2023 IEEE International Conference on Data Mining Workshops (ICDMW)
DOI: 10.1109/ICDMW60847.2023.00171

Automatic Detection and Analysis towards Malicious Behavior in IoT Malware

1st Sen Li
School of Future Technology
Tianjin University
Tianjin, China
senli@tju.edu.cn

2nd Mengmeng GE
School of Computer Science and Engineering
Nanyang Technological University
Singapore
mengmeng.ge@ntu.edu.sg

3rd Ruitao FENG
School of Computing and Information Systems
Singapore Management University
Singapore
rtfeng@smu.edu.sg

4th Xiaohong Li
College of Intelligence and Computing
Tianjin University
Tianjin, China
xiaohongli@tju.edu.cn

5th Kwok Yan Lam
School of Computer Science and Engineering
Nanyang Technological University
Singapore
kwokyan.lam@ntu.edu.sg

***Abstract*—Our society is rapidly moving towards the digital age, which has led to a sharp increase in IoT networks and devices. This growth requires more network security professionals, who are focused on protecting IoT systems. One crucial task is to analyze malicious software to gain a deeper understanding of its functionalities and response methods. However, malware analysis is a complex process that requires the use of various analysis tools, including advanced reverse engineering techniques. For beginners, parsing complex binary data can be particularly challenging as they may be strange with these tools and the basic principles of analysis. Even for experienced analysts, understanding reverse engineering binary files and assembly lists is daunting.**

Facing these challenges, we propose a two-fold solution. Firstly, we create a detailed list of analysis tools and construct a malware analysis framework aimed at simplifying the analysis process. The framework will list the key data points that need to be addressed in the analysis, providing analysts with the tools and information needed for effective malware analysis. Secondly, we will demonstrate that advanced analysis techniques by providing analysis scripts which automate the reverse engineering process in malware analysis. To evaluate the accuracy of our behavior classification system, we will use our framework and analysis scripts to analyze known malware samples. Then, we will compare the accuracy of script-based analysis results and evaluate their ability to identify malicious software behavior. Our research results indicate that by following our framework and using our scripts, we can detect over 80% critical malware behaviors in known samples, which highlights the potential of simplifying the process of malware analysis, making it easier to learn and implement.

***Index Terms*—IoT malware, Automatic analysis, Malicious behavior analysis**

I. INTRODUCTION

The Internet of Things (IoT) has rapidly changed the way we live and work in society. Be it in healthcare with blood sugar monitoring devices or with smart bulbs to help our homes be more efficient [1], IoT devices have changed how we interact with the digital world and have also become integral components of our daily lives and critical infrastructure. And it brings an estimated 100 billion devices to be online by 2025 [2], IoT Networks and devices are going to become more and more commonplace.

Unfortunately, the rapid popularity of IoT devices has not escaped the attention of cybercriminals and malicious actors. Although these devices provide us with convenience, efficiency, and innovation, they also introduce new and important security challenges. Due to the diversity of IoT devices, the motivations behind malware targeting these devices have become diverse, such as financial interests, political agendas, or cyber espionage activities. This makes IoT devices more likely to become a key target for criminals and malicious actors. Unlike ordinary devices, IoT devices typically have a large number of low-quality code and architecture defects [3], which makes them more likely to stimulate the interest of network criminals and malware writers. In addition, many IoT devices are still using outdated firmware and different levels of security measures, making them more susceptible to malware attacks.

For instance, the Mirai malware is an iconic example in the IoT field, quickly gaining widespread attention as one of the earliest malware types targeting IoT devices. Mirai rapidly scans devices on the network that still use default login credentials, enabling hackers to simultaneously control over 400,000 devices for malicious purposes [4]. This allows hackers to easily control the computing power of these devices and launch highly destructive attacks. Once Mirai's source code was publicly released, it would lead to the emergence of more malware. One of the most prominent attacks was directed at the DNS service provider Dyn, resulting in the temporary shutdown of hundreds of well-known websites, including Twitter, Netflix, Reddit, and Github, for several hours [4]. The rise and wide dissemination of Mirai have underscored the security issues associated with IoT devices. It reveals that many IoT devices are not designed and deployed with basic security measures in mind, such as changing default

credentials or regularly updating firmware. This malware has also sparked broader discussions on IoT device security and prompted researchers and industry practitioners to explore new ways to protect the IoT ecosystem from threats.

IoT malware has increasingly become more prevalent. With an estimated 700% increase in attacks during the pandemic [5], it becomes evident that the diverse range and rapid growth of malware within the Internet of Things (IoT) presents significant challenges to security researchers and practitioners. Consequently, security experts and researchers must develop more effective methods for detecting, identifying, preventing, and mitigating these threats. Concurrently, detecting and pinpointing the crucial malicious behaviors within IoT malware are becoming increasingly critical.

However, in the struggle against malware, the process required to analyze the malware is often unlike the targets in the tasks of detection [6]–[9], requiring not only a detailed understanding of its operation [10] during security analysis but also identifying its malicious behavior [11] and functionality [12] through various methods. This complexity makes malware analysis cumbersome and requires a significant amount of manpower and time resources to be invested. Secondly, in the face of constantly evolving malware, attackers use increasingly complex emerging technologies to confuse and conceal their true intentions and activities, which further increases the difficulty for security personnel to analyze malware behavior.

To tackle this challenge, our work aims to establish an analysis framework for IoT malware and develop a malicious behavior identification system. This system will empower researchers to swiftly discern the purpose and detailed execution of specific malware behaviors. In doing so, it will mitigate the impact of IoT malware and automate the process of identifying malicious behavior in IoT malware, ultimately enhancing analysis efficiency.

In this paper, we conduct a manual analysis of a substantial volume of IoT malware samples and generate malware reports to gain insights into the malware analysis procedure. Building on this, we establish a malware analysis framework that connects malicious behavior with its intricate implementation.

Finally, we develop an automatic malicious behavior recognition system to quickly identify fine-grained malicious behaviors [13] of malware and point out the implementation of each fine-grained malicious behavior in malware.

In summary, we make the following main contributions:

- We establish a malware analysis framework. This framework can be used by researchers to conduct malware analysis according to its steps. It will organize the key tools and technologies required for malware analysis and formulate a workflow that includes all necessary steps, tools, and technologies for systematic malware analysis. Subsequently, we will generate detailed analysis reports to provide comprehensive information on malware samples.
- We develop a system for classifying malware behavior in the IoT. This system enables us to identify and classify malicious behavior based on the code implementation of malware samples. It will greatly accelerate our ability to understand the intention and threat level of fine-grained malicious behavior exhibited by malware.
- We establish an automated analysis system for IoT malware. We build the automated analysis system based on Ghidra's API. When analyzing malware, using this system can quickly and accurately identify malicious behavior, thereby improving the efficiency and speed of IoT malware analysis.

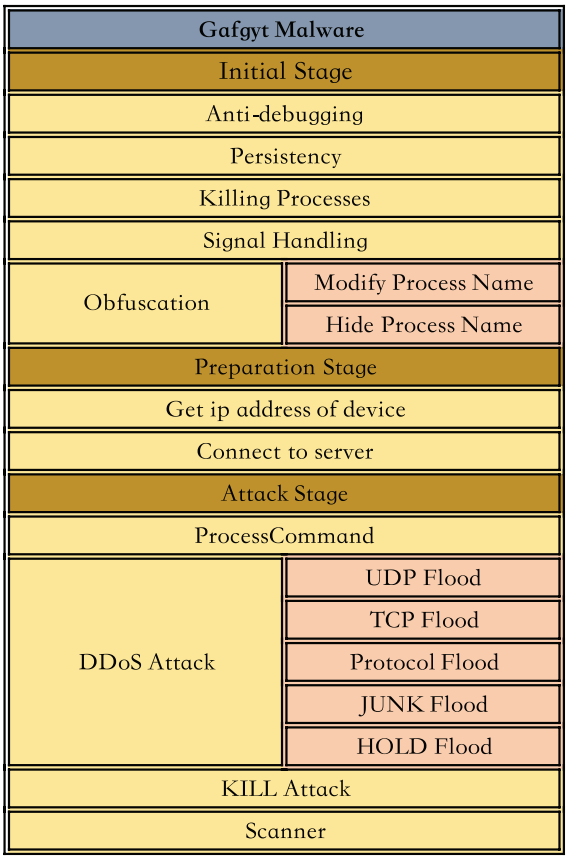

Fig. 1. Gafgyt's Malicious behavior

II. APPROACH

In this section, we introduce our workflow and implementation methods, which primarily consist of the following three aspects: (1) Behavior Analysis and Behavior Category Mapping: We conduct malware analysis to identify key malicious behaviors and categorize them into three distinct categories. (2) Analysis Framework Construction: We compile a detailed workflow and analysis framework, incorporating essential tools, techniques, and specific methods required during the analysis process. This framework enables systematic and organized malware analysis. (3) Establishment of an Automated Analysis System: Based on the knowledge acquired above, we develop an automated analysis system using Ghidra. This sys-

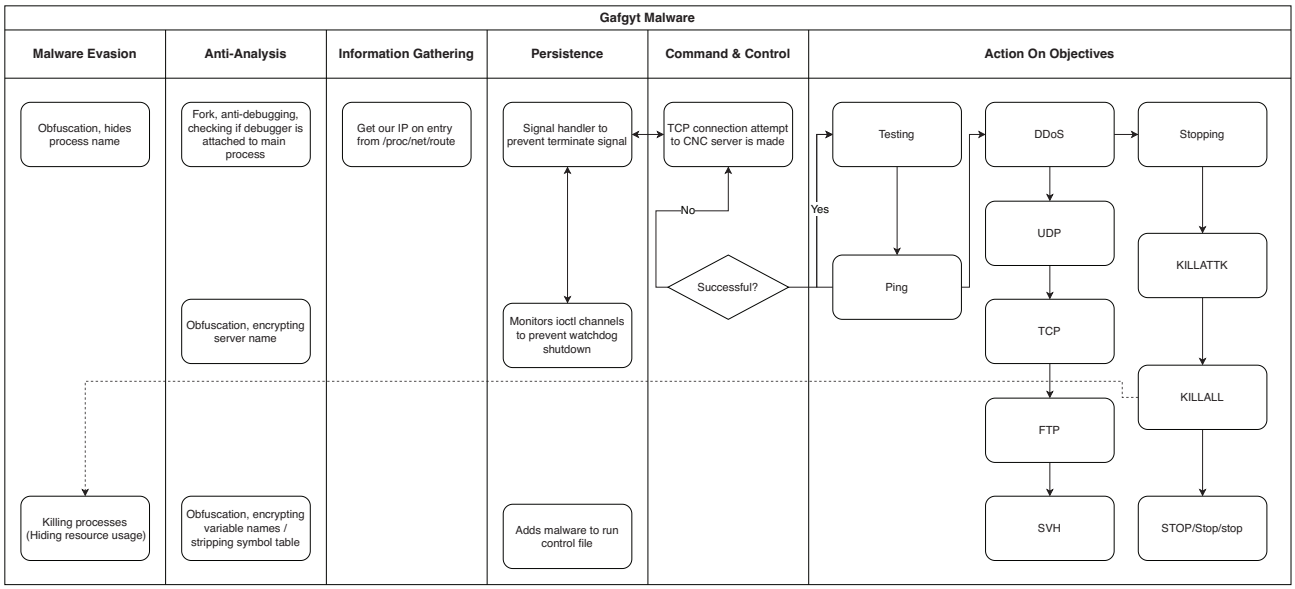

Fig. 2. Gafgyt's Malicious Behavior Category Mapping

tem facilitates rapid malware analysis, detection, identification, and localization.

A. Behavior Analysis and Behavior Category Mapping

We obtain malware and conduct critical malicious behavior analysis on the malware, then classify and map the observed critical malicious behaviors.

We first obtain over 100 IoT malware samples from various honeypots, which can be easily obtained from malware sample websites. In order to ensure the persuasiveness of key behaviors, we refer to the classification results of VirusTotal. Secondly, in order to cover all key malicious behaviors of malware as much as possible, we conduct a comprehensive analysis of malware based on the malware attack chain and combine dynamic and static analysis techniques. Through the above dual analysis, we obtain the key behaviors and implementations of each malware sample. By analyzing multiple malware samples, we can observe common patterns and techniques that hackers use to make malware more effective. Next, based on the results of these analyses, we classify the key malicious behaviors based on their functions and combine them with the MITRE ATT&CK matrix, and provide detailed definitions for each category after division. By categorizing malicious behaviors, security professionals can quickly understand the functional intent and purpose of key malicious behaviors. Therefore, a standard category definition is still necessary.

Standard categorization can help capture program structural information, communication, and control flow. This will help provide more comprehensive information on malware behavior [14]. As shown in Figure 1, Based on the attack chain of the malware family, we have listed all observed behaviors. Based on the functionality of key malicious behaviors, we divide them into three categories: initial stage, preparation stage, and attack stage. Then we map all observed malicious behaviors to these three categories. In order to link malware behavior with specific implementations, we have listed detailed information in each behavior, mainly including header files, system calls, and parameters. Next, in order to provide a clearer description of the classification and the specific implementation of key malicious behaviors under each category, as shown in Figure 2, we use the well-known Gafgyt family in IoT malware as an example for behavior detection, recognition, and behavior category mapping.

1) Initial Stage: The initialization phase is to establish a foothold in the device, ensure its stable operation on the target system, and execute subsequent malicious behaviors, which may include data collection, obfuscation, anti-debugging, etc. The initialization phase is the first step in implementing these behaviors [15]. The following are the key malicious behaviors observed during the initialization phase of the analysis process and their implementation.

Anti-debugging. During the initial stage of malware, attackers typically attempt to use fork anti-debugging techniques to evade dynamic analysis [16]. Gafgyt malware often uses forks and exit system calls to attempt to interfere with and prevent the debugger from attaching. One implementation method is to use fork() twice to create another child process within one child process. Then, by using the waitpid() function, once a second child process is created, the main process quickly shuts itself down. The purpose of this strategy is to prevent the debugger from tracking the rapid shutdown of the main process, thereby preventing it from detaching itself and stopping the execution of the second child process, which will continue to execute the rest of the malware.

Persistence. Running control persistence is sometimes used

by Gafgyt malware to ensure its existence in the system. Run control persistence is a technology adopted by Gafgyt malware, aimed at ensuring that the malware will automatically start every time an infected system restarts, thereby maintaining its continuous activity [17]. By modifying the system's startup configuration, Gafgyt malware ensures that regardless of whether the security tool terminates or removes the malware process, it can start the malware process when the system restarts. This allows the malware to remain present on the infected system and continue to execute its malicious activities. When Gafgyt malware infects the system, it will modify the system's startup configuration to add a new malware startup entry. In Linux systems, especially on IoT devices, this is typically achieved by editing the rc.local or rc.conf files. In this way, malware ensures that it is activated every time when the system starts to maintain its existence.

Obfuscation. The Gafgyt malware often uses some technical means to confuse its process name to increase its detection difficulty [18]. Some security tools may identify and intercept malicious processes based on their names. By modifying the process name, malware can effectively evade the detection of these security tools, thereby continuously lurking in the system. In addition, this behavior also helps malware disguise itself as a legitimate process, making it more difficult for system administrators to identify it as an abnormal situation and take necessary clearance measures. This confusion technique is typically achieved by utilizing the PR_SET_NAME option of a system call called prctl.

Signal Handling. The Gafgyt malware often utilizes signal processing to control and manipulate its process behavior. For example, using the SIGPIPE signal can prevent malware from self-closing when the CNC server closes its socket. This signal-processing technology is typically used to ensure that malware does not automatically shut down when the connection is disconnected. This process can be achieved by calling the Signal system call.

Killing Processes. The malware sometimes chooses to kill other processes running in the system. Similarly, with the help of a system call called kill, the malware can release system resources to support its malicious behavior. This helps to provide the malware with sufficient resources to execute its malicious activities, while avoiding detection by using this method to terminate security tools or monitor processes that may discover their existence and alert system administrators.

2) Preparation Stage: The preparation phase is a crucial part of malware, providing attackers with the necessary resources and information to ensure the success of the attack. At this stage, attackers usually invest a lot of time and effort to ensure the effectiveness and concealment of the attack. The following are the key malicious behaviors observed during the analysis process that belong to the preparation stage and their implementation.

Get IP Address of Device. After the initial stage, The Gafgyt malware will take further measures to prepare for deeper operations. It will obtain its local IP address, which is necessary for communication with the CNC server that attempts to connect later [19]. At this point, the malware will create a socket and send domain name system (DNS) queries for the infected device from that socket. Through this socket, malware can use the getsockname() function to query and retrieve the address information of the socket, including the IP address.

Connect to Server. In order to obtain further instructions from the attacker, the malware will need to establish a connection with the CNC server. This will require creating a SOCK_STREAM type socket using the TCP protocol. A "stream socket" or "SOCK_STREAM" socket is a type of socket that provides a reliable, bidirectional, byte stream communication channel that connects two endpoints. This is necessary to ensure accurate communication with the CNC server and the ability to send and receive instructions. After creating this socket, use the connect() system call for socket communication to communicate with the CNC server.

3) Attack Stage: The attack phase is the main part of malware, and after establishing contact with the CNC server, the malware will receive command strings from the CNC server. When receiving these command strings, malware will use the receiving system call associated with the socket created when establishing contact with the CNC server. After receiving the command, any string will be processed and compared with the command string. If the commands match, the malware will execute its expected attack behavior. The observed attack behaviors through analysis include: Distributed Denial of Service (DDoS) attacks, such as UDP Flood Attacks, KILL Attacks, and Scanners.

DDoS Attack. The specific implementation process for DDoS Attacks is as follows:

1) Receive the address of the attack target from the attacker.
2) Tokenize the target addresses (In the case of multiple targets).
3) Receive the following parameters: destination, port, time, subnet mask, packet size, and time polling interval.
4) Use sockets and the netinet library to obtain the IP address of the target.
5) Create a socket.
6) If spoofing is set, create a forged packet IP address.
7) Enter a while loop and repeatedly send data packets for a certain amount of time.

In other words, DDoS attacks involve processing target addresses received from attackers, matching them with their IP addresses, and then repeatedly sending packets in an attempt to overwhelm the target server.

Kill Attack. The specific implementation process of Kill Attack is as follows: After receiving the attack command, the malware will traverse the active processes and terminate them using the kill system call. Similar to the initial stage, killing a process may be done to free up resources or attempt to evade potential detection systems. It may also be to shut down any operations that are running on IoT devices.

Scanner. The specific implementation process of Scanner is as follows: When running Scanner to spread malware, the malware first obtains information about the maximum number of processes it can run by using getdtablesize(). Next, it will use file descriptors as much as possible and use custom remote state data structures to track the status of these file descriptors. Each file descriptor attempts to establish a connection with a random IP address and checks if they are using telnet. At the same time, monitor each file descriptor to see if they are connecting to a new device, successfully connecting, or closing due to a connection interruption. If successfully connected, the malware will attempt to forcibly enter the new device and grant a shell from which it can download and spread the malware. Next, the malware uses the file descriptor methods in the select libraries of various systems to search for login and password responses. Finally, once the shell is obtained, the malware will use the send() method to send a string command, download and execute the malware from the CNC server.

B. Analysis Framework Construction

When conducting malware behavior analysis, our main goal is to summarize the key malicious behaviors and specific implementation methods [14].

1) Analysis Process: To achieve this goal, we utilize both static and dynamic analysis methods [10] to make the analysis of malware more comprehensive. For a malware to be analyzed, the specific analysis process is as follows:

Analysis based on VirusTotal. We first use an antivirus program to check if malware has been identified. These antivirus tools rely on signature-based detection and heuristic-based detection [20]. Signature-based detection relies on viewing collected malicious file databases [21], which typically use hash algorithms such as MD5, SHA1, or SHA256 to generate unique hash values for files, such as SHA256. The heuristic-based detection method relies on behavior and pattern matching analysis to identify suspicious files [22]. Usually, each antivirus program uses different signature and heuristic detection methods. In order to obtain the most comprehensive coverage, we will check multiple antivirus programs to see if there is any available information. A useful tool is VirusTotal, which is a malware online search website that aggregates information on many antivirus products and online scanning engines. VirusTotal can match malware samples with various antivirus programs in the database.

Analysis based on Ghidra. In order to reverse engineer ELF files, we use a tool called Ghidra. Ghidra's working principle is to decompose binary code into smaller fragments and analyze each fragment independently [23]. When we load a binary file into Ghidra, it automatically recognizes the instruction set and file format. Then, it decomposes the binary files into assembly code for our research. In the assembly process starting from Ghidra, we first analyze the entry points of malware programs. By tracking entry points, we can easily find the main functions and understand the operations of malware programmed by the creator of the malware.

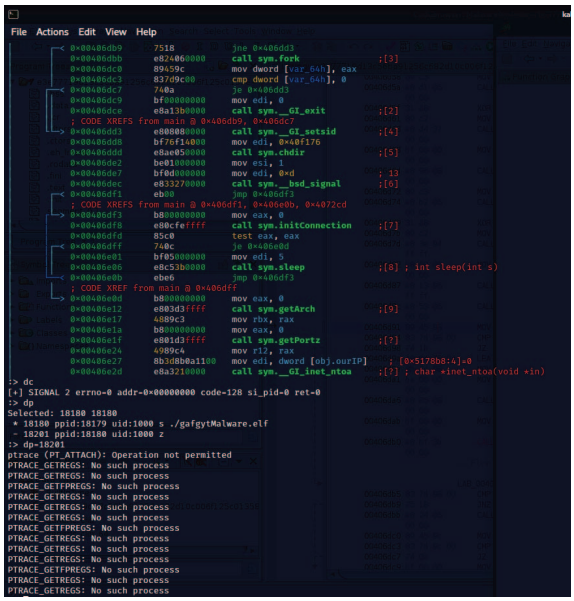

Fig. 3. Dynamic Analysis with Radare2

By analyzing the assembly code, we can infer function calls and values to understand the functionality of malware. A very useful tool in Ghidra is its built-in decompiler, which can convert assembly code into higher-level code that is easier to understand. This helps us better understand the execution process of the program. Based on Ghidra, we are able to record the key malicious behaviors of various malware samples, including those intentionally confused or analyzed by malware authors.

Advanced Analysis based on Ghidra. However, creators of malware utilize reverse analysis techniques in general, which increases the difficulty for analysts to analyze malware.

In our research, we observe that some common anti-reverse analysis techniques in IoT malware include: Code obfuscation, Anti-debugging, and Packers. etc.

Fortunately, Ghidra is a very powerful decompiler that can counter many of these reverse analysis techniques. For example, it has built-in code analysis functions that can automatically perform feature analysis such as data flow analysis, control flow analysis, and function recognition. We can assist us in our analysis work by generating function IDs online. Based on the architecture information obtained earlier, we can generate a function ID and then use Ghidra's function ID database file to analyze the signature of the function. Even when the symbol table has been stripped, Ghidra's function ID tool is still able to effectively identify common system calls and functions, greatly aiding our analysis process.

Analysis based on Radare2. For functions that cannot be identified through Ghidra's analysis tools, we can use debugging tools such as Radare2 or GDB for advanced dynamic analysis as shown in Figure 3. Debugging allows us to execute instructions one by one at a convenient time

and selectively execute specific functions without having to run the entire program [24]. By using a debugger, we can jump inside a single function to better understand its code functionality and the specific implementation of key malicious behaviors.

2) Malware Analysis Framework: By utilizing various malware analysis techniques, we can record multiple key behaviors in malware samples to help us comprehensively and deeply understand malware. As shown in Figure 4, To help us achieve this goal, we have constructed a malware analysis framework to standardize and simplify the analysis process of malware samples. This framework follows the malware analysis techniques we mentioned earlier.

We start with basic static analysis and use VirusTotal to record detection information, detailed information about files, and relationships between files in our framework. In addition to basic static analysis, VirusTotal also provides built-in sandboxes for dynamic analysis, where they place files into internal system sandboxes such as Linux Zenbox to run. Based on all analysis results, we record all suspicious behaviors to assist us in further reverse engineering of malware.

After completing the basic analysis, we begin the process of reverse engineering. Based on the information in the basic analysis, we may need to take additional steps. For example, if the symbol table has been stripped, we need to generate and download a function signature to assist in identifying the function ID [25]. We will follow the above analysis framework for all malware analysis, record the key malicious behaviors analyzed in the malware samples, and track their specific implementation methods.

C. Establishment of an Automated Analysis System

Based on the results of the above analysis, we begin to build an automated IoT malware behavior analysis system. This system will automatically analyze malware samples, classify key malicious behaviors discovered, record their implementation methods and specific locations, and output information to easily readable text files. This greatly simplifies the malware analysis process for security professionals, and can help them quickly obtain suspicious malicious behavior categories of malware, enabling them to quickly take response measures.

1) Malware Analysis Script: Based on the comprehensive consideration of various aspects of the malware automatic analysis system, we have decided to use Ghidra's script function for development. We utilize the scripting language and API provided by the Ghidra reverse engineering framework to automatically execute and customize various tasks related to analyzing binary files. Through Ghidra scripting, we can automate and write scripting tasks using Java or Python. The script interface allows us to automate tasks such as data extraction, symbol renaming, function recognition, and more. For our project, we have decided to use Java scripting because Java is Ghidra's native language. This means that Java will

IoT Malware Analysis Framework

Malware Source: http://98.159.98.37/x86

Input the malicious file into VirusTotal:

VirusTotal Result	
Detection	Score: 44/63 Identified Malware Families: Gafgyt, Mirai
Details	Basic Properties MD5: 75c207f575c6e8948e2a431496e02e46 SHA-1: 7f4542978b172a5a0dcc3657545cadf24575f4e2 SHA-256: 73f43564bfa83fea599274bfc99769362b5070f0d46bcb34da75180e9441be89 Magic: ELF 64-bit LSB executable, x86-64, version 1 (SYSV), statically linked, not stripped File size: 112.13kb ELF Info Class: ELF64 Data: 2's complement, little endian OS ABI: UNIX-System V Required Architecture: Advanced Micro Devices x86-64
Relations	Contacted IP addresses: 98.159.98.37
Behavior	Observed Behavior Data Obfuscation: Sample contains strings that are user agent strings indicative of HTTP manipulation

Fig. 4. IoT Malware Analysis Framework

provide us with all the functionality of the Ghidra API and allow us to leverage many existing Java libraries.

```
public void run() throws Exception {
    ...
    // obtain function using ghidra's API
    Function func = getFirstFunction();
    // Use a while loop to cycle through functions
    while (func != null && !monitor.isCancelled()) {
        ...
        // Behaviour analysis methods
        ...
        func = getFunctionAfter(func);
    }
}
```

Listing 1. Function Iteration

2) Implementation of an Automated Analysis Script: In our project, the goal of analyzing IoT malware is to summarize the key malicious behaviors and specific implementation methods of IoT malware. Therefore, We want to automatically analyze the key malicious behaviors and their specific implementations of malware through scripts, we must be clear about the specific implementation methods for manual analysis, including:

Identify key functions and code snippets: Analyze disassembly to identify key functions and code snippets that may provide insights into the functionality and behavior of executable files. Analyze control flow: Use Ghidra's flow chart feature to analyze the control flow of executable files and identify the main execution paths. Analyze data structure: Use Ghidra's data structure view to analyze the data structures used in executable files, such as arrays, structures, and linked lists. This helps determine how the program stores and processes data. View the strings used: Analyze the strings used in the assembly list, such as file names, comments, or conditional checks, which may provide clues about the intended use of the malware creator.

Based on this, we can use Ghidra's scripting function to

simulate the above analysis method, thereby achieving automatic analysis of malicious behavior in IoT malware. Based on the execution process of malware, we can know that the script will first check whether the function contains behavior under the "initial stage" node, followed by the "preparation stage" and "attack stage".

Therefore, As shown in Listing 1, a function iterator is used to cycle through each function obtained from reverse engineering the malware sample. This enables us to search through the entire binary effectively. For each behavior observed in our earlier analysis, we create a method to search through the function and detect individual behavior based on the implementations observed and demonstrated in the flow chart. Within each method, we return an object array that contains information on whether a particular behavior is detected, the Assembly implementation of the behavior, and most importantly the location where the behavior is detected.

Anti-debugging. Based on our previous analysis of fork anti debugging techniques, malware typically uses process forking and closing to avoid debugger detection. Due to the fact that process forking and closing typically follow a certain pattern, we can add a method to the script to detect the presence of this pattern in the function. As shown in Listing 2, Our script can utilize Ghidra's instruction API to iterate and detect function calls, determining when to make function calls by checking the instruction stream type. The script will detect whether the program calls the fork, waitpid, fork, and exit system calls in order, by comparing them with a specified function array to track each system call and match it with the previously mentioned pattern.

```
private Object[] antiDebugCheck(Function tgtFunc){
    ...
    // obtain listing from ghidra program
    Listing plist = currentProgram.getListing();
    ...
    // we set a target sequence that would be
   indicative of the fork anti-debugging behaviour
    String[] target = {"fork", "waitpid", "exit", "
   fork", "exit"};
    ...
    // We use an instructino iterator to iterate
   through all instructions in the function
    InstructionIterator iter = plist.
   getInstructions(tgtFunc.getBody(), true);
    while (iter.hasNext() && !monitor.isCancelled()
   ) {
        Instruction ins = iter.next();
        Address addr = ins.getAddress();
        // Check if the instructions match our
   target sequences
        if (ins.getFlowType().isCall()) {
       if(getFunctionAt(ins.getFlows()[0]).getName
   ().equals(target[count])) {
            ...
       }
        }
    }
}
```

Listing 2. Anti-Debugging Detection

Obfuscation. For process name confusion, we know that it is implemented using the prctl call with the PR SET NAME option set. As shown in Listing 3, to detect this behavior, we can use Ghidra's cross reference script to search for all references to the prctl function call.

After finding the prctl function call, we can use Ghidra's decompiler interface to identify all parameters in the prctl function call and check if the PR_SET_NAME option (value 15) has been set. If set, the detection location and its implementation method will be written to the report file.

```
DecompInterface ifc = new DecompInterface();
ifc.openProgram(currentProgram);
TaskMonitor taskmonitor = new ConsoleTaskMonitor();
DecompileResults res = ifc.decompileFunction(
   tgtFunc,0,taskmonitor);
HighFunction hfunc = res.getHighFunction();
Iterator<PcodeOpAST> opiter = hfunc.getPcodeOps();
while (opiter.hasNext()) {
    PcodeOp op = opiter.next();
    if (op.getOpcode() == PcodeOp.CALL) {
        if(getFunctionAt(op.getInput(0).getAddress
   ()).getName().equals("prctl") && op.getInput(1).
   getOffset() == 15) {
            println("prctl name change found");
        }
    }
}
```

Listing 3. Obfuscatio Detection

Killing Processes and Signal Handling. Signal processing and process termination are also detected and recorded in report files by tracking cross references and function parameters.

Persistence: For persistence, we know that it is achieved by adding content to the runtime control file. Therefore, by Ghidra's search function, we can search for run control strings in the program, which may indicate an attempt to manipulate the run control file. If such a string is detected, it will return its position in memory. Afterwards, we can check for any references to the file and whether to try using the fopen/fclose function call to edit the file. As shown in Listing 4, we can use Ghidra's Symbol interface to check if fopen is used on the runtime control file. Then, the implementation of the entire attempt can be accurately determined between fopen and fclose function calls. Afterwards, we will record the detected behavior, implementation, and location in the report file.

```
Reference refs[] = instr.getReferencesFrom();
for (int i = 0; i < refs.length; i++) {
    if (refs[i].getReferenceType().isCall()) {
        Address tgt = refs[i].getToAddress();
        Symbol sym = getSymbolAt(tgt);
        String sname = sym.getName();
        if (sname.equals("fopen")) {
           count += 1;
           startCount = true;
        }
        if (sname.equals("fclose")) {
           count -= 1;
        }
    }
}
if (startCount && count == 0) {
   Address endAddr = instr.getAddress();
   AddressSet addrSet = new AddressSet();
   addrSet.addRange(startAddr, endAddr);
   objArray[2] = addrSet;
   return objArray;
}
```

Listing 4. Persistence Detection

Get IP Address of Device. In the preparation phase, according to our analysis, EasySoft usually sends DNS requests, and then calls the getsockname() function to identify the IP address of the infected device. Therefore, we can use Ghidra's cross referencing function to search for getsockname() function calls. If such a function call is detected, we can conclude that the function is attempting to obtain the IP address of the device. After finding any references to getsockname, we will search for the socket() function call to determine its implementation, as based on our previous analysis and flowchart, this will be an indicator for obtaining the local IP.

Connect to Server. Similar to prctl behavior checking, we use Ghidra's decompiler interface to check the parameters passed during function calls. Then, we use Ghidra's pseudo code (also known as P-code) interface to filter out the parameters passed to the connect function call. We intentionally use P-code because it is a low-level, platform-independent representation of machine code used by the Ghidra disassembler and decompiler [26]. Given that IoT devices cover a wide range of platforms, this makes them particularly useful in analyzing IoT malware samples. Since we know that any reliable bidirectional communication attempt will require a flow socket (integer value 1) and a TCP protocol connection (integer value 0), we can filter based on the parameters of the function call and only select cases where the appropriate socket type and protocol are used. From here, we can add it to the report to detect CNC server connection attempt behavior.

```
// The address which we want to check if instruction is in a loop
Address addr = op.getSeqnum().getTarget();
//Use basic block model api to get basic blocks for analysis
BasicBlockModel blockModel = new BasicBlockModel(currentProgram);
ConsoleTaskMonitor monitor = new ConsoleTaskMonitor();
AddressSetView funcAddrSet = tgtFunc.getBody();
println("Address to string: " + addr.toString());
// Get the blocks of target address
CodeBlock[] block2 = blockModel.getCodeBlocksContaining(addr,monitor);
Address startAddr = block2[0].getFirstStartAddress();
// We use a dfs algorithm to check if there are any loops within the flow graph.
HashSet<CodeBlock> visited = new HashSet<>();
Stack<CodeBlock> stack = new Stack<>();
stack.push(block2[0]);
boolean isWhileLoop = false;
while (!stack.isEmpty()) {
    CodeBlock currBlock = stack.pop();
    visited.add(currBlock);
    CodeBlockReferenceIterator destinations = currBlock.getDestinations(monitor);
    while (destinations.hasNext()) {
   CodeBlockReference destblockRef = destinations.next();
   CodeBlock destblock = destblockRef.getDestinationBlock();
   if (visited.contains(destblock)) {
       if (destblock.getFirstStartAddress() == startAddr)
           isWhileLoop = true;
            break;
       }
       if (funcAddrSet.contains(destblockRef.getDestinationAddress())) {
           stack.add(destblock);
       }
    }
}
```

Listing 5. While Loop Check

DDoS Attack. After connecting to the CNC server, malware will need to receive instructions from the CNC server. Therefore, we can check whether the function made a recv() function call to parse the information received from the socket. Due to the fact that Gafgyt malware is known as botnet malware, its main malicious activity is mostly concentrated in DDoS attacks. After conducting malware analysis, we determine that during DDoS attacks, the malware would repeatedly create a network connection to the victim's server and then repeatedly send network packets to it, attempting to flood its server. Therefore, as shown in Listing 5, we can use the connect() function call to detect malicious DDoS behavior. By using function blocks, we can check whether the connect() function call is within the loop, indicating repeated sending of network packets. If detected, then determine the implementation method, check the time of socket creation, and wait until the connect() function call is made. Then insert the information into the report log.

3) Utilizing the Malware Analysis Script: To use this script, users first need to download and save the script. Simply input the script into Ghidra's script folder and open the Script Manager. Afterward, users only need to open the script when they want to analyze the binary file, first run Ghidra's automatic analysis, and then run the corresponding automatic detection script. The script will further analyze based on Ghidra's automatic analysis results, obtain relevant key malicious behaviors and their implementation methods, as well as location information, and ultimately form a complete malicious behavior analysis report for users to read. Based on this report, users can quickly understand detailed information about key malicious behaviors of malware.

III. EVALUATION

In this section, we demonstrate the power of automated analysis script.

A. Experiment Setup

In our experiment, we first obtain new malware samples from the malware bazaar and manually analyze them according to our developed analysis framework. Manual analysis can delve into each malware sample, identify malicious behavior within it, and record them. As shown in Figure 5, these manual analysis results serve as our benchmark for subsequent automated analysis comparisons. Then, we use the automated script we built to analyze new malware samples obtained from the malware market and generate a report file. Then extract and record the detected behavior from the generated report file.

At the same time, in order to evaluate the differences between the results of our automated analysis script and existing solutions, we compare the analysis results of our

Malware Analysis Reports Comparison		
MalwareHash	Number of Malicious Behaviors Detected	
	Ghidra Script	Manual Analysis
179387e44acac...	13	14
2b85778959c5f...	8	10
2d08cabf21996...	8	9
39dd070ea397d...	9	10
3bb0c13d6fc45...	16	18
489963f24cd60...	8	10
512fc8546c9e7...	8	10
5805fb0757ccb...	15	17
73f43564bfa83...	25	28
8371943e6c6f1...	14	17
b379d138e0bfa...	10	12
b9eeaa5c661b5...	13	16
c29859c0acdf6...	16	18
c5702da4f46e6...	25	28
d08b9242657a1...	8	10
e3e77719d13c5...	19	29
eb101b9e7a0cf...	13	14
eb3094e350e6a...	15	16
f62321904855f...	10	12
fcd71773cb0f4...	8	9
Sum	261	307

Fig. 5. Analysis Results

constructed automated analysis script with the more mature existing Sandbox. In this experiment, we decided to use Joe Sandbox to analyze malware samples to identify their behavior, as it provides the best behavioral feedback among the commonly used sandboxes in VirusTotal and the VirusTotal community.

B. Evaluation Indicators

Our main evaluation metric is based on the number of key malicious behaviors identified from malware analysis. In the process of malware analysis, we pay special attention to the attack behavior and malicious functions of malware, including but not limited to establishing connections with command and control servers, executing distributed denial of service (DDoS) attacks, bypassing analysis and detection mechanisms, obtaining system permissions, and so on. These key malicious behaviors are crucial for evaluating the effectiveness and accuracy of malware analysis methods. We measure the performance of automated analysis methods by analyzing the number of these behaviors to help enhance network security and threat detection capabilities. Meanwhile, with the help of these evaluation indicators, we can have a more comprehensive understanding of the characteristics and behavior of malware, which can help strengthen defense measures for IoT security.

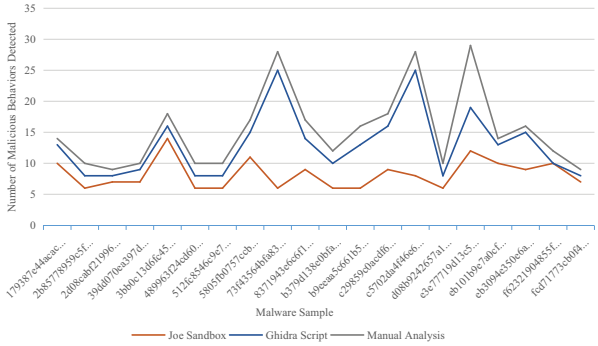

Fig. 6. Experiment Results

C. Experiment Results

From our experimental results, automated scripts can detect 85% of the malicious behavior detected in the manual analysis as shown in Figure 6. However, there are still some behaviors that we cannot detect using scripts. For example, a hard-coded string sent by a malware attacker to the server to notify them of successfully entering the malware version of the device is difficult to detect due to the highly irregular strings involved. We hope that through further improvements, we will be able to improve the accuracy of the script.

Compared to the currently available sandbox-based automated analysis methods, our automated analysis script has improved accuracy by 30%. Given the presence of anti-analysis techniques in malware and the need for proactive malicious activities during the analysis process, the accuracy of automated analysis based on sandboxes that rely on dynamic analysis may be greatly challenged, resulting in significantly lower detection rates. In addition, automated sandboxes place more emphasis on malware detection rather than reverse engineering, as they do not focus on reverse engineering. Therefore, when malicious behavior is detected, our developed scripts provide more insights into malware and provide implementation details about the detected behavior.

IV. DISCUSSION AND FUTURE WORK

A. Development of a Comprehensive Dataset

Although we have collected and analyzed a considerable dataset in our experiment, it is not exhaustive enough. Future work can focus on collecting a larger and more diverse dataset of malware samples, especially those specifically targeting IoT devices. By adding samples from different kinds of malware families, automated scripts will be able to detect more behavior in different samples. This will enable us to have a more comprehensive understanding of the behavioral characteristics of different malware samples.

B. Incorporating within a Sandbox

We can try integrating Ghidra into a sandbox environment to achieve more powerful automation of the analysis process. Based on this, we can combine basic static and dynamic analysis with Ghidra's advanced reverse engineering static

analysis to further simplify the analysis process for researchers and analysts.

C. Development of a Real-time Detection System

Our project focuses on detecting key malicious behaviors of malware and providing specific behavior implementation details. However, developing a real-time detection system that can analyze the behavior of malware running on IoT devices will be beneficial for strengthening IoT security construction. This requires further development of a feature dictionary based on malicious and benign behavior for use within IoT programs. Then, we can use a combination of system hooks and event listeners to monitor system/function calls related to malware behavior, thereby intercepting and recording calls in real time.

V. CONCLUSION

In this paper, we develop a malware analysis framework to examine IoT technology behavior. We harness Ghidra's scripting capabilities to automate the reverse engineering process within this framework, enabling rapid analysis of malware samples. Our experiments validate the effectiveness of our approach in identifying and analyzing malware behavior. It efficiently and accurately identifies malware behavior, offering valuable insights into how malware operates in IoT devices.

Overall, our project has advanced IoT malware behavior analysis. By continuously refining our methods, we can enhance the defense of IoT devices and their connected networks against persistent malware threats.

REFERENCES

[1] T. N. Gia, M. Ali, I. B. Dhaou, A. M. Rahmani, T. Westerlund, P. Liljeberg, and H. Tenhunen, "Iot-based continuous glucose monitoring system: A feasibility study," *Procedia Computer Science*, vol. 109, pp. 327–334, 2017, 8th International Conference on Ambient Systems, Networks and Technologies, ANT-2017 and the 7th International Conference on Sustainable Energy Information Technology, SEIT 2017, 16-19 May 2017, Madeira, Portugal. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1877050917310281

[2] R. T. D. B. D. Schmidt, "The world in 2025 - predictions for the next ten years," in *2015 10th International Microsystems, Packaging, Assembly and Circuits Technology Conference (IMPACT)*. IEEE, 2015.

[3] A. Wang, R. Liang, X. Liu, Y. Zhang, K. Chen, and J. Li, "An Inside Look at IoT Malware," in *Industrial IoT 2017: Industrial IoT Technologies and Applications*. Springer, Cham, 2017.

[4] C. K. G. K. A. S. J. Voas, "DDoS in the IoT: Mirai and Other Botnets," in *ICECCS*. IEEE, 2017.

[5] H. N. H. Security. Iot malware attacks rose 700% during the pandemic - help net security. [Online]. Available: https://www.helpnetsecurity.com/2021/07/20/iot-malware-attacks-rose/

[6] R. Feng, Y. Liu, and S. Lin, "A performance-sensitive malware detection system on mobile platform," in *Formal Methods and Software Engineering*, Y. Ait-Ameur and S. Qin, Eds. Cham: Springer International Publishing, 2019, pp. 493–497.

[7] R. Feng, S. Chen, X. Xie, L. Ma, G. Meng, Y. Liu, and S.-W. Lin, "MobiDroid: A Performance-Sensitive Malware Detection System on Mobile Platform," in *2019 24th International Conference on Engineering of Complex Computer Systems*, 2019.

[8] R. Feng, S. Chen, X. Xie, G. Meng, S.-W. Lin, and Y. Liu, "A Performance-Sensitive Malware Detection System Using Deep Learning on Mobile Devices," *IEEE Transactions on Information Forensics and Security*, 2020.

[9] R. Feng, J. Q. Lim, S. Chen, S.-W. Lin, and Y. Liu, "SeqMobile: An Efficient Sequence-Based Malware Detection System Using RNN on Mobile Devices," in *2020 25th International Conference on Engineering of Complex Computer Systems*, 2020.

[10] A. H. Michael Sikorski, *Practical Malware Analysis: The Hands-On Guide to Dissecting Malicious Software*. No Starch Press, 2012.

[11] Q. Qiao, R. Feng, S. Chen, F. Zhang, and X. Li, "Multi-label classification for android malware based on active learning," *IEEE Transactions on Dependable and Secure Computing (TDSC)*, 2022.

[12] G. Meng, R. Feng, G. Bai, K. Chen, and Y. Liu, "Droidecho: an in-depth dissection of malicious behaviors in android applications," *Cybersecurity*, vol. 1, no. 1, pp. 1–17, 2018.

[13] Q. Qiao, R. Feng, S. Chen, F. Zhang, and X. Li, "Multi-label classification for android malware based on active learning," *IEEE Transactions on Dependable and Secure Computing*, pp. 1–18, 2022.

[14] G. Meng, R. Feng, G. Bai, K. Chen, and L. Yang, "Droidecho: an in-depth dissection of malicious behaviors in android applications."

[15] O. Alrawi, C. Lever, K. Valakuzhy, K. Snow, F. Monrose, M. Antonakakis *et al.*, "The circle of life: A {large-scale} study of the {IoT} malware lifecycle," in *30th USENIX Security Symposium (USENIX Security 21)*, 2021, pp. 3505–3522.

[16] M. Saad and M. Taseer, "The study of the anti-debugging techniques and their mitigations," *International Journal for Electronic Crime Investigation*, vol. 6, no. 3, pp. 33–44, 2022.

[17] T. Hasan, J. Malik, I. Bibi, W. U. Khan, F. N. Al-Wesabi, K. Dev, and G. Huang, "Securing industrial internet of things against botnet attacks using hybrid deep learning approach," *IEEE Transactions on Network Science and Engineering*, vol. 10, no. 5, pp. 2952–2963, 2023.

[18] S. Torabi, M. Dib, E. Bou-Harb, C. Assi, and M. Debbabi, "A strings-based similarity analysis approach for characterizing iot malware and inferring their underlying relationships," *IEEE Networking Letters*, vol. 3, no. 3, pp. 161–165, 2021.

[19] J. Choi, A. Anwar, A. Alabduljabbar, H. Alasmary, J. Spaulding, A. Wang, S. Chen, D. Nyang, A. Awad, and D. Mohaisen, "Understanding internet of things malware by analyzing endpoints in their static artifacts," *Computer Networks*, vol. 206, p. 108768, 2022.

[20] Q.-D. Ngo, H.-T. Nguyen, V.-H. Le, and D.-H. Nguyen, "A survey of iot malware and detection methods based on static features," *ICT Express*, vol. 6, no. 4, pp. 280–286, 2020.

[21] S. Gaba, S. Nagpal, A. Aggarwal, R. Kumar, and S. Kumar, "An analysis of internet of things (iot) malwares and detection based on static and dynamic techniques," in *2022 Seventh International Conference on Parallel, Distributed and Grid Computing (PDGC)*, 2022, pp. 24–29.

[22] P. Vinod, R. Jaipur, V. Laxmi, and M. Gaur, "Survey on malware detection methods," in *Proceedings of the 3rd Hackers' Workshop on computer and internet security (IITKHACK'09)*, 2009, pp. 74–79.

[23] K. N. Kris Eage, *The Ghidra Book The Definitive Guide*. No Starch Press, 2020.

[24] Z. Liu, L. Zhang, Q. Ni, J. Chen, R. Wang, Y. Li, and Y. He, "An integrated architecture for iot malware analysis and detection," in *IoT as a Service: 4th EAI International Conference, IoTaaS 2018, Xi'an, China, November 17–18, 2018, Proceedings 4*. Springer, 2019, pp. 127–137.

[25] X. Yin, S. Liu, L. Liu, and D. Xiao, "Function recognition in stripped binary of embedded devices," *IEEE Access*, vol. 6, pp. 75 682–75 694, 2018.

[26] D. Votipka, M. N. Punzalan, S. M. Rabin, Y. Tausczik, and M. L. Mazurek, "An investigation of online reverse engineering community discussions in the context of ghidra," in *2021 IEEE European Symposium on Security and Privacy (EuroS&P)*. IEEE, 2021, pp. 1–20.