at each folder location before encryption. The *wsprintfW* writes the formatted data to the buffer while *CreateFileW* function creates a lock file as a footprint.

Chain G3 deals with creating a unique user id to identify the victim's system. A unique ransom id is calculated using the *RtlComputeCrc32* function, which uses the CPU

```
00402634 c7 45 ec       MOV       dword ptr [EBP + local_18],0x444
         44 04 00 00
0040263b c7 45 f0       MOV       dword ptr [EBP + local_14],0x818
         18 08 00 00
00402642 c7 45 f4       MOV       dword ptr [EBP + local_10],0x819
         19 08 00 00
00402649 c7 45 f8       MOV       dword ptr [EBP + local_c],0x82c
         2c 08 00 00
00402650 c7 45 fc       MOV       dword ptr [EBP + local_8],0x843
         43 08 00 00
00402657 ff 15 78       CALL      dword ptr [->KERNEL32.DLL::GetUserDefaultUILan...
         e0 40 00
0040265d 0f b7 f0       MOVZX     ESI,AX
00402660 ff 15 30       CALL      dword ptr [->KERNEL32.DLL::GetSystemDefaultUIL...
         e1 40 00
00402666 0f b7 d0       MOVZX     EDX,AX
00402669 33 c0          XOR       EAX,EAX
0040266b eb 03          JMP       LAB_00402670
```

FIGURE 7. Disassembled code to check default user language.

name and the windows volume serial number as parameters. *RtlComputeCrc32* calculates the CRC32 checksum of a block of bytes. There may be other ways to create a unique user id, for example, based on mac address or a combination of system information and a unique random number. Various cryptographic algorithms can be used as well.

Chain G4 deals with revealing information about disk usage. *GetDriveTypeW* determines whether a disk drive is a removable, fixed, CD-ROM, RAM disk, or network drive. Parameter *lpRootPathName* gives the root directory for the drive. This function returns a value from 0 to 6, 3 being fixed hard drive. *GetDiskFreeSpaceW* receives information about the given disk. It also reveals how much free space is available on the disk.

Chain H deals with encryption setup, which is the most crucial action performed by ransomware.

Chain H0 is concerned with RSA key pairs generation. The *CryptAcquireContextW* function is used to acquire a handle to a key container implemented by either a 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 an *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 H1 deals with private keys and the nonce generation. Private keys can be generated using various cryptography algorithms such as AES, DES, and Salsa20. Most ransomware encrypts user files using a hybrid approach. Locally generated keys encrypt the victim's files, and those keys, in turn, are encrypted by the attacker's public key. Only attackers corresponding private keys can

decrypt the locally generated key. The purpose of function call sequences in this chain has already been discussed before except for *CryptGenRandom*. Random IV and random keys are generated using getRandomBytes() method available via *CryptGenRandom* function.

Chain H2 encrypts the locally generated keys by using RSA public key generally obtained via the .data section. RSA private key is required from malware writers to decrypt private or local keys and finally decrypt the encrypted user files.

Chain H3 is used for storing cryptographic key pairs. *RegCreateKeyExW* creates a specified registry keys while *RegSetValueExW* sets its data and type. Ransomware, including Gancrab family ransomware, often stores its encrypted RSA and Salsa20 keys in the registry.

Chain H4 creates a new thread to encrypt the user files. Encryption setup is in one thread, while encryption occurs in different threads. Thread creation is done using *CreateThread*. The main thread waits for all threads running on the current drive to finish by calling *WaitForMultipleObjects*. As soon as one drive is finished and all its threads end, the next drive is encrypted, and it continues until all drives have been encrypted. *WaitForSingleObject* waits for a single object to finish or for the time-out interval to elapse.

Chain I deals with file encryption.

Chain I0 deals with file encryption via Crypto API.

At first, ransomware iteratively finds the next files in a given folder to encrypt using *FindNextFileW* then writes the filename with 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 system's normal functioning and should not encrypt the ransom message. *lstrcmpiW* is used to compare 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 called twice: once to generate a random 32-byte key and next time to get an 8-byte nonce. This differs among the type of symmetric encryption technique used. *GetModuleHandleA* refers to the handle for *Advapi32* DLL and *GetProcAddress* gets the *CryptGenRandom* function. After the random key and nonce generation, the cryptographic handle is released and allocated virtual memory is freed.

The next *CryptAcquireContextW* is setting the cryptographic handle ready to encrypt the local private key and nonce using malware writers RSA public key. Again, the next *CryptAcquireContextW* is to encrypt the user generated RSA private key with Salsa20 keys. Here *CryptImportKey* imports the necessary keys and *CryptGetKeyParam* gets data that handle the operations of a key. *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.

The same sequence of calls(*CryptAcquireContextW* through *CryptReleaseContext*) is observed as just before but this time to encrypt data portion from the user's file obtained via *FindNextFileW*.

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. *SetFilePointerEx* moves the file pointer to the specified location, and *WriteFile* writes the given data of buffer pointer to the specified file. Finally, the *MoveFileW* function moves the 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 *CryptGenRandom* function.

Chain I1 This chain deals with encryption via CNG (Microsoft Cryptographic API Next Generation) implementation. Newer versions of ransomware, including Petya, uses this.

The *CryptAcquireContextW*is setting the cryptographic handle ready using the latest CNG. *CryptGenKey* generates symmetric keys to encrypt local files. For example, ff Algid = $0 \times 660e$, then it refers to 128-bit AES keys as symmetric keys. *CryptGetKeyParam* gets data that handle the operations of a key. *PathCombineW* Concatenates two strings that represent properly formed paths into one path. It also concatenates any relative path elements example: C:/ and *. *FindFirstFileW* searches a directory for a file or subdirectory with a name that matches a specific name (or partial name if wildcards are used). The wildcard * is used to specify all files. If no files are found, and instead, a sub-directory is found, it will use *PathCombineW* as an immediate next function. Since drives generally have many directories and sub-directories, it will again call *PathCombineW*. Example: C:/ and Windows.

FindNextFileW continues a file search from a previous call to the *FindFirstFile*, *FindFirstFileEx*, or *FindFirstFileTransacted* functions. If a file is found, it combines that filename with the found path using *PathCombineW*. *PathFindExtensionW* returns the filename extension. *wsprintfW* function writes the string in .extension format to the buffer. *StrStrIW* compares this string with its given list of hardcoded extension strings. If it returns 1 meaning match found then this particular file is encrypted.

CryptEncrypt function does the real encryption of text or strings. Instead of *WriteFile* as seen in the previous chain, *FlushViewOfFile* writes the encrypted string to the disk, i.e., data obtained from a buffer of the *CryptEncrypt* function is written to the same file causing the file content to be replaced by the encrypted text. *UnmapViewOfFile* function unmaps a mapped view of a file. It removes the working set entry for each unmapped virtual page being used previously. Lastly, *CloseHandle* closes the open handle of the file that was to be encrypted.

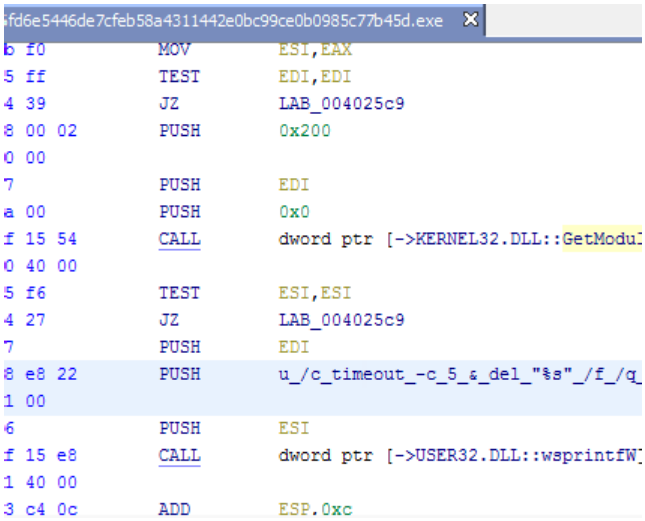

FIGURE 8. Disassembled code for self-deletion of ransomware binary.

Chain J deals with creating a ransom message. *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 the length of the text to be written while *WriteFile* is used to write the ransom note to the specified file. Finally, *CloseHandle* closes the filehandle 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 user machines.

Chain L deals with delete operations by ransomware.

Chain L0 deals with self-delete.

GetModuleFileNameW gets the malware executable location while the function *wsprintfW* writes the previously obtained command line parameter to buffer. This is also shown in figure 8 *ShellExecuteW* function via *lpFile* parameter value as *cmd.exe* executes the given command. This is a good indication of malicious activity. Why would a normal program execute a command prompt or other executable unless specified by a user? For example, web browser firefox.exe would execute its portable executable file unless we give some explicit execution commands to execute other applications. Malware can specify the number of seconds to wait. It would wait till the time elapses or until the user presses any key. The DLL, function call, and assembly used for this chain are shown in Table 5.

Chain L1 deletes the shadow copies. *GetSystemDirectoryW* gets the location to system32 director then concatenates wbem/wmic.exe to it using *lstrcatw*. Shadowcopy delete is a parameter that wmic.exe takes. It deletes shadow copies that are created when system restore points are made. These are generally backup files created by system restore operation.

Chain M 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.

TABLE 5. Chain L: Self deletion of ransomware binary.

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

B. DATASET AND EXPERIMENTS

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

1) DATASET USED

The focus of this work was an analysis of crypto-ransomware and proposing an efficient detection framework. Accordingly, the need for recent crypto-ransomware binaries was fulfilled by VirusTotal [11] which is trusted by most of the researchers among others. Both the binary executable and analysis result in Json format was received. The pre-computed Json result was used to classify the given executable. Moreover, VirusTotal provides an easy to use API which can be leveraged to compare the detection by several anti-virus engines for any of the given sample. We collected 2600 malware samples, including 550 crypto ransomware from VirusTotal [11] and the time period of collection was last six month of the year 2019. The labelling of malware samples is based on the detection category by Malwarebytes, Kaspersky and Microsoft as a reference as given in the Json result of VirusTotal API. These three anti-virus engines gave consistent naming for crypto-ransomware. The malware sample distribution and the size range are shown in table 6. Also, 540 benign application samples were collected from Windows 7/10 OS and open-source applications. It includes normal to advance programs such as *cmd.exe*, *explorer.exe*, *bitlocker.exe*, *firefox.exe*, *openssl.exe*, *taskkill.exe*, *ssh-agent.exe*, *winScp.exe*, *ssh-keygen.exe* and so on. Among them, there are certain samples such as *Openssl*, *WinScp*, and *BitLocker* which uses cryptographic and communication operations which are also the properties seen in the ransomware samples. Normal binaries size varied from

TABLE 6. Malware families and number of samples (with sizes) used.

MALWARE TYPE	NO. OF SAMPLES	SIZE RANGE
CRYPTO RANSOMWARE	550	17.1KB - 31.4MB
ADWARE	524	8.1KB - 11.7MB
BACKDOOR	498	22.4KB - 21MB
TROJAN	513	5.1KB - 25.5 MB
WORM	515	51KB - 17.5 MB

16.4 KB to 36.3 MB which resembled the size of ransomware binaries.

2) EXPERIMENTAL PROTOCOLS AND EVALUATION MEASURES

The experiments were performed using coding in python, bash script, along with machine learning libraries. Malware samples were experimented with within six virtualized environments set up with five i7 processors.

We used widely used performance metrics of True Positive Rate (TPR), False Positive Rate (FPR), Precision, Recall, F-measure, and accuracy for performance evaluation. These metrics were calculated from the confusion matrix of a machine learning classifier.

3) EXPERIMENTAL RESULTS

The experiment done with combined multi-level features with term frequency for two classes (ransomware and benign samples) is shown in table 8. SVM and Adaboost with J48 reported the second-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. A Neural network with two hidden layers, ReLu activation function, and L1-L2 regularizers, was used as an unsupervised learning technique, which achieved the highest accuracy of 99.69% and a false positive rate 0.005. This achieved improved detection due to improved optimization during training phases.

Another experiment done with TF and tri-gram TF-IDF for the same two classes as above is shown in the table 9. This table shows some improvements to the previous one due to the use of the tri-gram TF-IDF approach. SVM reported the highest accuracy with 99.72% and the lowest false positive rate of 0.003. SVM performed better than other algorithms because SVM works more effectively to handle high dimensional spaces, and there is less overlapping among target classes. The improved accuracy seen here than the similar previous approaches [25], [26] is due to the hybrid reverse engineering techniques used, which builds a unique feature set.

Even the experiments done with multiple malware classes show promising results with an overall accuracy of 94.6% with the J48 machine learning classifier as shown in table 10. The confusion matrix for the same experiment is shown at figure 9. The table and confusion matrix shows the high true positive rate and very low false-positive rate for various malware classes, including the normal class. Here, the Adaboost with J48 outperformed SVM because this experiment included more than 3000 samples and had more overlapping than the previous dataset with two classes. J48 decreases the complexity of the final classifier by effective pruning and thus handling the problem of overfitting while the Adaboost further boosts its classification tasks. TPR and FPR score inspection for multiple classes, including normal samples, show our approach's performance is not degraded even in the case of multiple malware classes.

Table 11 shows the ransomware profiling chain ratio, including other scores for six crypto-ransomware family samples. A chain score close to 1 is expected to determine the sample as ransomware. A score equal to one is a perfect score. The threshold chain ratio score is chosen as 0.8 for RPCR. The table shows all the samples to achieve this score and can be determined as a ransomware sample.

4) COMPARISON WITH OTHERS

Many works have been done using API or opcode analysis for ransomware detection that leverages portable executable file format. However, we have compared the recent and notable ones in Table 7. This table shows different features used, classification techniques, resilience to obfuscation, accuracy, and false-positive rate of the proposed model. The description of each work is already discussed in the literature review section. Here, we summarize the table and note the differences with our work. Most of the compared works deal with dynamic analysis and a few with static and hash-based analysis. But our approach deals with hybrid analysis which is a combination of both static and dynamic analysis techniques. The advantage of hybrid analysis is its ability to capture features and behaviors without executing the binaries. Static analysis, though it can achieve a high code coverage, misses the actual run-time conduct. However, there are also few downsides of dynamic analysis like missing parameter values, and we may not capture the actual behavior or miss certain notable behaviors of ransomware executables. The significance of the hybrid approach, along with the feature capture at Dll, function call, and assembly level, can be observed by our work which also achieved high accuracy and lowest false positive rate, among other results. Another notable difference is how resilient your approach is to obfuscation. Most related work has low resilience to obfuscation and fails to explain whether their model can detect obfuscated binaries. Our approach has high resilience to obfuscation as we try to capture malware at three levels; even if we miss at the top Dll level, we can capture at either function call level or the lowest assembly level. Malware writers often trick the defenders by adding logic bomb codes or junk codes to hide their real behavior. Our unique hybrid analysis method using advanced tools, including intel's PIN and NSA's Ghidra, allows capturing the obfuscated behavior and creating a rich feature set for machine learning training models. Thus, we see a high potential for our work compared to other related works.

TABLE 7. Comparison based on various factors of our framework against existing approaches.

Analysis/ Reference	Feature	Classification	Accuracy	FPR
Static [23]	Opcode	CNN	97%	NA
Dynamic [20]	API	RF, DNN	97.3%	0.048
Dynamic [19]	API	Frequency analysis	NA	NA
Dynamic [14]	HPC event traces	ANN, FFT	NA	NA
Dynamic [15]	Dlls, system resources	Supervised, unsupervised	75.01%	NA
Dynamic [27]	API	Deep learning	95.96%	0.059
Dynamic [16]	API	Supervised	98.65%	NA
Hash based [18]	Data entropy	NA	99.24%	0.0049
Hybrid [Ours]	Dll, function, assembly	Supervised, unsupervised	99.72%	0.001

TABLE 8. Machine learning algorithms' evaluation for multi-level combined approach with TF for two classes (Ransomware and benign samples).

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
Neural network	1.0	0.005	0.993	1.0	0.996	99.69

TABLE 9. Machine learning algorithms' evaluation for multi-level combined approach with TF and Tri-gram TF-IDFs for two classes (Ransomware and benign samples).

Machine learning Classifier	TPR	FPR	Precision	Recall	F-measure	Accuracy (%)
Logistic Regression	0.996	0.004	0.996	0.996	0.996	99.63
Support Vector Machine	0.997	0.003	0.997	0.997	0.997	99.72
Random Forest(RF)	0.990	0.010	0.990	0.990	0.990	98.99
J48	0.996	0.004	0.996	0.996	0.996	99.63
Adaboost with RF	0.992	0.008	0.992	0.992	0.992	99.17
Adaboost with J48	0.995	0.005	0.995	0.995	0.995	99.54

TABLE 10. Machine learning algorithms' evaluation for multi-level combined approach with TF for multi-class malwares using Adaboost with J48 (Malware families and benign samples; accuracy: 94.58%).

TPR	FPR	Precision	Recall	F-measure	Class
0.933	0.015	0.926	0.933	0.930	Adware
0.934	0.013	0.930	0.934	0.932	Backdoor
0.993	0.001	0.996	0.993	0.994	Normal
0.955	0.010	0.955	0.955	0.955	Ransomware
0.889	0.020	0.896	0.889	0.892	Trojan
0.969	0.006	0.969	0.969	0.969	Worm

TABLE 11. RPCR calculation for different crypto ransomware families.

RANSOMWARE FAMILY	DCR	FCR	ACR	RPCR
LOCKY	0.78	0.94	0.81	0.84
TESLACRYPT	0.81	0.95	0.85	0.87
GANDCRAB	0.88	0.93	0.89	0.9
CRYPTOWALL	0.85	0.91	0.81	0.85
CERBER	0.88	0.95	0.88	0.903
PETYA	0.78	0.81	0.83	0.806

C. DESIGNING A PROTOTYPE SYSTEM AIRaD

We designed a prototype system based on the proposed HMLP approach described earlier. The prototype that we developed is referred to as AIRaD (AI-powered Ransomware Detection) tool. This tool is in the process of development. Figure 10 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 the rightmost column shows buttons to choose either one level of analysis or multi-level analysis. 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. The user can download the summary report for record-keeping and further analysis. This tool is built upon the foundations provided by our proposed architecture and can be considered as an explanatory AI tool as it identifies the signature generating features.

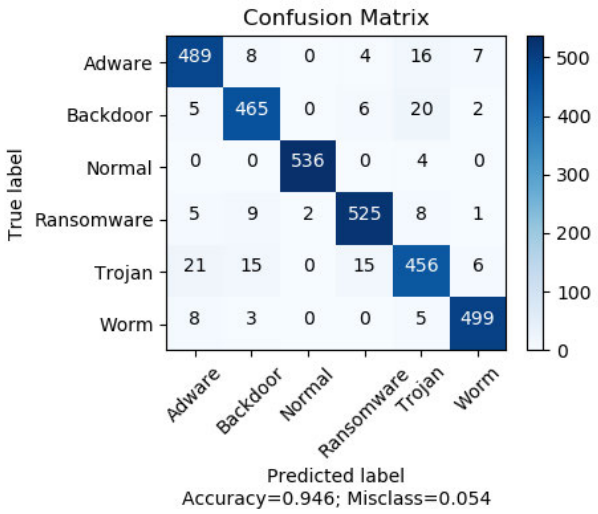

FIGURE 9. Confusion matrix for multi-class malware using Adaboost with J48.

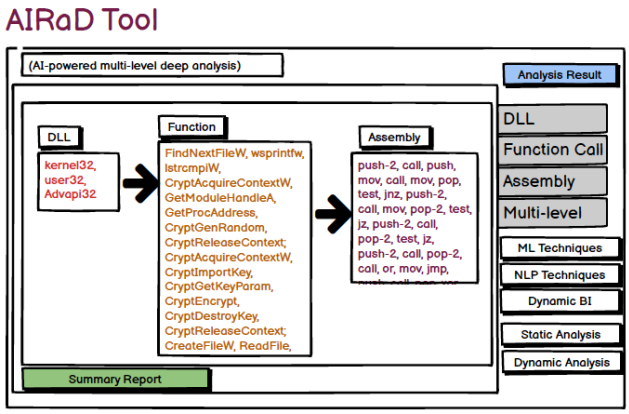

FIGURE 10. Snapshot of AIRaD tool with associated analysis components and summary report.

VI. FUTURE DIRECTION AND LIMITATION

The proposed HMLP framework shows great promise for security-related companies and the research community due to its unique way of integrated analysis with a high detection rate and very low false positives. However, there are certain limitations and constraints for this approach due to various factors. The experiment for this approach has been conducted with a fair amount of samples, i.e., around 3000. In the future, we plan to increase the number of samples and also train our model with advance deep learning techniques.

Due to the dynamic nature of malware, some samples did not go well with dynamic analysis; in that case, instead of exploring deobfuscation techniques, we choose static analysis methods as it was out of the scope of this paper. For behavioral chaining with first phase manual chaining few crypto-ransomware families(six families) were only considered. This might have a partial match or full match with the latest version that we missed or the upcoming version of ransomware samples. However, we found that the crypto-ransomware functionality was uniformly found in several other ransomware families that we analyzed. A periodic evaluation of chain ingredients to incorporate the new variants will make our framework more robust, and automating this task will be a part of future work. Future work will include a wide range of ransomware families. To improve the performance of this framework, one possible direction for extending this work will be to use cloud computing with parallel processing capabilities.

VII. CONCLUSION

In this paper, we performed a deep forensic analysis of crypto-ransomware using hybrid multi-level profiling. We adopted a unique approach of behavioral chaining along with association rule mining and AI techniques. Hybrid multi-level inspection at DLL, function call, and assembly revealed unique behavioral chains that help create unique ransomware signatures and a distinguishing dataset for the machine learning model. Our approach is validated with experiments where we achieved high accuracy with low false positives. Results show that one of the machine learning algorithms achieved the highest accuracy of 99.72% and a false positive rate of 0.003 with two class datasets. Experiments done at multi-class malware families also revealed a reasonable accuracy rate (94.6%) with a very low false-positive rate of 0.001. Ransomware behavioral profiling chain ratio is a novel approach to identify ransomware binary, and it shows significant detection accuracy. The following claims support the novelty of this work.

- Claim 1: Behavioral-based multi-level chains form a unique feature set for advanced malware detection.
- Claim 2: Chaining multi-level features using association rule mining and hybrid analysis automates the task of feature selection.
- Claim 3: Automated hybrid analysis tool for malware detection with an explainable component showing the relations at multi-level with behavioral chains.

We believe that the proposed HMLP approach provides a robust analysis of ransomware (and advanced malware); however, further testing with wild malware be needed for practical use.

REFERENCES

[1] *Colonial Pipeline Attack: Everything You Need to Know*. Accessed: May 11, 2021. [Online]. Available: https://www.zdnet.com/article/colonial-pipeline-ransomware-attack-everything-you-need-to-know
[2] *Cuckoo Automated Malware Analysis*. Accessed: Jul. 22, 2020. [Online]. Available: https://cuckoosandbox.org
[3] *FlsAlloc Function*. Accessed: Jun. 20, 2020. [Online]. Available: https://docs.microsoft.com/en-us/windows/win32/api/fibersapi/nf-fibersapi-flsalloc
[4] *Ghidra*. Accessed: Aug. 22, 2020. [Online]. Available: https://www.nsa.gov/resources/everyone/ghidra/
[5] *How to Adapt to the New Threat Environment*. Accessed: Oct. 10, 2020. [Online]. Available: https://home.kpmg/xx/en/home/insights/2020/05/rise-of-ransomware-during-covid-19.html

[6] *Microsoft Windows SMB Server (MS17-010) Vulnerability*. Accessed: Jul. 10, 2020. [Online]. Available: https://www.cirt.gov.bd/microsoft-windows-smb-server-ms17-010-vulnerability/
[7] *Pe-Parse Tool*. Accessed: Jul. 5, 2020. [Online]. Available: https://github.com/trailofbits/pe-parse
[8] *Pin Tool*. Accessed: Jul. 22, 2019. [Online]. Available: https://software.intel.com/content/www/us/en/develop/articles/pin-a-dynamic-binary-instrumentation-tool.html
[9] *Risksense Spotlight Report Exposes Top Vulnerabilities Used in Enterprise Ransomware Attacks*. Accessed: Jul. 10, 2020. [Online]. Available: https://risksense.com/press_release/risksense-spotlight-report-exposes-top-vulnerabilities-used-in-enterprise-ransomware-attacks/
[10] *Sophos 2020 Threat Report*. Accessed: Jul. 20, 2019. [Online]. Available: https://www.sophos.com/en-us/medialibrary/Gated-Assets/white-papers/sophos-the-state-of-ransomware-2020-wp.pdf
[11] *Virus Total*. Accessed: Jun. 22, 2020. [Online]. Available: https://www.virustotal.com/gui/home/upload
[12] *What is Wannacry Ransomware and Why is it Attacking Global Computers?* Accessed: Jun. 20, 2020. [Online]. Available: https://www.theguardian.com/technology/2017/may/12/nhs-ransomware-cyber-attack-what-is-wanacrypt0r-20
[13] *Yara Rules*. Accessed: Oct. 10, 2020. [Online]. Available: http://virustotal.github.io/yara/
[14] M. Alam, S. Sinha, S. Bhattacharya, S. Dutta, D. Mukhopadhyay, and A. Chattopadhyay, "RAPPER: Ransomware prevention via performance counters," 2020, *arXiv:2004.01712*. [Online]. Available: http://arxiv.org/abs/2004.01712
[15] A. Arabo, R. Dijoux, T. Poulain, and G. Chevalier, "Detecting ransomware using process behavior analysis," *Procedia Comput. Sci.*, vol. 168, pp. 289–296, Jan. 2020.
[16] S. I. Bae, G. B. Lee, and E. G. Im, "Ransomware detection using machine learning algorithms," *Concurrency Comput., Pract. Exp.*, vol. 32, no. 18, Sep. 2020, Art. no. e5422.
[17] R. Canzanese, S. Mancoridis, and M. Kam, "System call-based detection of malicious processes," in *Proc. IEEE Int. Conf. Softw. Quality, Rel. Secur.*, Aug. 2015, pp. 119–124.
[18] F. Faghihi and M. Zulkernine, "RansomCare: Data-centric detection and mitigation against smartphone crypto-ransomware," *Comput. Netw.*, vol. 191, May 2021, Art. no. 108011.
[19] N. Hampton, Z. Baig, and S. Zeadally, "Ransomware behavioural analysis on Windows platforms," *J. Inf. Secur. Appl.*, vol. 40, pp. 44–51, Jun. 2018.
[20] J. Hwang, J. Kim, S. Lee, and K. Kim, "Two-stage ransomware detection using dynamic analysis and machine learning techniques," *Wireless Pers. Commun.*, vol. 112, no. 4, pp. 2597–2609, Jun. 2020.
[21] F. Khan, C. Ncube, L. K. Ramasamy, S. Kadry, and Y. Nam, "A digital DNA sequencing engine for ransomware detection using machine learning," *IEEE Access*, vol. 8, pp. 119710–119719, 2020.
[22] Y. Ki, E. Kim, and H. K. Kim, "A novel approach to detect malware based on API call sequence analysis," *Int. J. Distrib. Sensor Netw.*, vol. 11, no. 6, Jun. 2015, Art. no. 659101.
[23] H. Kim, J. Park, H. Kwon, K. Jang, and H. Seo, "Convolutional neural network-based cryptography ransomware detection for low-end embedded processors," *Mathematics*, vol. 9, no. 7, p. 705, Mar. 2021.
[24] S. Poudyal and D. Dasgupta, "AI-powered ransomware detection framework," in *Proc. IEEE Symp. Ser. Comput. Intell. (SSCI)*, Dec. 2020, pp. 1154–1161.
[25] S. Poudyal, D. Dasgupta, Z. Akhtar, and K. Gupta, "A multi-level ransomware detection framework using natural language processing and machine learning," in *Proc. 14th Int. Conf. Malicious Unwanted Softw. (MALCON)*, Nantucket, MA, USA, 2019, pp. 1–9.
[26] S. Poudyal, K. P. Subedi, and D. Dasgupta, "A framework for analyzing ransomware using machine learning," in *Proc. IEEE Symp. Ser. Comput. Intell. (SSCI)*, Nov. 2018, pp. 1692–1699.
[27] S. Sharmeen, Y. A. Ahmed, S. Huda, B. Ş. Koçer, and M. M. Hassan, "Avoiding future digital extortion through robust protection against ransomware threats using deep learning based adaptive approaches," *IEEE Access*, vol. 8, pp. 24522–24534, 2020.
[28] S. K. Shaukat and V. J. Ribeiro, "RansomWall: A layered defense system against cryptographic ransomware attacks using machine learning," in *Proc. 10th Int. Conf. Commun. Syst. Netw. (COMSNETS)*, Jan. 2018, pp. 356–363.
[29] R. Sihwail, K. Omar, and K. A. Z. Ariffin, "A survey on malware analysis techniques: Static, dynamic, hybrid and memory analysis," *Int. J. Adv. Sci., Eng. Inf. Technol.*, vol. 8, nos. 4–2, p. 1662, Sep. 2018.
[30] Y. Takeuchi, K. Sakai, and S. Fukumoto, "Detecting ransomware using support vector machines," in *Proc. 47th Int. Conf. Parallel Process. Companion*, Aug. 2018, pp. 1–6.
[31] Y. K. B. M. Yunus and S. B. Ngah, "Review of hybrid analysis technique for malware detection," *IOP Conf. Ser., Mater. Sci. Eng.*, vol. 769, Jun. 2020, Art. no. 012075.

SUBASH POUDYAL (Member, IEEE) received the bachelor's degree in computer science and information technology from St. Xavier's College, Tribhuvan University, in 2011, and the Ph.D. degree in computer science from The University of Memphis, in August 2021. He is currently a Cyber-Security Researcher focusing on malware analysis, risk modeling, data science, and software engineering. After completing his undergraduate studies, he worked for around five years in the software industry as a Software Engineer. He joined The University of Memphis, in Spring 2017. Besides, he conducted hands-on lab topics related to web security, network analysis, and malware analysis for graduate and undergraduate students. His research interests include malware analysis, reverse engineering, secure coding, vulnerability analysis, web security, data mining, big data, natural language processing, and AI. He also serves as a technical cyber-security reviewer for various journals and conferences, with several works published in different conferences and journals.

DIPANKAR DASGUPTA (Fellow, IEEE) joined The University of Memphis as an Assistant Professor, in 1997, and became a Full Professor, in 2004. He has been an Advisory Board Member with the Geospatial Data Center (GDC), Massachusetts Institute of Technology, since 2010. He has published a number of books and edited volumes, including *Advances in User Authentication* (2017), *Immunological Computation: Theory and Applications* (2008), *Artificial Immune Systems* (1999), and another book on genetic algorithms (1996). He has published more than 300 research papers with 20000 citations of his work. His research interests include the computational intelligence (including AI and machine learning) for the design and development of intelligent solutions. He is one of the founding father of the field of artificial immune systems, making major contributions in developing tools for digital immunity and survivable systems. He was a recipient of the 2011–2012 Willard R. Sparks Eminent Faculty Award, the highest distinction and most prestigious honor given to a Faculty Member by The University of Memphis. He was also a recipient of the 2014 ACM SIGEVO Impact Award and an ACM Distinguished Speaker, from 2015 to 2020. He currently holds the William Hill Professorship at The University of Memphis.