CS4110 Software Testing and Reverse Engineering

MALWARE ANALYSIS

0xcafebabe team

Pham Duy Phuc (s1750550)

University of Twente

Table of abbreviations

Abbreviation	**Full name**
RE	Reverse Engineering
AV	Antivirus
SSH	Secure Shell
SE	Social engineering
API	Application Programming Interface
OS	Operating system
C&C	Command and control
CFG	Control flow graph
OS	Operating system

Contents

1. Introduction 3
2. Basic principles of malware analysis 3
2.1. What is malware? 3
2.2. Malware analysis: 3
2.3. An example of static analysis - Metamorphic malware analysis and real-time detection 4
2.3.1. Metamorphic Malware Analysis and Real-Time Detection (MARD) 4
2.3.2. Annotated Control Flow Graph detection technique 5
2.3.3. Sliding Window of Difference and Control Flow Weight detection technique 5
2.3.4. Evaluation 5
2.4. Dynamic malware behavior analysis 6
2.4.1. Analyzing malware behavior 6
2.4.2. Experimental validation 6
3. The malware analysis arms race and its consequences 7
3.1. Malware evasive techniques 7
3.2. Malware analysis tools 7
3.3. Bare-metal Analysis-based Evasive Malware Detection 7
3.3.1. Bare-metal system overview 8
3.3.2. Evaluation 8
4. Automatic protocol RE 8
4.1. Dispatcher implementation overview 8
4.2. Evaluation 9
5. Automatically Reconstruct Android Malware Behaviors 9
5.1. CooperDroid 9
5.2. Results 10
6. References 12

1. Introduction

The malware threat landscape is continuously evolving. In this summary report I will introduce the basic concept of malware and malware analysis, the ideas of both static and dynamic malware analysis. Besides, malware evasive techniques and novel solutions will be introduced in section 3. Modern research such as automatic protocol RE and Android malware behavior analysis will be mentioned in last sections.

2. Basic principles of malware analysis

2.1.What is malware?

Software that "deliberate fulfills the harmful intent of an attacker". It motivates to create tools to detect and mitigate malicious software. There is a common signature-base AV scanners which match pre-generated signatures against files. But this approach is error-prone task and not be able to detect unknown, specially tailored malware.

Types of malware: Worm, virus, Trojan horse, spyware, bot(net), rootkit.

Infection vectors: Exploiting vulnerable services, drive-by download and SE.

2.2.Malware analysis:

Static analysis: refers to techniques that verify the actions the program performs in practice, without actually executing it. Analyst often disassemble the program to understand their behavior but it might result in ambiguous results if the binary uses self modifying code techniques (e.g., packer programs). Additionally, malware relying on outside values cannot be statically determined correctly (e.g., current system date, indirect jump instructions) Therefore, it is necessary to develop analysis techniques that are reliably analyze malicious software.

Dynamic analysis: refers to techniques that execute functions, verify the actions the program performs in practice by executing it. To monitor what functions are called is to intercept function calls (hooking). Then output the invocation to a log file or analyze input parameters or output results. There are 3 kinds of function calls: API, system calls and Windows Native API.

Implementing function hooking: by inserting to source code (if exists), binary rewriting using Detours library, debugging techniques, replacing dynamic shared libraries.

To analyze function parameter: Information flow tracking (taint sources and sinks), irect data dependencies (taint labels), address dependencies, control flow dependencies, implicit information flow.

Implementation strategies for malware analysis:

- Analysis in User-/Kernel-mode: ease to invoke functions or API calls.
- Analysis in emulator: Memory&CPU emulation (libemu, qemu etc.), full system emulation (Boshs, etc.)

- Analysis in Virtual Machine
- Network simulation: no internet but simulated network, or filtered network.

2.3. An example of static analysis - Metamorphic malware analysis and real-time detection

Metamorphism [4] is a technique that mutates the binary code using different obfuscations. It changes the opcodes with each run of the infected program and does not use any encryption or decryption (different from Polymorphism). The malware never keeps same sequence of codes in the memory.

The authors present a new framework named Metamorphic Malware Analysis and Real-Time Detection (MARD). It builds a behavioral signature and detect metamorphic malware in real-time using two techniques: ACFG (Annotated Control Flow Graph) provides a faster matching of CFGs, without compromising detection accuracy and SWOD-CFWeight (Sliding Window of Difference and Control Flow Weight) mitigates and addresses key issues in current techniques, related to the change of the frequencies of opcodes, such as the use of different compilers, compiler optimizations, operating systems and obfuscations.

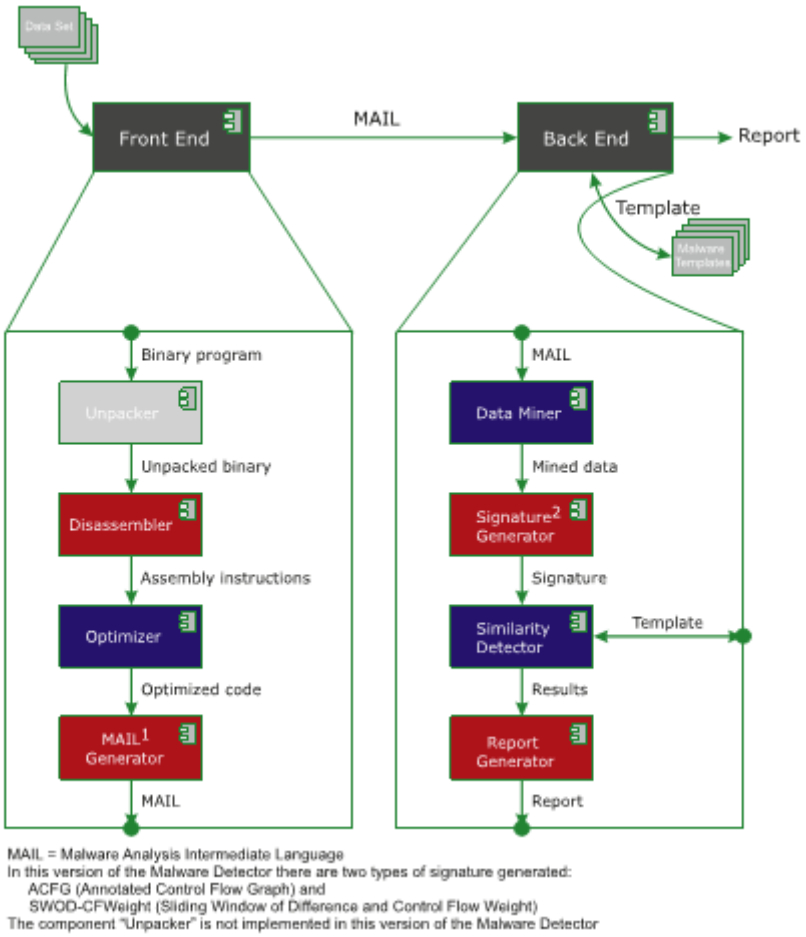


Fig 1. *Overview of MARD*

2.3.1. Metamorphic Malware Analysis and Real-Time Detection (MARD)

First a training dataset Malware Templates is built using the malware training samples. After a sample is translated to Malware Analysis Intermediate Language (MAIL) and to a behavioral signature, the Similarity Detector detects the presence of malware in the program, using the Malware Templates. All the steps as shown are completely automated. The tool automatically generates the report after all the samples are processed.

2.3.2. Annotated Control Flow Graph detection technique

A sample is first disassembled and translated to a MAIL program. The MAIL program is then annotated with patterns, then they build a CFG of the annotated MAIL program yielding the corresponding ACFG. The constructed ACFG becomes part of the signature of the program and is matched against a database of known malware samples to see if the program contains a malware or not. For detecting unknown malware, the authors build an ACFG for each function in the program is built. Then divide a program into smaller ACFGs, with one ACFG per function instead of building a large ACFG as signature. A sample which contains part of the control flow of a training malware sample, is classified as a malware, i.e. if a percentage (compared to some predefined threshold) of the number of ACFGs involved in a malware signature match with the signature of a sample then this sample is classified as a malware.

2.3.3. Sliding Window of Difference and Control Flow Weight detection technique

Shahid Alam et al propose a new opcode-based malware detection technique by transforming the assembly code to an intermediate language that makes the analysis independent of different compilers. Then they extract and analyze the control flow semantics of the program in order to mitigate the effect of obfuscations by polymorphic and metamorphic malware. The control flow semantics were applied by statistical analysis of opcode distributions to develop a set of heuristics.

2.3.4. Evaluation

Out of the 10 systems, ACFG clearly shows superior results and, unlike others is fully automatic, supports malware detection for 64 bit Windows (PE binaries) and Linux (ELF binaries) platforms and has the potential to be used as a realtime detector.

System	Analysis type	DR	FPR	Data set size Benign/Malware	Real time	Platform
MARD-ACFG	Static	98.9%	4.5%	2330/1020	✓	Win & Linux 64
API-CFG (Eskandari and Hashemi, 2012a, b)	Static	97.53%	1.97%	2140/2305	✗	Win 32
Call-Gram (Faruki et al., 2012)	Static	98.4%	2.7%	3234/3256	✗	Win 32
Code-Graph (Lee et al., 2010)	Static	91%	0%	300/100	✗	Win 32
DTA (Yin and Song, 2013)	Dynamic	100%	3%	56/42	✗	Win XP 64
Model-Checking (Song and Touili, 2012a)	Static	100%	1%	8/200	✗	Win 32
MSA (Vinod et al., 2012)	Static	91%	52%	150/1209	✗	Win 32
VSA-1 (Leder et al., 2009)	Dynamic	100%	0%	25/30	✗	Win 32
VSA-2 (Ghiasi et al., 2012)	Dynamic	98%	2.9%	385/826	✗	Win XP 64

Real-time here means the detection is fully automatic and finishes in a reasonable amount of time. The perfect results should be validated with more number of samples than tested in the paper. The values for *Opcode-Graph* are not directly mentioned in the paper. We compute these values by picking a threshold of 0.5 from the similarity score in the paper.

Fig 2. *Comparison of ACFG with the metamorphic malware detection techniques based on control and information flow analysis.*

2.4.Dynamic malware behavior analysis

This paper [3] implement an automated approach to extract malware behaviors by observing all the system functions calls in virtualized environment. A virtualized W32 machine is created through *wine* with common Windows system configuration and an emulated network infrastructure. Execution monitor, environment is controlled via *SSH*.

2.4.1. Analyzing malware behavior

Malware behavior: The authors create a set of actions (virtual operating system function call) that a malware *M* can perform.

Determination of malware actions: There is an important issue that they must differentiate between function calls called by operating system and by malware. Gerard Wagener et. al create a set of return memory addresses to characterize correctly executed functions.

Malware behavior similarities: 2 malware behaviors are compared with each other and a similarity function is evaluated. This function compares pair-wise all the action and attaches scores for matching. A malware with low similarity can be seen as unknown behavior and vice versa.

Distance between malware behaviors: The previous comparison technique only consider the similarity of function call order. In some cases, the malfunctioned activities can run concurrently or depends on the operation scheduler. Then they improve the model by rely on the frequencies of function calls as well by using Hellinger distance. Besides, a malware did not call a function does not mean it never calls these functions. So they apply the technique of statistical smoothing then show how many information is contained in a malware behavior.

Phylogenetic tree for malware: A phylogenetic tree is a branching diagram or "tree" showing the relationships among various biological species based upon similarities and differences in their genetic characteristics. The authors apply a binary tree by grouping similar nodes based on the previous similarity matrix until the matrix is empty.

2.4.2. Experimental validation

- Malwares are captured through a honeypot called Nepenthes.
- They created some anti-RE binaries to include as a behavior should be observed.
- They created obfuscated assembler code, used a debugger to see the code changes its behavior or not.
- They handled sleep(), exception division by zero (which is often used as an anti-sandbox), observed virtual operating system environment.

Using above sequences to compute the similarity, they generated a top 10 list of most common malware behavior, average similarity etc. and used AV software to explain the similarity. Then they applied phylogenetic tree to highlight 3 malware families then built 2 trees based on similarity matrix and Hellinger distance matrix.

The authors introduced the automated malware behavior analysis, then they can detect 0day malware based on low average distance similarity from real world data. It is considered that

malwares behave similarly at the beginning, it means the malware author copied parts from previous malware. They plan to extend the work with better heuristics and a differentiated analysis.

3. The malware analysis arms race and its consequences

3.1. Malware evasive techniques

Self-modifying code and packers [1]: The malware self modifies its binary and store data in memory using encoding or encryption techniques. It also can be polymorphic using different encryption keys. Countermeasures are RE the unpacking algorithm, generic detection and reconstruction of packed code (catching the original binary in process address space).

Detection of Analysis environment: Malware tries to detect analysis platforms based on: Hardware fingerprint, execution environment (eg. IsDebuggerPresent()), external applications (such as debuggers), behavioral. Mitigations are eliminating the differences between analysis environment and real environment or modifying the malware during execution to force the wanted path to be executed.

Logic bombs: A malware might only execute on a certain date or reliance on user input.

Analysis performance: The execution of malware analysis might slow down the timing in the operating system which executes malware. Countermeasures: RDTSC time-stam counter register, slowing down time in an emulated/virtual environment; terminate the analysis once a given timeout process.

3.2. Malware analysis tools

In order to deal with thousands of malwares with modern techniques, security analysts create many novel tools to ease these challenges.

- Anubis: analysis of unknown libraries based on TTAnalyze, executes the sample in a Windows OS, focus on the executed operations on behalf of the sample, and produce a report (now support Android APK as well).
- Multiple path exploration: This tool recognizes a branching point whenever a control flow decision is based on data outside the monitored process. Then it is detected if a control flow decision is based on a return value of a system call (e.g., the current system time). Multiple path exploration is a countermeasure of logic bombs.
- CWSandbox: a virtual environment based on invoking API functions by injecting DLL to the running process. Then it analyze malware behavior and produce a malware analysis report.
- Norman Sandbox: also used in [3] to reference to anti-emulation code. The Norman Sandbox provides a simulated environment with fake impression that it is running on a real system to the sample under analysis that consists of custom-made versions of user-land APIs necessary for the sample to execute.

3.3. Bare-metal Analysis-based Evasive Malware Detection

Nowadays, malwares become more and more sophisticated especially improved techniques to identify analysis environment and refraining from performing malicious actions. In this section, I introduce the research of BareCloud [5] - an automated evasive malware detection system based on bare-metal dynamic malware analysis. This is a novel approach of hierarchical similarity-based malware behavior comparison to analyze the behavior of a sample in the various analysis systems.

3.3.1. Bare-metal system overview

The bare-metal malware analysis system is a cluster of hardware-based modular worker units based on a remote storage disk. The bare-metal system also has a software-based remote control mechanism to automate the malware execution process on the workers. This mechanism is based on the IPMI remote administration features,iSCSI protocol (Internet Small Computer System Interface) to attach remote disks to worker units, Logical Volume Manager (LVM)-based copy-on-write snapshots to host the remote disk used by the analysis system running on the worker units. After the completion of each malware analysis, the corresponding volume snapshot is recreated from a clean volume.

The bare-metal malware analysis system is a real hardware-based hence there is really difficult for malware to identify the analysis environment. With these network log and disk capture, the authors make a behavior comparison after applying behavior deviation, behavioral profile, behavior extraction and hierarchical similarity.

3.3.2. Evaluation

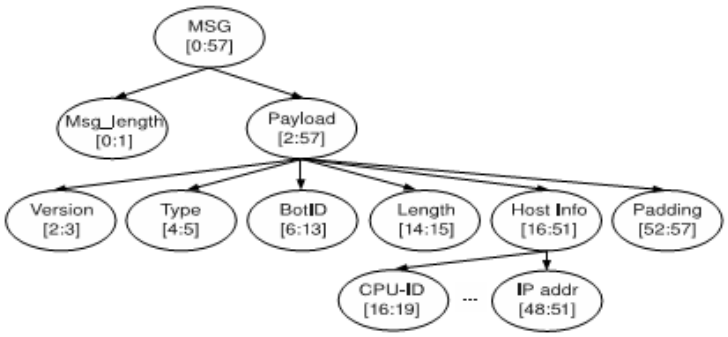
The experiments show that research produces better evasion detection results compared to previous methods. BareCloud was able to automatically detect 5,835 evasive malware out of 110,005 recent samples.

4. Automatic protocol RE

Previous sections focus on techniques to analyze and classify malware based on hooking functions, system calls. However, understanding the C&C botnet's protocol is a hard work and analysts need to rewrite every sent and received messages. In the next section, I will introduce Dispatcher [2] – an automatic protocol RE to analyze botnet networking communication.

4.1.Dispatcher implementation overview

The captured output buffer (in case of encrypted protocols, they capture the buffer before being encrypted and sent or they capture the buffer after being decrypted and received, therefore they can obtain unencrypted data) will be *deconstructed* to build message field tree.



Field Semantics	Received	Sent
Cookies	yes	yes
IP addresses	yes	yes
Error codes	no	yes
File data	no	yes
File information	no	yes
Filenames	yes	yes
Hash / Checksum	yes	yes
Hostnames	yes	yes
Host information	no	yes
Keyboard input	no	yes
Keywords	yes	yes
Length	yes	yes
Padding	yes	no
Ports	yes	yes
Registry data	no	yes
Sleep timers	yes	no
Stored data	yes	no
Timestamps	no	yes

Fig 3. *Message field tree and Field semantics identified by Dispatcher*

To infer the field semantics, Juan Caballera et al. use type-inference-based technique that leverage the observation that many functions and instructions are used by known programs contain semantic information. Such interference is general and can be used to identify large scale of field semantics such as cookie, timestamps, hostname, port, IP address etc.

Combining 2 above solutions, the authors introduced Dispatcher which firstly makes a forward pass over the execution trace collecting information needed using loop analysis, callstack analysis and buffer liveness analysis. Then it use a recursive process to deconstruct the buffer based on program locations, dependency chains (a sequence of write operations). Dispatcher infer the field attributes eventually by determine: keywords, length fields, delimiters, variable-length fields and arrays.

4.2.Evaluation

The authors evaluate these techniques on MegaD C&C which uses a proprietary encryption protocol on port 443 (instead of SSL) and they successfully rewrite and summarize its field semantics and compare this with unencrypted messages using BinPac. Besides, they evaluate their techniques on 4 open protocols: HTTP, DNS, FTP and ICQ. The results show that the accuracy of the message format automatically extracted by Dispatcher can rival that of Wireshark, without requiring a manually generated grammar.

5. Automatically Reconstruct Android Malware Behaviors

With a huge user base of Anroid mobile devices, Android malwares are rising with an alarmingly pace. This section I will summarize basic concepts of CopperDroid [6], an approach built on top of QEMU to automatically perform out-of-the-box dynamic behavioral analysis of Android malware.

5.1.CooperDroid

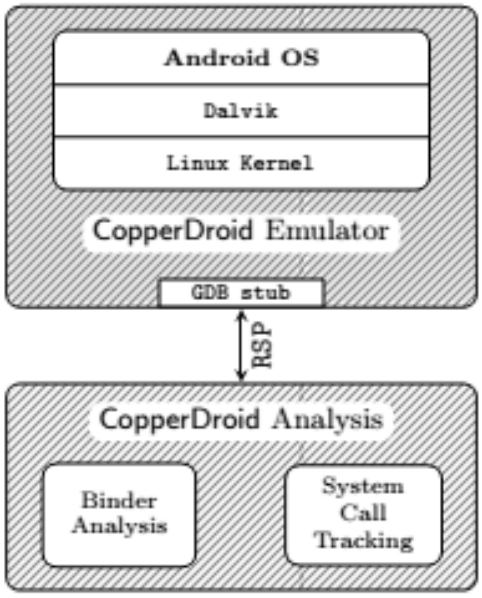


Fig 4. *CooperDroid Architecture*

The whole Android system runs on top of the CopperDroid emulator based on qemu (a generic and open source machine emulator and virtualizer). It enhances the Android emulator to enable system call tracking and support our out-of-the-box system call-centric analyses. The communication between host and emulator is using GDB support.

- Tracking System Call Invocations: This is the basic dynamic malware analysis technique mentioned in 2.4. However, the ARM architecture underlying the Android operating system and CopperDroid **swi** instruction for invoking system calls which are different from Windows/Linux OS.
- Binder Analysis: Dissecting interprocess communication (IPC) and remote procedure calls (RPCs). Android uses Binder which allows a Java process to invoke methods of remote objects as if they were local methods, through synchronous calls. CopperDroid carry out a detailed analysis of communication channels to specify behaviours of malicious Android applications. For example, when sending a SMS message, the system invokes service *isms* and remotely invoking its *sendText* function with the proper arguments. CopperDroid catches the arguments of each binder-related ioctl system call to reconstruct the remote invocation.
- Path Coverage: Unanalysed path is exacerbated when mobile applications are user driven and interaction with applications. To address this problem, CopperDroid implements an approach based on extracting information from the Manifest to stimulate the analyzed sample with a number of events of interest. For example, injecting events such as phone calls and reception of SMS texts would lead to the execution of the registered application's broadcast receivers.

5.2.Results

The authors observed an average of 28% additional behaviors on more than 60% of the Android Malware Genome Project's samples, and an average of 22% additional behaviors on roughly 73% of the Contagio's samples.

6. References

[1] Egele, Manuel, et al. "A survey on automated dynamic malware-analysis techniques and tools." ACM Computing Surveys (CSUR) 44.2 (2012): 6.

[2] Caballero, Juan, et al. "Dispatcher: Enabling active botnet infiltration using automatic protocol reverse-engineering." Proceedings of the 16th ACM conference on Computer and communications security. ACM, 2009.

[3] Wagener, Gérard, Radu State, and Alexandre Dulaunoy. "Malware behaviour analysis." Journal in Computer Virology 4.4 (2008): 279-287.

[4] Shahid Alam, R.Nigel Horspool, Issa Traore, Ibrahim Sogukpinar. “A framework for metamorphic malware analysis and real-time detection”. Computers & security 48 (2015) 212-233

[5] Dhilung Kirat, Giovanni Vigna, and Christopher Kruegel. “BareCloud: Bare-metal Analysis-based Evasive Malware Detection”. Proceedings of the 23rd USENIX Security Symposium. ISBN 978-1-931971-15-7

[6] Alessandro Reina, Aristide Fattori, Lorenzo Cavallaro. “A System Call-Centric Analysis and Stimulation Technique to Automatically Reconstruct Android Malware Behaviors”. In the Proceedings of the 6th European Workshop on Systems Security (EuroSec).