

Real-time Detection of Malware Activities and Analysis of Behavioral Differences Between IoT Malware

韓, 燦洙

<https://hdl.handle.net/2324/4475145>

出版情報 : Kyushu University, 2020, 博士 (工学) , 課程博士
バージョン :
権利関係 :



Real-time Detection of Malware Activities and Analysis of Behavioral Differences Between IoT Malware

Chansu Han

December, 2020

**Department of Informatics,
Graduate School of Information Science and Electrical Engineering,
Kyushu University**

Contents

List of Figures	vi
List of Tables	viii
Abstract	ix
Acknowledgments	xii
1 Introduction	1
1.1 Background	1
1.2 Motivation and Challenge	2
1.3 Approach	4
1.4 Contribution	6
2 Background	9
2.1 Darknet	9
2.2 IoT Malware and Botnet	12
2.3 Boosting Cybersecurity with Machine Learning Techniques	14
2.3.1 Automation Technology Required for Cybersecurity	14
2.3.2 Machine Learning Technology Applied to Cybersecurity	15
2.3.3 Analyzing Communication and Network Events	16

2.3.4	Analyzing Binary and Source Code	19
2.3.5	Analyzing Threat Intelligence	20
2.3.6	Preparing Dataset	22
2.3.7	Direction of Future Research	24
3	Dark-GLASSO: Real-time Detection of Malware Activities Using Graphical Lasso	25
3.1	Introduction	25
3.2	Preliminaries	28
3.2.1	Graphical Gaussian Model	28
3.2.2	Graphical Lasso	30
3.3	Related Work	31
3.4	System Overview: <i>Dark-GLASSO</i>	33
3.4.1	Creating Dataset D	34
3.4.2	Sparse Structure Learning: Graphical Lasso	35
3.4.3	Quantification: Graph Density	35
3.4.4	Outlier Detection	35
3.4.5	Procedure for Online Processing	37
3.5	Parameter Tuning	38
3.5.1	Length T of Time Slot	38
3.5.2	The Number of Time Series Samples M	38
3.5.3	Regularization Coefficient r	39
3.5.4	The Number of Graph Densities K and Threshold θ for Alert Judgment	40
3.6	Performance Evaluation	40
3.6.1	Demonstration Experiment	40
3.6.2	Analysis of Alert Result	42
3.6.3	Detection Performance	44
3.6.4	Details of Detected Cyberthreat	47
3.7	Discussion	51

3.7.1	Limiting the Number of Source Host: β	52
3.7.2	Limitation	55
3.8	Conclusion	56
4	Dark-NMF: Real-time Detection of Malware Activities Using NMF	57
4.1	Introduction	57
4.2	Related Work	59
4.3	System Overview: Dark-NMF	60
4.3.1	Overview	60
4.3.2	Step 1: Creating Observation Matrix V	61
4.3.3	Step 2: Applying NMF	61
4.3.4	Step 3: Determining Anomalous Spatial Feature Group	62
4.3.5	Step 4: Issuing Alert	63
4.4	Implementation and Dataset	63
4.4.1	Hyperparameter Tuning	64
4.4.2	Dataset and Ground Truth	64
4.5	Performance Evaluation	66
4.5.1	Experiment 1: Anomalous Synchronization Detection of Source Host Feature Space	66
4.5.2	Experiment 2: Anomalous Synchronization Detection of Destination Port Feature Space	67
4.5.3	Comparative Evaluation with Related Works	68
4.6	Discussion: Dark-NMF vs. Dark-GLASSO	71
4.7	Conclusion	72
5	FCSG: Investigating Behavioral Differences between IoT Malware	75
5.1	Introduction	75
5.2	Preliminaries	77

5.2.1	Instruction Set Architecture	78
5.2.2	Disassemble by IDA Pro	78
5.3	Proposed Method	79
5.3.1	Overview	79
5.3.2	Function Call Sequence Graph	80
5.3.3	Signature Matching	83
5.4	Implementation	84
5.4.1	Outline of Proposed Method	84
5.4.2	Generating Signature-FCSGs	85
5.4.3	GenerateFCSGs	87
5.4.4	SignatureMatching	89
5.5	Investigation	91
5.5.1	Dataset	91
5.5.2	Signature Matching Results	93
5.5.3	Investigation 1: Specimens Mixed with Malware Families	93
5.5.4	Investigation 2: Specialized Specimens for Some behaviors	95
5.5.5	Investigation 3: Specimens Matched with All Signatures	96
5.5.6	Investigation 4: Specimens Unmatched with Any Signatures	97
5.5.7	Investigation 5: Specimens in Other Malware Families	99
5.6	Related Work	99
5.7	Conclusion	100
6	Conclusion	101
	References	105
	Publications	116

List of Figures

2.1	An outline of darknet	10
2.2	Total number of observed packets per 1 IP address on the darknet.	11
2.3	The timeline of Mirai	13
3.1	The example of GGM graph 1	29
3.2	The example of GGM graph 2	29
3.3	The overview of <i>Dark-GLASSO</i>	33
3.4	The number of source hosts per one hour in October 2018 at TCP ports where Mirai-related cyberthreats were observed. (sensor A, part I)	48
3.5	The number of source hosts per one hour in October 2018 at TCP ports where Mirai-related cyberthreats were observed. (sensor B, part II)	48
3.6	The number of source hosts per one hour in October 2018 at TCP ports where Mirai-related cyberthreats were observed. (sensor A and NICTER, part III)	49
3.7	The number of source hosts per one hour in October 2018 at TCP ports where router vulnerability-related cyberthreats were observed. (sensor A)	49
3.8	The number of source hosts per one hour in October 2018 at TCP ports where other vulnerability-related cyberthreats were observed. (sensor A, part I)	50
3.9	The number of source hosts per one hour in October 2018 at TCP ports where other vulnerability-related cyberthreats were observed. (sensor A, part II)	51
4.1	Overview of Dark-NMF.	60

4.2	Three-dimensional plot of the number of packets per minute as an example of the alert result in Experiment 2.	69
5.1	Overview of proposed approach (yellow: inputs; orange: outputs; blue: automatic process; green: manual process)	80
5.2	The control flow of <code>main</code> function from ARM binary (in Program 5.1)	82
5.3	The control flow of <code>get_random</code> function from ARM binary (in Program 5.1)	83
5.4	FCSGs for each control flow (in Fig. 5.2 and Fig. 5.3)	84
5.5	The partial control flow of <code>main</code> function from MIPS binary (in Program 5.1)	85
5.6	The partial control flow of <code>get_random</code> function from MIPS binary (in Program 5.1)	86
5.7	Combined FCSG from Fig. 5.4 based on <code>main</code> function	87
5.8	Sample signature-FCSG	87
5.9	Generated FCSG from a Bashlite specimen	91
5.10	Matched signature-FCSG on Fig. 5.9's FCSG (part 1)	92
5.11	Matched signature-FCSG on Fig. 5.9's FCSG (part 2)	92

List of Tables

3.1	The number of observed IP addresses (IPs), alerts, and unique ports for each darknet sensor	42
3.2	Ground truth list of TCP ports used for cyberthreats observed on our darknets in October 2018	44
3.3	Evaluation results of the detection performance of <i>Dark-GLASSO</i> and ChangeFinder (confusion matrix	45
3.4	Evaluation results of the detection performance of <i>Dark-GLASSO</i> and ChangeFinder (4 major metrics)	46
3.5	Average performance when <i>Dark-GLASSO</i> was run five times for each β , using a confusion matrix	52
3.6	Average performance when <i>Dark-GLASSO</i> was run five times for each β , using five evaluation metrics	53
3.7	Summaries on the number of source hosts and runtime per time slot for each sensor (Min, Median, Mean, Max), and the setting of β in the demonstration experiment	54
4.1	Results of Experiment 2 for each α and each sensor. Integration is the result of integrating all eight sensors. (#TPs, #FPs)	67
4.2	Malware activity detection results of ChangeFinder, Dark-GLASSO, and Dark-NMF for each sensor. Integration is the result of integrating all eight sensors. (#TPs, #FPs)	70

4.3	Results of Experiment 1 with four tuned hyperparameters $\{u, f, \alpha, \beta\}$. (#TPs, #FPs)	74
5.1	List of created Bashlite signatures	89
5.2	List of created Mirai signatures	90
5.3	Breakdown of dataset by ISA and malware family	93
5.4	Order and size of Signature-FCSGs and FCSGs	94
5.5	How many Bashlite signatures out of 13 signatures were matched to specimens in the dataset (e.g., five Bashlite signatures were matched to 40 Tsunami specimens). The signatures are listed in Table 5.1.	95
5.6	How many Mirai signatures out of 15 signatures were matched to specimens in the dataset. The signatures are listed in Table 5.2.	96
5.7	The number of specimens matching each Bashlite signature	97
5.8	The number of specimens matching each Mirai signature (None of the signatures were matched to Tsunami and Ligtaidra)	98

Abstract

Large-scale cyber attacks that exploit Internet of Things (IoT) devices have been frequently confirmed in recent years, and IoT malware has become diversified and sophisticated. Infection with such malware has become a global social problem. Therefore, an early understanding of infection behavior and information sharing among related organizations is essential, but countermeasures are currently insufficient. As a result, after a security incident occurs, the initial response is delayed, often leading to serious incidents such as the leakage of confidential information. To solve this problem, it is necessary to automate the analysis of malware attack behavior based on the data collected in many ways. The main goal of this thesis is to complete the analysis of malware attack behavior and derive early warning information before it becomes a significant incident.

Specifically, the first challenge is the early detection of malware activities by darknet analysis. It is essential to rapidly and accurately detect malware activities when malware infection spreads. However, accurately distinguishing potential malware activities from countless indiscriminate scanning attacks is a huge challenge. A darknet, an unused IP address space, has a high signal-to-noise ratio, making it easier to know the global trend of ill-intentioned traffic in cyberspace than other monitoring networks.

In this study, we aim to capture global cyberthreats in real-time based on darknet analysis. Multiple hosts infected with similar malware that share a scan module tend to scan in a similar spatiotemporal pattern to search for the next infection target, the tendency observed on the darknet. We proposed a system, Dark-GLASSO, that estimates a degree of synchronizations from the patterns of packet transmission time among the source hosts observed in unit

time of the darknet and detects anomalies in real-time. Also, we analyzed Dark-NMF, which focuses on synchronizing the spatiotemporal features seen when malware infection spreads and detects abnormally synchronous spatial features (source hosts and destination ports) in real-time. Dark-NMF measures the synchronization of spatial features by decomposing spatiotemporal patterns from darknet traffic using Nonnegative Matrix Factorization (NMF). We tuned the hyperparameters of Dark-NMF and evaluated the detection performance of malware activities against the performance of Dark-GLASSO and an existing method, ChangeFinder, using human-labeled ground truth.

In our evaluation, we performed our proof-of-concept implementation of Dark-GLASSO to demonstrate its feasibility and effectiveness, and we detect cyberthreats with an accuracy of 97.14%. In addition, We found that Dark-NMF detects all malware activities that should be detected in the ground truth without a miss. We also showed that Dark-NMF has many advantages over existing methods and provided a highly practical operation guideline. These works are the first practical trial that detects cyberthreats from in-the-wild darknet traffic regardless of new types and variants in real-time, and it quantitatively evaluates the result. Hence, we expect Dark-GLASSO and Dark-NMF to contribute to the development of rapid responses to malware activities.

The second challenge is to investigate the functional differences between a large number of malware variants by malware analysis. IoT malware that infects IoT devices is rampant. Most IoT malware variants are generated by changing various behaviors such as an attack method based on existing malware families. Nearly all antivirus software only identifies the malware family's name; thus, we cannot acquire further details about differences between malware behaviors. In this study, we propose a graph-based method for confirming differences in malware behaviors and investigating the actual conditions of malware variants. The proposed method first extracts a sequence of function calls from a binary file of malware and represents the sequence to a directed graph, which we refer to as a function call sequence graph (FCSG). Next, the method automatically checks if the FCSG matches signature-FCSGs, which are manually generated as small-scale FCSGs representing malicious behaviors of known malware such as a function of attacks and network scans. To demonstrate the usability of our proposed method,

we applied the proposed method to 24,126 in-the-wild IoT malware specimens and investigated the existence of specimens with mixed behaviors from multiple malware families or were specialized for some attacking behaviors.

This thesis is composed of the following chapters. In Chapter 1, we describe the research background, challenges, and motivation in this thesis. We also discuss our approaches and contributions to this study. Chapter 2 introduces the darknet, a trap-based indiscriminate attack monitoring network, and details the current state of IoT malware, which has caused tremendous damage in recent years. We also introduce a survey of the cybersecurity field, where machine learning is being actively practiced, from a broad aspect. The darknet analysis systems Dark-GLASSO and Dark-NMF that detect malware activity in real-time are described in Chapters 3 and 4. Chapter 5 introduces the function call sequence graph (FCSG), a malware analysis system investigating functional differences between a large number of malware variants. Finally, Chapter 6 presents our conclusions and discusses future works.

Acknowledgments

Until the undertaking this Ph.D., I received warm help from so many people and had a valuable experience. I would like to thank all of them. Thanks to all of them, I was able to proceed with my research in a comfortable place.

First of all, I am extremely grateful to my supervisor, Prof. Jun'ichi Takeuchi, for his continuous support, invaluable advice, and patience during my Ph.D. study. His immense knowledge and plentiful experience have encouraged me all the time. His enthusiasm and dedication to academic research have been very inspiring to me.

I would like to thank Associate Prof. Shuji Kijima from Kyushu University, Associate Prof. Katsunari Yoshioka from Yokohama National University, and Prof. Noboru Murata from Waseda University for their valuable comments and advice in advancing the doctoral research. I would also like to thank Assistant Prof. Masanori Kawakita, who currently belongs to Nagoya University. I am also very grateful to all the members of our laboratory who conducted the joint research in this research project.

The research experience at Cybersecurity Laboratory, National Institute of Information and Communications Technology (NICT) has deepened my understanding of the academic field of cybersecurity. I greatly appreciate Dr. Daisuke Inoue, who lead the laboratory, Mr. Koji Nakao, and other researchers, engineers, technical staff, and all the members. In particular, this research could not have been achieved without the support of our Artificial Intelligence for CyberSecurity (AICS) team members led by Dr. Takeshi Takahashi. I would also like to thank Mr. Jumpei Shimamura for his great professional cooperation in creating ground truth and evaluation experiments in darknet analysis, and Dr. Ryoichi Isawa for his technical

cooperation in malware analysis.

This research was partially conducted under a contract of “Research and development on IoT malware removal / make it non-functional technologies for effective use of the radio spectrum” among “Research and Development for Expansion of Radio Wave Resources (JPJ000254)”, which was supported by the Ministry of Internal Affairs and Communications, Japan.

Last but not least, I would like to express my gratitude to my dearest wife, Erina, my lovely daughter, Saran, and my parents and family. Without their tremendous understanding and encouragement, it would be impossible for me to complete my study.

Chapter 1

Introduction

1.1 Background

It is predicted that the number of Internet of Things (IoT) devices connected to the Internet worldwide exceeds 50 billion by 2020 [1]. Under such circumstances, IoT devices are being used in many industries such as medical care, agriculture, automobiles, and in 5G, a new wireless communication infrastructure. On the other hand, distributed denial of service attacks (DDoS attacks) by IoT malware that infects these IoT devices are frequently occurring. According to the observation and analysis results of cyberattack activities conducted by the National Institute of Information and Communications Technology (NICT), a large number of cyberattack scans on ports related to IoT devices (TCP ports 22, 23, 80, etc.) were observed [2]. Many scans have been analyzed as cyberattacks targeting IoT devices. Such scans have reached about half of the total observation traffic, and there is substantial concern about the increase in attacks using infected IoT devices.

In fact, in October 2016, a DDoS attack using more than 100,000 IoT devices infected with the malware “Mirai” caused DYN’s domain name service (DNS) service to stop [3]. In May 2018, it was confirmed that more than 500,000 IoT devices were infected with the malware VPNFilter in 54 countries worldwide [4]. In recent years, large-scale cyberattacks with traffic exceeding several hundred Gbps to 1 Tbps that exploit such IoT devices have been frequently

confirmed.

Furthermore, at present, IoT malware has expanded to the following problems. The emergence of diverse and sophisticated IoT malware has further complicated cyber threats, making it difficult to prevent damage caused by IoT malware. For example, early Mirai used Telnet service (TCP port 23) to spread the infection, but cyberattacks on other service ports have increased, and cyberattacks have diversified in recent years [2]. Conventional observation and analysis technologies are no longer able to fully fulfill the functions of cyberattacks on IoT that are diversifying and IoT malware that is becoming more sophisticated. The IoT devices developed and sold by various vendors worldwide are extremely diverse and have various vulnerabilities.

In addition, although conventional IoT malware has high infectivity, it is known that it can be erased by turning off the power of the infected device and restarting it, and the removal work itself was easy. However, in recent years, there have been some reports of persistently infectious IoT malware that remain infected even after the device is restarted [5]. It is difficult for general users to remove persistently infectious malware in IoT devices that do not have widely used countermeasure tools such as antivirus software. Infected devices continue to carry out cyberattacks until the devices are discarded and no longer used. Furthermore, more than 500 types of vulnerable IoT devices have been confirmed to be infected with malware, and the problem of IoT devices whose administrator is unknown has been unveiled.

1.2 Motivation and Challenge

As described, malware infection has become a global social problem, and cyber threats to organizations such as governments and critical infrastructure are steadily increasing. However, early understanding of infection behavior and sharing among related organizations are insufficient. As a result, the initial response after a security incident occurs is delayed, leading to serious incidents such as the leakage of confidential information in many cases.

To resolve this issue, it is necessary to detect the possibility of a security incident earlier and collect and analyze information on the malware and vulnerabilities that caused the incident.

Since these tasks require analysis of a large amount of data, manual analysis is unrealistic, but it is an area where the effect of automatic processing by a computer can be significantly expected. In addition, these analyzes are not completed by a single analysis, and it is necessary to judge the results of analysis from various information sources comprehensively. Based on the data collected from many sources, this thesis's primary goal is to automate the analysis of malware attack behavior, complete the behavior analysis before it becomes a significant incident, and derive early warning information.

Specifically, the first challenge is the real-time detection of malware activities by darknet analysis. Malware, a self-propagating mechanism via a network, indiscriminately examines more vulnerable victims by infected victims and spreads the infection. To prevent damage caused by such indiscriminate scanning attacks, it is essential to detect potential malware activities accurately at an early stage before the infection of the malware spreads. In the darknet, indiscriminate scan attacks are observed for malware to search for vulnerable targets on the Internet one after another, which is suitable for such analysis. However, accurately distinguishing potential threats from a myriad of scanning and probing is like finding a needle in a haystack. In this challenge, efforts were made to identify indiscriminate threats quickly and accurately with few false positives and no false negatives.

The second challenge is to investigate the functional differences between many malware variants by malware analysis. Reusing a code base or sharing code logic is common in malware [6]. In addition, malware variants evolve as attackers gradually change their behaviors. Thus, it is important for malware variants to see what behaviors have been added or removed. Antivirus software analyzes malware specimens to determine which malware family a specimen belongs to. However, it is very difficult to identify what behaviors each malware specimen can invoke and what behavioral differences exist between malware specimens even in an identical malware family. In this task, without executing a large number of malware specimens, we attempt to automatically estimate the behaviors of large-scale malware specimens and extract the behavioral differences between malware specimens to investigate the actual conditions of IoT malware specimens. Here, a behavioral difference refers to changes in attack types and malware behaviors.

1.3 Approach

In the conventional method for analyzing darknet traffic, the change point is detected by ChangeFinder algorithm [7] from the time series data of the traffic volume or the number of unique source hosts to detect cyber threats [8]. However, with the increasing complexity of cyber threats, in many cases, threats cannot be detected by ChangeFinder. Hence, a new approach is required. In this thesis, we propose two darknet analysis systems.

There is no synchronization or cooperation between the packet transmission time patterns of unrelated source hosts. On the other hand, a single host tends to perform similar behavior at each darknet address [9]; hence we assume that when source hosts infected with similar malware (such as IoT malware), the source hosts are synchronizing with each other in the darknet. In fact, devices infected with similar malware that share a scan module tend to scan in a similar spatiotemporal pattern to search for next infection targets [10]. The spatial pattern refers to the source host feature space and the destination port feature space in network traffic. Also, the infected hosts that form a botnet (such as IoT botnet) have the characteristic of acting synchronously when they receive commands from a command and control (C&C) server [11]. Moreover, we consider that if synchronization among source hosts in a time slot is abnormally high, a cyberthreat event has occurred during that time slot.

However, it is difficult for humans to check synchronization among source hosts manually. To handle this issue, as a first approach, we propose a system called Dark-GLASSO [12–14], which estimates a synchronization of the source hosts in real time by using a sparse structure learning algorithm called the graphical lasso [15]. It also announces alerts in real time when the estimated synchronization changes significantly. It can detect a group of related campaigns by estimating synchronization among many pairs of source hosts for each time slot and by finding outliers in synchronization among different time slots. It is able to ignore events with weak synchronization due to an effect of ℓ_1 regularization penalty term of the graphical lasso algorithm.

As a second approach, in a previous study [16], attempts were made to detect potential malware activities by determining the source hosts that are highly synchronized with the spa-

tiotemporal patterns on darknet traffic. We call this method Dark-NMF. Dark-NMF employs nonnegative matrix factorization (NMF) to decompose darknet traffic into latent temporal and spatial patterns to determine the spatiotemporal features that present anomalous synchronization. We developed Dark-NMF into a real-time processing method. We also validated a practicality of malware activity detection with hyperparameter tuning and quantitative evaluation.

Dark-GLASSO and Dark-NMF take a similar approach that focuses on malware synchronization. Therefore, these methods are resistant to noise; packets that arrive accidentally in a darknet space due to misconfigurations will not show synchronization. However, there are differences in the calculation process and the results obtained. Dark-GLASSO employs graphical lasso [15], the algorithm that estimates conditional independence between spatial feature variables, and quantifies the degree of synchronous variables. On the other hand, Dark-NMF adopts NMF [17], the algorithm that decomposes spatiotemporal feature space into potential frequent temporal patterns and spatial patterns, and detects spatial features with high synchronization in the decomposed spatial patterns. In other words, Dark-GLASSO detects anomalies by comparing the quantified degree of synchronization with the numerical values of other periods, while Dark-NMF recognizes anomalous patterns in the decomposed spatial patterns without the need to compare the results with other periods. As a result of quantitative evaluation, it is found that Dark-GLASSO is good at detecting large-scale malware activities without false detection. Dark-NMF can detect small-scale malware activities, but it tends to have more false positives than Dark-GLASSO.

Next, we introduce the malware analysis approach in this thesis. To investigate the actual conditions of malware, especially IoT malware, we attempt to automatically estimate behavioral differences between large-scale malware specimens by focusing on which functions (e.g., `open`, `read`, `printf`, and `close` in C programming) are called and their sequence. Some of those function calls are used to implement specific behaviors of malware.

We propose a method to generate a function-call-sequence graph (FCSG) that considers both API names and names of called user-defined functions from the disassembled code of malware specimens [18]. In addition, we implement a method to avoid environmental problems in IoT malware, e.g., instruction set architecture (ISA) and compilers, by applying nor-

malization on function names or removing nodes in FCSGs. Next, we generate the FCSGs from the disassembly result or the released source code of a specific IoT malware family, and the characteristic behaviors of the malware are created manually as signature-FCSGs. When a signature-FCSG is matched to the FCSGs of a malware specimen, the malware specimen can be considered to have a behavior that corresponds to the signature-FCSG.

The proposed method is used to specify malicious behaviors of malware binaries captured by honeypot systems for investigating functional behaviors between malware, assuming that the honeypot systems do not capture any benign software (i.e., the proposed method does not target any benign software in this paper). Also, an FCSG and a signature-FCSG can handle malware on different ISAs because it focuses only on function calls. Moreover, an analyst can confirm function calls briefly, and signatures can be created easily from the source code or disassembled code.

1.4 Contribution

Dark-GLASSO in Chapter 3 offers the following contributions: First, we propose Dark-GLASSO, which statistically estimates a synchronization between the source hosts from traffic data and detects anomalies. Second, we perform a demonstration experiment of Dark-GLASSO in real time using in-the-wild darknet traffic and evaluate the detection performance of cyberthreats using the prepared ground truth. We also make a comparison with a conventional method (ChangeFinder). As a result, we achieve a detection accuracy of 97.1%, which is better than the conventional method and describe the details of the detected cyberthreats. Finally, as a discussion and a consideration, we investigate the adverse effect of preprocessing on the accuracy and describe the technical limitations of our method. In addition, we publish the datasets used for demonstration experiment on the web¹.

Next, Dark-NMF in Chapter 4 made the following contributions: First, we developed Dark-NMF into a real-time processing method. We also validated a practicality of malware activity detection with hyperparameter tuning and quantitative evaluation. Second, in conventional

¹<https://csdataset.nict.go.jp/darknet>

Dark-NMF [16], only the source host feature space is considered as a spatial feature; in contrast, in our modified Dark-NMF, the destination port feature space too is treated as a spatial feature for improving the detection accuracy of malware activities. Therefore, Dark-NMF can detect all malware activities that should be detected in the ground truth without false negatives; however, compared to the existing methods, more false positives are obtained. In malware activity detection, although fewer false positives are desired, a system without false negatives is highly valued. Finally, Dark-NMF has the advantages such as low calculation cost, no past data required for detection. Dark-GLASSO and Dark-NMF can be operated immediately in real-time, and we provide a guideline on selecting an algorithm suitable for practical operation.

Finally, the FCSG in Chapter 5 provides the following contributions: To the best of our knowledge, this is the first work to extract behaviors contained in IoT malware specimens in a graph, confirm the behavioral differences between specimens using graph-based signatures, and discover specimens containing activities from multiple malware families. In this study, we used FCSGs to investigate behaviors of 24,126 in-the-wild IoT malware specimens running on Linux of IoT devices and confirmed that ISA-independent signature matching is possible. First, we propose generating a FCSG from the disassembled code of a malware specimen and estimating the behavioral differences among large-scale malware specimens regardless of ISAs or malware families. Second, we identify mixed variants with multiple malware family features that match some signature-FCSGs generated from the source code of different malware families. Finally, we identify malware variants that may have been removed from some behaviors and specialized in some attacking behaviors.

Regarding the results of this research, the following effects and social effects can be achieved. First, the activity of the initial stage of cyber attacks caused by the malware can be derived, and the malware activities can be captured before the incident caused by the malware develops on a large scale. Therefore, it is possible to reduce malware damage in cooperation with organizations that carry out security operations and incident response. In addition, by identifying similar malware families and groups and investigating functional differences, information can be quickly provided to organizations that perform malware behavior analysis, and highly accurate malware analysis can be realized. The above effects are expected in security operations

centers and security incident response teams in organizations.

Chapter 2

Background

This chapter introduces the darknet, a trap-based indiscriminate attack observation network, and describes the current state of IoT malware, which has caused enormous damage in recent years. In addition, we introduce an overview of the cybersecurity field, where machine learning is being actively applied, from a broad perspective.

2.1 Darknet

In this section, we describe the background of the darknet. As illustrated in Fig. 2.1, the darknet is a network of unused IP addresses. We have set up receivers (sensors) on our darknets and passively capture all the traffic arriving at these receivers. In theory, any packets should not arrive at darknets because they are not connected to any devices. However, in fact, quite few packets do arrive. Almost all packets reaching darknets are caused by malicious activities. In the darknet, indiscriminate scanning attacks from malware with a propagation mechanism are globally scoped, regardless of geographic or topological boundaries. Types of packets include scans and attacks by malware, backscattering (a reflection of DDoS attack), results of misconfiguration, etc.

Darknet traffic reflects global trends in malicious activities on the Internet (particularly the status of worm-type malware infections) [19]. Most of the packets in the traffic are TCP

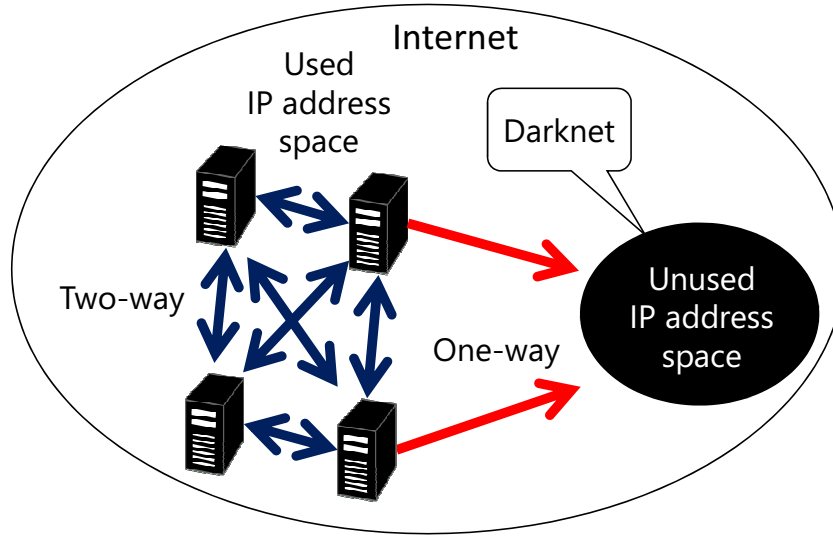


Figure 2.1: An outline of darknet

packets with the SYN flag on, i.e., TCP-SYN packets. TCP-SYN packets reaching a darknet are non-ordinary packets; these are often scans or unintended packets sent by misconfigured network devices. The scans may be initiated by malware for indiscriminately locating the next victim candidates or by any other entities for investigating and researching the Internet. In other words, by analyzing darknet traffic, it is possible to capture trends of cyber attacks and malware infections in real-time. Accordingly, darknets are very useful to observe the behavior of serious attacks over the Internet.

In this study, we use darknet traffic monitored and logged by the Network Incident Analysis Center for Tactical Emergency Response (NICTER) [20]. The purpose of NICTER is that it enables the early detection of such incidents, which are detrimental to networks, and determines the quickest and most effective countermeasures. Darknets observed by NICTER are spread in various countries through international collaboration, and the scale of observation is about 300,000 IP addresses in 2017. Each sensor has different IP subnets (IP network parts), and one sensor belongs to almost the same subnet.

Fig. 2.2 shows that our 300,000 darknet IP addresses witnessed a surge in traffic over

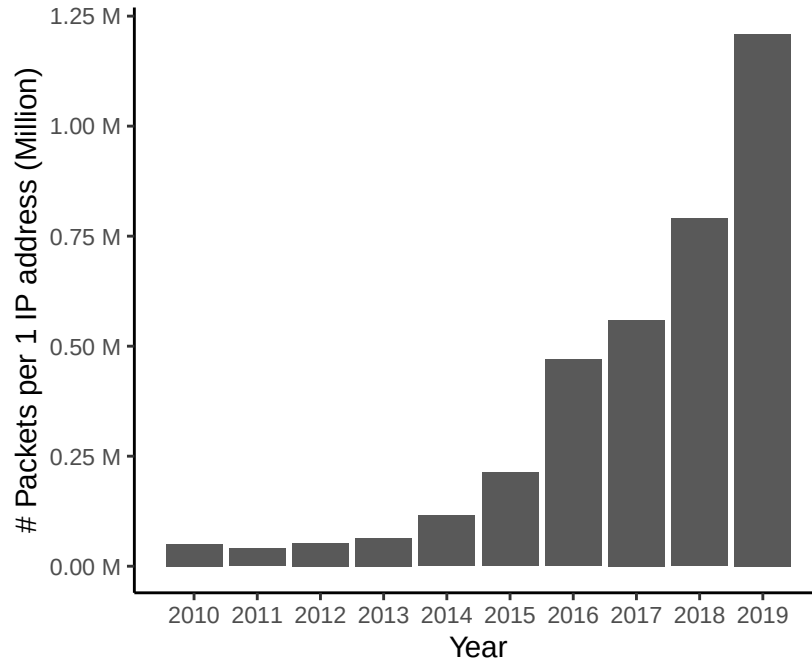


Figure 2.2: Total number of observed packets per 1 IP address on the darknet.

the last few years. A major cause is the IoT malware that appeared in 2016 [3]. In 2019, the total of observation packets per one IP address was 1.2 million packets combined with TCP and UDP protocols from NICTER's darknet. In this thesis, a network scan that has indiscriminately attempted attacks and intrusions by malware through vulnerable TCP ports is called a cyber attack, and a network scan that has indiscriminately attempted surveys and research by organizations like Shodan¹ and Censys² is called a survey scan.

We investigated studies that used darknets as a data source. Many studies using darknets to research countermeasures against malicious activities are ongoing. These can be classified into two categories, managing and designing darknet platforms and darknet analysis. Bailey *et al.* introduced the Internet Motion Sensor (IMS), to measure, characterize and track threats [9,21]. Choi *et al.* proposed a security monitoring and response model to analyze cyberthreats trend and to trace potential attackers [22]. NICTER is also a study of darknet platforms [23].

¹<https://www.shodan.io/>

²<https://censys.io/>

Under darknet analysis, since three-way handshaking is not performed on darknet traffic, and there is almost no payload data, most botnet detection techniques targeted for real network traffic can not usually be applied in a darknet. Therefore, a new method for darknet analysis is necessary. Ali *et al.* classified DDoS attacks by using Resource Allocating Network with Locality Sensitive Hashing (RAN-LSH) in which data to be trained are selected by using LSH, and fast online learning is actualized by training only selected data [24]. Ban *et al.* proposed an abrupt-change detection algorithm that can detect botnet-probe campaigns with a high detection rate by exploring the temporal coincidence in botnet activities visible in darknets [25].

2.2 IoT Malware and Botnet

The aim of this study is the detection of malware activities on the Internet by analyzing traffic data. Primarily we are paying attention to IoT malware and botnets. The botnet refers to a network consisting of an unspecified number of computers hijacked by a malicious person (commander, herder, botmaster). Types of botnet attacks include DDoS (Distributed Denial of Service) attacks, adware, spyware, and spam emails. Figure 2.3 shows the recent trend of “Mirai,” which is a representative type of IoT malware. In the autumn of 2016, the malware “Mirai” made unprecedented large-scale and destructive DDoS attacks. From September to October 2016, a DDoS attack exceeding 1 Tbps at the peak was reported on Krebs on Security, OVH, and DYN. In addition, since the source code of Mirai has been released, and its variants and subspecies are being created in large quantities, there is a high possibility that attacks and damages will be expanded in the future.

Though a communication volume for botnets is vast, that for each of bots is small, hence it is challenging to identify the botnet commander (Herder) in general. Bots are becoming more sophisticated increasingly, such as not functioning until actually connecting to the Internet. There are many variants in bots; there is a limit to the detection of bots by signatures. The paper [11] pointed out that botnets have the following properties:

- Relationship: A botnet has a one-to-many relationship between the bot commander and bots.



Figure 2.3: The timeline of Mirai

- Response: After bots receive commands from their master, they perform with a constant response time.
- cooperativeness (synchronization): All bots may be synchronized with each other.

Unlike conventional botnets, IoT malware composes a large-scale botnet with a higher infection spread speed owing to their aggressive scanning, e.g., parallel scans to multiple ports [26]. Moreover, they tend to be discarded and rebuilt within a short period. The emergence of diverse and sophisticated IoT malware has further complicated cyber threats, making it difficult to detect IoT botnets. For example, early Mirai used Telnet to spread the infection, but attacks on other service ports have increased, and attacks have diversified in recent years [2]. In addition, multiple IoT malware with “persistent infectivity” that are not deleted even when the device is turned off were discovered [5]. Furthermore, more than 500 types of vulnerable IoT devices have been confirmed to be infected with malware, and the problem of IoT devices whose management entity is unknown has been revealed. Hence, a system for conducting investigations while the IoT botnet is sustained and the prompt and accurate detection of potential threats is essential.

In [27], network-based botnet detection technology characteristics are classified into three main classes: detection algorithms, detection techniques, and detection sources. According to [27], all the papers surveyed had used heuristic-based rules at some point in the analysis. In [28], botnet detection approaches were compared based on five key features: the ability to detect unknown bots, the capability of botnet detection regardless of botnet protocol and structure, the ability to detect botnets with encrypted C&C channels, real-time detection, and

overall accuracy.

Many techniques have been studied for detecting botnets via cooperative behaviors. Bot-Sniffer uses statistical algorithms to detect botnets in a centralized topology based on their multiple crowd-like behaviors [10]. BotMiner uses a detection framework that extends Bot-Sniffer to perform clustering with monitored C&C communication and malicious activities, respectively, and issue final detection results by cross-correlation [29]. Strayer *et al.* proposed a temporal correlation algorithm on packet inter-arrival time and packet size in a five-dimensional space [30]. García *et al.* applied a clustering algorithm to detect synchronization in bots and botnets behavior; this synchronization was studied as the relationship between IP addresses, ports, and time frames only [31].

2.3 Boosting Cybersecurity with Machine Learning Techniques

When improving the efficiency and automation of security operations, it is conceivable to transfer human knowledge and know-how to computers and gradually replace them with computers. Machine learning is a very effective method to achieve this. This section introduces examples of the use of machine learning in several fields to give an overview of how machine learning is utilized in research and development in the cybersecurity field.

2.3.1 Automation Technology Required for Cybersecurity

To respond to the increasing number of cyber threats, it is necessary to carry out various security operations such as vulnerability management and malware analysis. However, the human resources required to realize it entirely are insufficient, and it is essential to expand it and improve the efficiency and automation of operations. Education-related institutions such as universities are making efforts for the former, but the latter is an area that we researchers consider. Here, it is necessary to transfer human knowledge and know-how to computers and gradually consider alternative methods using computers. Machine learning is an effective method of achieving this and is being actively utilized.

Machine learning has been widely used in image recognition, text classification, sentiment analysis, voice recognition, and virtual personal assistant. Its development in recent years has been remarkable. The cybersecurity field has already been used in various fields such as spam mail filtering, phishing detection, fake news detection, fake review detection, biometric authentication, malware analysis, and automatic driving security. It is not a recent event, and spam email filtering, for example, has been under consideration for more than ten years [32]. Since that time, the possibility of attacks that exploit machine learning weaknesses has been investigated. However, recently, it has become easier to obtain a computer environment and tools essential for applying machine learning. Therefore, attempts are being made to utilize machine learning in a broader variety of cases. The approach of machine learning is no longer novel, and machine learning has become an indispensable tool for achieving strong cybersecurity.

2.3.2 Machine Learning Technology Applied to Cybersecurity

The technical area covered by automation of security operations is wide. The possibility of utilizing machine learning in all areas of various security operations is considered. However, to use machine learning, research and development can only be carried out in areas where the dataset to be learned can be prepared. From the viewpoint of dataset type, this area can be roughly organized into three areas: communication traffic log analysis, software binary and code analysis, and threat intelligence analysis. Research on malicious communication detection, malware detection, and intelligence analysis is being conducted in these areas. In reality, to automate cybersecurity operations with high accuracy, it is often necessary to carry out analysis across these multiple areas. Although it is not meant to examine these areas in detail this time, we would like to take a bird's-eye view of the current state of research and development by picking up some areas in general along this area division. The related research areas are wide and not limited to the above. For example, there have been many reports on privacy-related technologies that identify posters from anonymized posts and the security of machine learning itself, but these are omitted in this section.

2.3.3 Analyzing Communication and Network Events

In the research area for analyzing communication, there are such things as analyzing communication packets itself on the network or analyzing communication event logs. There are various communication types, such as communication that is different from regular communication and technology that detects communication patterns peculiar to malware. The research areas we are tracking are introduced below.

Automation of Network Intrusion Detection There are two types of network intrusion detection systems (NIDS): signature and anomaly detection. In addition, there are packet capture (PCAP) and NetFlow as data used in network traffic analysis. Although the signature method has a low false-positive rate, it cannot detect unknown attacks (zero-day attacks) for which no signature is registered. The signature method uses a machine learning method to classify attack patterns from known signatures or extract new signatures from anomalous data. For example, it is sufficient to generate human-readable signatures using decision trees, genetic algorithms, and correlation rule mining. Although the anomaly detection method may detect zero-day attacks, the false positive rate is relatively high. The anomaly detection method uses a method of classifying and clustering abnormal network traffic. For example, even a simple one-class support vector machine (SVM) achieves sufficiently good accuracy. NIDS is usually used in a hybrid system that combines both methods.

Network data may not be modeled appropriately for simple distributions such as Gaussian. Variabilities of multiple protocols, user behaviors, payloads are often complicatedly formed from multiple inseparable probability distributions or simultaneous probability distributions. Therefore, the paper [33] shows that the Bayesian network and the hidden Markov model are insufficient.

This field has been studied for a long time, but progress has still been reported. Kitsune [34] is an unsupervised anomaly detection method that trains on-site using an autoencoder for each feature of packet data and determines anomalies when the autoencoder's reconstruction error during testing is large. Kitsune extracts a total of 115-dimensional feature vectors from the

traffic data by various approaches. Moreover, Kitsune is so low in calculation cost that it can be operated on a small computer such as Raspberry Pi. Next, Nodoze [35] improves the efficiency of alert analysis work by graphically expressing the dependency of the origin of alerts from IDS. Nodoze triaged many alerts by stripping graph paths with high transition probabilities from dependency graphs and stripping alerts with overlapping origins.

Detection and Evaluation of Malicious URLs Various methods using different datasets have been reported to identify malicious URLs that cause phishing and drive-by download attacks. Research [36] proposed in which the URL character string itself is considered to have the characteristics of a malignant site and analyzed. Since this method has a purpose of realizing online learning, classification is performed using only the URL string and information about the host without using information that takes time to acquire. Specifically, it is used as hostname, primary domain, top-level domain, *whois* information, AS number, and area information. Since online learning can easily follow changing characteristics, it is considered meaningful for identifying malicious sites where characteristics are expected to change.

WebWitness [37] proposed a technology that extracts the path leading to the user's download of malware by analyzing the access log. Besides, machine learning is used to determine whether the path ultimately belongs to a social engineering attack or a drive-by download attack. When making that determination, it is used as features such as the number of hops to the exploit page and the domain's age. Classification is performed by supervised machine learning using a random forest.

There are also reports of deepening analysis specializing in social engineering. Paper [38] focuses on advertisements' social engineering attacks and proposes a technology to detect similar attacks in the future by random forest using features extracted from download paths and labels of past social engineering attacks. They use the proximity of file names, the proximity of file sizes, the proximity of URL structures, and whether or not domains and hosts are shared as features.

There is also an approach to analyze the Web pages that users visit. When analyzing Web pages, it is often a problem that the content is obfuscated, but paper [39] proposed to overcome

that problem by analyzing the JavaScript code at the bytecode level. In this research, bytecode is generated using Google's V8 engine and vectorized using word2vec. Using the vector as an input and learning the malicious code of JavaScript with LSTM, the classification of JavaScript code is realized.

Analysis of User Access Tendency to Malicious Sites Although it is positioned as an extension of the research as mentioned above on malicious URL analysis, there are also studies in which the aim is not to detect malicious URLs but to analyze the user's characteristics. For example, the features related to each user's Web browsing record are extracted from the communication log on the mobile phone network. Based on this, a method for predicting whether the user may visit a malicious URL in the near future has been reported [40]. From the previous month's information, they use random forest and DNN to predict whether the user will access the malicious URL in the next month and whether the user will access the malicious URL in the future in one session.

Analysis of Encrypted Communication In the technical field of analyzing communication, many analyzes of encrypted communication have been reported. When implementing security measures, it is sometimes necessary to analyze encrypted communications and binaries. Various studies have been reported that utilize machine learning to identify the contents of encrypted communication. FlowPrint [41] proposed a technology to extract an app's fingerprint by analyzing the encrypted communication emitted from the Android and iOS apps. Unlike the conventional method, the ability to extract fingerprints using unsupervised learning is a significant contribution. Since various new apps and malware are continually appearing in the security area, scalability is often an issue when using supervised learning. Therefore, it is highly evaluated that highly accurate discrimination can be realized by using unsupervised learning.

2.3.4 Analyzing Binary and Source Code

In recent years, malware has been used for criminal activity affecting individuals, businesses, and even governments. In addition, malware is diverse and evolving, and new types and variants of malware that support multiple architectures and operating system platforms appear one after another. Therefore, due to the rapid increase in the number of malware specimens, malware analysis, a process for investigating the functions and potential effects of specimens, has become an essential issue in the security field. In the research area of binary analysis, technologies for detecting malware in applications and technologies for analyzing their functions have been studied. Here, we will give an overview of the malware classification technology.

Malware analysis technology that analyzes malware functions and their potential effects are essential in the security field. Malware analysis is broadly divided into the dynamic analysis, which executes and investigates malware on a sandbox, and the static analysis investigates without executing malware. Conventionally, high personal analysis skills have been required for these analyzes, but studies are underway to replace the analysis using machine learning.

Dynamic Analysis of Malware Initially, automation using machine learning was actively studied for dynamic analysis that does not require disassembly. Using logs (system calls, API calls, etc.) obtained by dynamic analysis as features, many malware samples can be detected efficiently and with high accuracy, and malware family classification and clustering can be achieved. In the past, signatures were used to detect and classify large numbers of malware samples automatically. However, with the introduction of machine learning, it has become possible to analyze large numbers of samples without using signatures.

Static Analysis of Malware In recent years, reports on static analysis using machine learning have become active. If we have the source code when we analyze the binary, we can analyze it, but there are only a few cases where we can get the source code. Therefore, before performing the analysis, binaries are usually disassembled or decompiled; the features are extracted from the obtained code, and the analysis is performed. For malware detection, there are technologies to judge whether it is close to malware by evaluating the similarity between

binaries; various features and indicators of the similarity are to be used. As feature values, various forms such as control flow graphs reconstructed from binary, instruction statements, number, and types of function arguments are used. They are vectorized using machine learning techniques such as deep learning. Then, the distance between the binaries is evaluated based on the vector. With these technologies, it is possible to detect new and subspecies of malware efficiently. By automatically creating malware signatures based on the results, further automation of malware analysis can be expected.

In paper [42], a control flow graph is created from multiple platforms' binaries, embedded in a vector-based on a deep neural network. The similarity between binaries is measured from the cosine similarity. As an extension study of [42], paper [43] pointed out that information such as the meaning and dependence of the instruction itself is missing; they treated basic blocks and functions as words with instructions and used them for vector embedding to obtain more accurate similarity results. Paper [44] proposed a method of embedding a binary one-instruction sentence in a vector, learning the number and type of arguments of a function using a recurrent neural network, and using it as a disassembly signature. Measuring malware binaries' similarity by machine learning, as described above, is useful in determining variants and new species. In addition, by automatically extracting functions and creating signatures, it is possible to know the functions of malware specimens efficiently. From the above, it can be expected to reduce the burden on malware analysts, reduce costs, and lower the hurdles for static analysis.

2.3.5 Analyzing Threat Intelligence

Threat intelligence is useful and essential information generated by analysts and others that contributes to security operations and is usually highly reliable. In recent years, there has been a movement to utilize the information collected as threat intelligence to analyze security incidents. Below, we introduce some technologies for generating and analyzing that intelligence.

Generating Threat Intelligence Security reports, personal blog articles, and incident information posted on SNS such as Twitter, which is regularly issued by each security vendor, are useful for analyzing security incidents within the organization. However, this information is written in natural language, and in most cases, it is written in its format or has its tagging. Even for security reports, there are many cases where the standards for labeling attached to the structure and documents differ depending on the publisher's security vendor. Therefore, when unstructured threat information is used for analysis, it is difficult to automatically match it with trace information indicating intrusion into a computer or automatically assign an appropriate label. Until now, security operators have collected useful information from security reports based on their experience. In recent years, technologies that utilize machine learning technology to extract intelligence from unstructured documents, label them and automatically generate reports in a standardized standard format have been studied. In that case, there are cases where machine learning and ontology, which is an abstract conceptual model, are used.

Annotating Threat Intelligence Studies have also been reported in which information is annotated to something similar to the technology for generating intelligence from natural language. This adds more detailed information about vulnerabilities and threats. The structured information published as a database includes the following: JVN ³, CVE ⁴, and NVD ⁵ databases that aggregate vulnerability information, and the knowledge base ATT&CK [45] developed and published by MITER ⁶. The vulnerability information database also contains a description that outlines the vulnerability. Many methods have been proposed to model cyberattacks' activity into a fixed phase, and the cyber kill chain proposed by Lockheed Martin is incredibly widely used ⁷. The cyber kill chain is a concept that breaks down attack activities represented by targeted attacks and advanced persistent threat (APT) attacks into seven phases. In the cyber kill chain model, an attacker is thought to take a specific step to progress through

³Japan Vulnerability Notes, <https://jvn.jp/>

⁴Common Vulnerabilities and Exposures, <https://cve.mitre.org/>

⁵National Vulnerability Database, <https://nvd.nist.gov/>

⁶<https://attack.mitre.org/>

⁷<https://www.lockheedmartin.com/en-us/capabilities/cyber/cyber-kill-chain.html>

each phase. It is possible to analyze the attacker's attack procedure after abstracting it. For example, when a security incident occurs in the organization, we can use it to analyze an attacker's behavior by matching the incident log in the organization with the cyber kill chain model.

Detection and Identification of Noise Information While there is a large amount of security information, there are cases where there is information that causes noise. Such information may exist in structured information, and studies have been reported to identify it. Such noise is often caused by human error, and it is possible to generate more valuable intelligence by identifying and correcting the cause of the noise. For example, technology has been reported that compares vulnerable software versions inferred from natural language information with the information described in intelligence and automatically detects inconsistencies. Fake news also exists as a malicious case of the above noise, and the technology to detect it has also been reported. The dataset used in security operations may contain information posted by malicious user accounts, and technologies for detecting dataset contamination are also being studied.

2.3.6 Preparing Dataset

So far, we have picked up multiple areas and introduced research on the utilization of machine learning technology in the cybersecurity area, but how to secure research datasets is essential in this field. Even if someone wants to carry out a study, machine learning cannot be applied unless constructing a dataset to be analyzed to carry out the study. Furthermore, to carry out highly competitive research and development, it is indispensable to give a high-quality and unique dataset and a reliable and convincing label.

Collecting Competitive Data Major search engines, Internet service providers, and social media service providers have vast amounts of data, and the availability of such data makes it possible to carry out highly competitive research. When such data is inaccessible, how to collect and construct data that can be differentiated from those data dramatically affects the quality of the final research results.

In cybersecurity, open-source intelligence (OSINT), which collects confidential and threat information from publicly available information sources, is an essential service for data analysis. In recent years, various OSINT services have been developed, and security analysts and researchers collect data from OSINT and obtain meaningful information for analysis. When evaluating research using machine learning techniques, reliable labeled data is needed. However, since OSINT is composed of publicly available information, its reliability is often questioned.

Moreover, it is even more challenging to claim the reliability of data collected from private sources. There are many other types of cybersecurity-related data, but few are publicly available and have reliable labeled datasets. In addition, there are many areas where trends change rapidly, so once a dataset is created, it may become obsolete. Finally, when data contains confidential information, crowdsourcing and natural language processing are often not available. Such problems are annoying to many researchers.

Preparing Labeled Data and Ground Truth When conducting supervised learning, labeled data is required, but labeling standards are not clear, and it is often difficult. In addition, ground truth is indispensable for the performance evaluation of machine learning models. For example, it is necessary to add label information to whether a binary is malware in malware detection technology. However, the criteria for what should be labeled as malware are not clear. Although various measures have been taken, a method of determining the label name by majority vote from multiple sources is proposed based on VirusTotal⁸ information that can collect the results of about 90 antivirus software [46]. In addition, although many cases of labeling by experts have been reported, experts in the context of the data to be handled are usually not experts in machine learning. Therefore, research on interfaces that enable non-machine learning experts to realize labeling efficiently and effectively is also reported [47].

⁸<https://www.virustotal.com/>

2.3.7 Direction of Future Research

Machine learning is beginning to realize the automation of security operations that could not be realized until now. In particular, there is great merit in automating what was unrealistic in processing time to be classified manually. Some operations can be realized to be more efficient by machine learning, such as detecting spam emails, exceptional communication on the intranet, and screening malware candidates. On the other hand, machine learning alone cannot solve all security issues. Looking at the security society presentations at this time, many presentations do not depend on machine learning. In addition, papers suggesting the limits of machine learning have been submitted.

We feel that the last few years have been a time of widespread exploration of machine learning's adaptability. From now on, we believe that it will be necessary to continue exploring the possibility and consider using machine learning technology for more in-depth analysis. In order to analyze more deeply, it is indispensable to make an effort to actively overcome the barriers of the area in the research industry where the area tends to be subdivided. It is then necessary to steadily grow the technology by absorbing the know-how in the field of security operations more deeply and reproducing it by machine learning. Attackers do not wait for the development of technology, so implementing and integrating the development of conventional heuristic-based technology and the development of security technology using machine learning technology in parallel, a sense of speed we want to do some research and development.

Chapter 3

Dark-GLASSO: Real-time Detection of Malware Activities Using Graphical Lasso

3.1 Introduction

Network security threats (cyberthreats) by malware such as worms, bots, and automated exploit tools send Internet-wide scans to a large number of unspecified hosts on the Internet and attempt to exploit vulnerabilities, attack and intrude into targeted systems. When the malware succeeds in infecting vulnerable hosts, the infected hosts will search for next vulnerable targets one after another, and the infection of the malware will spread worldwide.

To minimize the impact of these threats, it is important to understand and determine the behaviors of malware on the Internet quickly and precisely. Many network intrusion detection systems (NIDS) and firewalls have already performed access control functions that are signature-based, but these cannot cope with zero-day attacks and brand-new malware variants. Network security operators or analysts may also monitor the cyberthreats using heuristic and rule-based systems, but these are costly, and human errors may happen. Accordingly, a method that can catch cyberthreats automatically, quickly and precisely is needed.

What data should we use to achieve this goal? Cyberthreats, including fast-moving worms, distributed reflection denial of service (DRDoS) attacks, routing exploits, and network scan-

ning and probing, are globally scoped, irrespective of geographic or topological boundaries. Passively monitoring unused or dark address space (a darknet) is one promising method of investigating these threats [9, 21], because there are no legitimate hosts there. Other than misconfiguration, packets destined for the darknet are almost always malicious [48]. In general, it is suitable for monitoring explosions, not small events [49]. In short, to understand a trend of global cyberthreats, darknet data is more useful than other observations. However, it does not mean that it is easy to capture cyberthreats accurately. Here, the difficulty lies in how we distinguish the ill-intentioned traffic from others (misconfiguration).

We consider the following approach to resolve the above difficulty. There is no synchronization or cooperation between the packet transmission time patterns of unrelated source hosts. On the other hand, a single host tends to perform similar behavior at each darknet address [9], hence we assume that when source hosts infected with similar malware (such as IoT malware), the source hosts are synchronizing with each other in the darknet. Also, the infected hosts that form a botnet (such as IoT botnet) have the characteristic of acting synchronously when they receive commands from a command and control (C&C) server [11]. Moreover, we consider that if synchronization among source hosts in a time slot is abnormally high compared to other time slots, a cyberthreat event has occurred during that time slot.

However, it is difficult for humans to check synchronization among source hosts manually. To handle this issue, we propose a system called *Dark-GLASSO*¹, which estimates a synchronization of the source hosts in real time by using a sparse structure learning algorithm called the *graphical lasso* [15]. It also announces alerts in real time when the estimated synchronization changes significantly. It can detect a group of related campaigns by estimating synchronization among many pairs of source hosts for each time slot and by finding outliers in synchronization among different time slots. This method is resistant to noise; packets that arrive accidentally in a darknet space due to misconfigurations will not show synchronization. Moreover, it is able to ignore events with weak synchronization due to an effect of ℓ_1 regularization penalty term of the graphical lasso algorithm.

In the evaluation, we measure the performance of *Dark-GLASSO* with a demonstration

¹The engine is named after the library “*glasso*” of R [50].

experiment using in-the-wild darknet traffic to show its feasibility. This experiment, whose purpose is detection of anomalous synchronization from darknet traffic, has broadly identified three types: 1) large-scale cyberthreats by malware (for example, IoT malware, worms, bots, and exploit tools), 2) survey scans by universities or other organizations, and 3) sporadically-focused traffic considered to be misconfiguration traffic. In this study, we evaluate and analyze the cyberthreats from malware. Then, a darknet analysis expert gathers a cyberthreat list with relatively large host-scales infection over a certain period for each destination port. This is the most important task for operators to understand cyberthreats. We evaluate the detection results of *Dark-GLASSO* using the list as the ground truth. As a result, we confirm that *Dark-GLASSO* can detect cyberthreats with somewhat massive host-scale infection with high accuracy, and which shows its effectiveness and usefulness. In other words, *Dark-GLASSO* can successfully identify global cyberthreats in real time in the darknet that could lead to serious security incidents.

This study offers the following contributions:

1. We propose *Dark-GLASSO*, as shown in Fig. 3.3, which statistically estimates a synchronization between the source hosts from traffic data and detects anomalies. (Discussed in Section 3.4)
2. We perform a demonstration experiment of *Dark-GLASSO* in real time using in-the-wild darknet traffic and evaluate the detection performance of cyberthreats using the prepared ground truth. We also make a comparison with a conventional method (ChangeFinder). As a result, we achieve a detection accuracy of 97.1%, which is better than the conventional method and describe the details of the detected cyberthreats. (Discussed in Section 3.6)
3. Finally, as a discussion and a consideration, we investigate the adverse effect of preprocessing on the accuracy and describe the technical limitations of our method. (Discussed in Section 3.7)

This study is the first practical trial that detects global cyberthreats in real time and early

stage based on in-the-wild darknet traffic regardless of already known or new types or variants of threats. In addition, related works on darknet analysis discussed in Section 3.3 only described case studies using their method, but this work is the first study where we have quantitatively evaluated the detection accuracy of cyberthreats during a certain period. We believe that this is an innovative achievement that can significantly reduce oversights, human-errors, and burdens on security operators and incident response teams. Moreover, operators can utilize the information about the source hosts that are detected as synchronized by *Dark-GLASSO*. In addition, we publish the datasets used for demonstration experiment in this chapter on the web².

3.2 Preliminaries

Here, we present an overview of essential elements; the graphical Gaussian model and the graphical lasso algorithm.

3.2.1 Graphical Gaussian Model

A graphical Gaussian model (GGM) is a probabilistic model for which a graph expresses the dependence structure between random variables. In the GGM, it is assumed that a N -dimensional random variable sequence $\mathbf{x} = (x_1, x_2, \dots, x_N)^T \in \mathbb{R}^N$ follows the N -dimensional multivariate Gaussian distribution. The multivariate Gaussian distribution can be expressed as

$$\mathcal{N}(\mathbf{x}|\boldsymbol{\mu}, \boldsymbol{\Sigma}) = \frac{\det(\boldsymbol{\Sigma}^{-1})^{1/2}}{(2\pi)^{N/2}} \exp\left(-\frac{1}{2}(\mathbf{x} - \boldsymbol{\mu})^T \boldsymbol{\Sigma}^{-1}(\mathbf{x} - \boldsymbol{\mu})\right).$$

Where $\boldsymbol{\mu} \in \mathbb{R}^N$ is the mean of $\mathbf{x}$, $\boldsymbol{\Sigma} \in \mathbb{R}^{N \times N}$ is a covariance matrix and $\boldsymbol{\Sigma}^{-1} \in \mathbb{R}^{N \times N}$ is a precision matrix. Both a covariance matrix $\boldsymbol{\Sigma}$ and a precision matrix $\boldsymbol{\Sigma}^{-1}$ are symmetric positive definite matrix. Also, the determinant of a matrix $\mathbf{A}$, written $\det(\mathbf{A})$.

To measure the dependency structure between random variables, there is a method of obtaining a precision matrix $\boldsymbol{\Sigma}^{-1}$ (the inverse of the covariance matrix $\boldsymbol{\Sigma}$) from which the conditional independence of a pair of random variables is able to be estimated [51, 52]. If and only

²<https://csdataset.nict.go.jp/darknet>

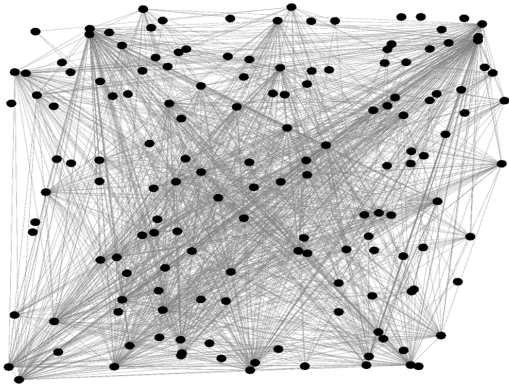


Figure 3.1: The example of GGM graph 1

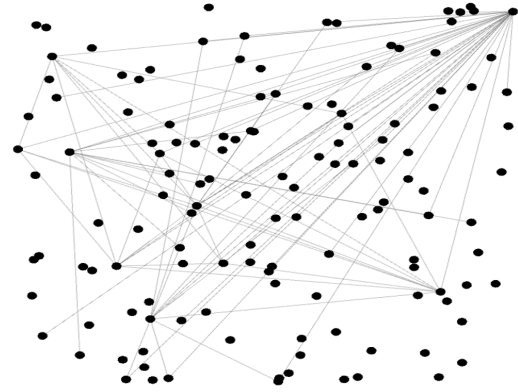


Figure 3.2: The example of GGM graph 2

if $(\Sigma^{-1})_{ij} = 0$, then x_i and x_j are independent, conditioned on all the other variables. In other words, it can be expressed as follows.

$$\Sigma_{ij}^{-1} = 0 \Leftrightarrow x_i \perp x_j | \mathcal{X} \setminus \{x_i, x_j\}.$$

For the multivariate Gaussian distribution, the conditional independence implies no correlation.

The definition of the graph in the GGM using the precision matrix $\Sigma^{-1} \in \mathbb{R}^{N \times N}$ of a sequence of random variables following an N -dimensional multivariate Gaussian distribution is as follows—an undirected graph $G = \{V, E\}$ in the GGM is represented by node set $V = \{x_1, \dots, x_N\}$ of N random variables and edge set $E = \{(i, j) | (\hat{\Sigma}^{-1})_{ij} \neq 0\}$. In other words, graph G shows the conditional independence of all pairs of random variables. Each node corresponds to a random variable, and the existence of an edge between nodes indicates a presence or absence of a cooperative relationship between nodes (variables). Here, although all of the diagonal components of an actual precision matrix Σ^{-1} are positive real numbers, let all of the diagonal components be 0 for graph representations. In addition, in the analysis of darknets data, if a variable of the number of packets from a source host is made to correspond to a node, an undirected graph of the precision matrix is a graph showing cooperative relationships between the source hosts. Figure 3.1 and 3.2 are examples of undirected graphs obtained from applying the GGM to actual darknet data.

3.2.2 Graphical Lasso

The most natural way to estimate the precision matrix Σ^{-1} is to do a maximum likelihood estimation, and it had been studied a lot in the past [53, 54]. Under an assumption of the multivariate Gaussian distribution, the maximum-likelihood estimate of a precision matrix Σ^{-1} is the inverse of the sample covariance matrix S . We expect the precision matrix Σ^{-1} to be a sparse matrix such that for the variable pairs with essential dependencies, the corresponding elements are nonzero, and for the weakly related variable pairs, the corresponding elements are zero. In general, however, the elements of the inverse of the sample covariance matrix S cannot be strictly zero. In addition, if strongly correlated variables exist, there is a problem that the inverse matrix of the sample covariance matrix S^{-1} causes a rank deficient, and the inverse matrix does not even exist.

Accordingly, the graphical lasso, which is a sparse structure learning algorithm, optimizes the precision matrix by solving a penalized maximum-likelihood equation with ℓ_1 regularization term $r \sum_{ij} |(\Sigma^{-1})_{ij}|$ [15]. The flow of a sparse structure learning by the graphical lasso is shown below. First, we obtain a sample covariance matrix S from data. Then, we substitute the sample covariance matrix S into the following log-likelihood function

$$\ln \prod_{m=1}^M N(\mathbf{x}^{(m)} | \mu, \Sigma) = \text{const.} + \frac{M}{2} \left\{ \ln \det(\Sigma^{-1}) - \text{Tr}(S \Sigma^{-1}) \right\}.$$

Finally, we calculate the penalized maximize log-likelihood function with a ℓ_1 -regularization term in the form

$$\arg \max_{\Sigma^{-1} > 0} \left(\ln \det(\Sigma^{-1}) - \text{Tr}(S \Sigma^{-1}) - r \|\Sigma^{-1}\|_1 \right), \quad (3.1)$$

where r is a positive real number, $\|\Sigma^{-1}\|_1$ is a ℓ_1 -norm of Σ^{-1} defined as

$$\|\Sigma^{-1}\|_1 = \sum_{ij} |(\Sigma^{-1})_{ij}|.$$

The graphical lasso is an algorithm to solve the equation (3.1) at high speed and high accuracy using the block coordinate descent method [55]. The graphical lasso is implemented as an R language package “glasso.” The input arguments of the graphical lasso algorithm are the sample covariance matrix S and the ℓ_1 regularization coefficient r (≥ 0). Here, r is

a hyperparameter for deciding how much dependency is regarded as noise-derived, and it is possible to adjust the sparsity of the precision matrix to be estimated [56]. From the above, the GGM graph using the precision matrix $\hat{\Sigma}^{-1}$ estimated by the graphical lasso algorithm expresses the more essential dependencies of all variable pairs than the maximum-likelihood estimate of the precision matrix.

3.3 Related Work

In this section, we review previous research about botnet detection, darknet data analysis, including our achievement so far. The BotSniffer [10] and the BotMiner [29] adopt C&C traffic detection method based on the spatial-temporal correlation approach, which is a similar approach to this study. However, their methods require traffic responses that include the payload of particular protocols (IRC, HTTP, P2P). On the other hand, since the darknet does not return a response, only the first packet can be observed in all protocols. In other words, *Dark-GLASSO* estimates a synchronization between the source hosts with a small amount of information that does not limit the port, protocol and does not include the payload. Furthermore, the scale of the practiced observation sensors is thousands of times different, and the scale of observed scans is overwhelmingly large. *Dark-GLASSO* tackles the problem that scale, scope, and amount of information are entirely different from the BotSniffer and the BotMiner. Their method cannot be applied to darknet analysis of this scale.

The darknet has been attracting attention in the network security field since 2000, and many researchers have been actively studying its development, analysis, and visualization [57]. In [9, 21, 48, 49], the basics of the darknet were discussed, including its various configurations, deployment technology, and sensor placement technology, and the effectiveness of the darknet was clarified. In general, darknet data is composed of scanning, backscatter, and misconfiguration traffic. Research on profiling, filtering, and classification of each type of traffic has been actively conducted as a darknet analysis. Dainotti et al. developed and evaluated a methodology to remove spoofed traffic from both a darknet and a live network and contributed to supporting census-like analyses of IP address space utilization [58]. Durumeric et al. ana-

lyzed the large-scale darknet to investigate scanning activities and identify patterns in sizable horizontal scanning operations [59]. Fachkha et al. devised inference and characterization modules to extract and analyze cyber-physical systems' (CPS) probing activities toward sufficient CPS protocols by correlating and analyzing the various dimensions of a large amount of darknet data [60]. Most of the analysis research using a darknet is measurement analysis that investigates something in detail. Many studies not only use a darknet but also honeypots and other trap-based monitoring systems at the same time.

According to the paper [57], filtering misconfiguration packets is still not thoroughly investigated and remains as a gray area that requires more attention from the research community. We believe that our method can avoid misconfiguration packets by detecting anomalies with the synchronization of hosts. Furthermore, our method is unique in that it can uniformly detect cyberthreats in real time in suspicious scans that reach the darknet. There are several methods to detect change points of traffic transition on the darknet and detect anomalies [8, 25, 61]. In Network Incident analysis Center for Tactical Emergency Response (NICTER) project [8] conducted by NICT, they used a ChangeFinder algorithm [7] to detect a rapid change in darknet traffic in real time with a low computational cost. ChangeFinder algorithm adopts the Sequential Discounting AutoRegressive (SDAR) forgetting learning algorithm that improves the autoregressive model so that it can be learned online, calculates only new time-series data, and reduces the influence of past data.

However, these change point detection methods have many limitations in actual operation. For example, when analyzing all TCP destination port numbers separately (2^{16}), an enormous amount of alerts can be obtained, and multiple high-performance servers are required. As an alternative, it is conceivable to aggregate and analyze for each range of the destination port number, but it is necessary to adjust the parameters according to the time when the traffic volume changes dramatically or moderately. In the last few years, the amount of traffic reaching a darknet has increased sharply due to the sophistication of scanning tools such as Masscan and Zmap, and the spread of malware targeting IoT devices such as Mirai. For these reasons, a large amount of constant and complicated traffic has arrived, and technical limitations such that no change point can be found have been an issue in these studies [8, 25, 61]. We have started

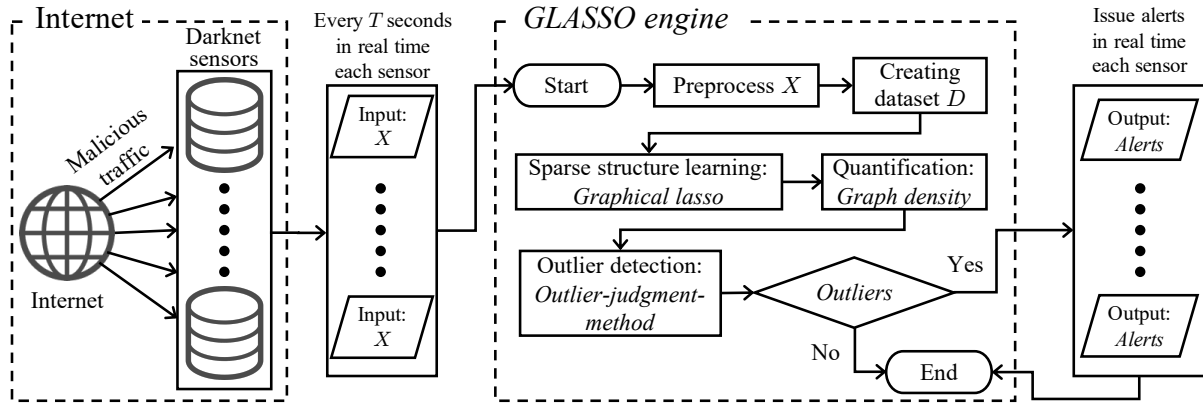


Figure 3.3: The overview of *Dark-GLASSO*

to study *Dark-GLASSO*, which is a method to detect cyberthreats using an approach that can avoid the problem of change-point detection methods, as mentioned above. *Dark-GLASSO* detects anomalies based on more abundant information, such as the synchronization of source hosts, than the change point detection method. In addition, *Dark-GLASSO* performs analysis without separating the traffic of all destination ports, so it is designed to achieve its purpose even in constant and complicated traffic where there is no change point.

3.4 System Overview: *Dark-GLASSO*

In this section, we describe *Dark-GLASSO* that estimates synchronization among source hosts reaching a darknet in real time and automatically, and determines the time slots where there is an anomaly synchronization. By capturing an anomaly in the synchronized time slots, *Dark-GLASSO* can automatically detect not only known cyberthreats but also zero-day cyberthreats quickly and accurately.

Figure 3.3 shows an overview of *Dark-GLASSO*, and Algorithm 2 shows the pseudocode of *Dark-GLASSO*. First, we prepare T seconds of traffic data X every darknet sensor. Here, *Dark-GLASSO* can analyze regardless of a protocol, but this study deals only with the TCP. Also, since packets other than the SYN flag on the darknet are not considered as network

scanning, we only capture TCP-SYN packets. We transfer the traffic data X to *Dark-GLASSO*, preprocess traffic data according to the situation ³, and then process it using the following four steps and issue an alert: creating the dataset, sparse structure learning, quantification, and outlier detection. The four steps are described in the order below, and finally, the procedure of online processing for operating *Dark-GLASSO* in real time is introduced.

3.4.1 Creating Dataset D

Dark-GLASSO counts the number of packets from traffic data X for every source host at regular intervals and creates a dataset D . We then treat the source host as one variable and apply a graphical Gaussian model (hereinafter, “GGM”) in Section 3.2.1 to estimate dependencies among source hosts. The procedure for creating a dataset is shown concretely. We assume that there are N source hosts in the preprocessed traffic data X . We also set the number of samples to M , and the counting (sampling) interval becomes T/M (sec.). We then convert from the X to a dataset

$$D = [D_{mn}] \in \mathbb{R}^{M \times N}, D_{mn} := \log(x_n^{(m)}), \mathbf{x}^{(m)} \in \mathbb{N}_0^N.$$

Here, $\mathbf{x}^{(m)}$ means N -dimensional variables of the number of samples M , and $x_n^{(m)}$ represents the number of packets at the m -th point of the n -th source host. In addition, $\mathbb{N}_0 = \{0, 1, 2, \dots\}$.

Since $x^{(m)}$'s distribution is far from Gaussian, we transform it by adapting a logarithmic function. This is a conventional method to transform a positive random variable concentrating zero so that it approximates a Gaussian random variable. In particular, if the considered variable is an exponential function of a Gaussian random variable, then its logarithm is Gaussian. In our case, it does not hold, but the graphical lasso algorithm works well in practice. Since log transformations cannot be performed when $x_n^{(m)} = 0$; we change a value 0 to a small real number 0.1. Next, we describe the graphical lasso (Section 3.2.2), one of a sparse structure learning algorithm, for estimating dependencies and synchronizations among variables from the dataset D .

³Detailed settings for preprocessing are described in Section 3.6.1.

3.4.2 Sparse Structure Learning: Graphical Lasso

As described in Section 3.2.1, the GGM that can show dependencies and synchronizations among variables requires a precision matrix that shows conditional independence among all variables. The graphical lasso algorithm described in Section 3.2.2 for estimating the precision matrix is a well-known sparse structure learning algorithm.

We calculate a sample covariance matrix S from the dataset D and substitute some positive real numbers $r \in \mathbb{R}$ ($= \{r_1, r_2, \dots, r_s\} \subset [0, \infty)$) and S into the graphical lasso algorithm to obtain a sparsely estimated precision matrix $(\hat{\Sigma}^{-1})^{(r)}$. Here, the adjustment of r is important, when the larger the value of r , the precision matrix is estimated more sparsely. In the next section, we introduce how to quantify synchronizations among variables from the estimated precision matrix.

3.4.3 Quantification: Graph Density

We can create the GGM undirected graph G from the estimated precision matrix. This graph $G = \{V, E\}$ has more edges if there are many non-zero elements. Also, the presence or absence of an edge indicates the presence or absence of synchronization between variables. However, since it is difficult to handle the graph structure as it is, we define a scalar value representing the overall degree of synchronizations among all hosts, a graph density $d^{(r)} = |E|/N(N-1)$. The graph density, also called a sparsity, is the ratio of the actual number of edges to the number of edges in a complete graph. By using this graph density value, it is possible to understand the degree of synchronizations among source hosts in the time slot of data X . Since the graph density value is closer to 1, it means that there is more synchronization among source hosts in that time slot.

3.4.4 Outlier Detection

In this section, we consider outlier detection to catch time slots in which the synchronization among source hosts is abnormally high compared to other time slots. First, we prepare

Algorithm 1 *outlier-judgment-method*

Input: $\mathbf{d}^{(r)} \in \mathbb{R}^K$, K , θ

Output: *outliers* or none

```

1:  $i, j \leftarrow 0$ 
2: while TRUE do
3:    $i \leftarrow i + 1$ 
4:    $\mathbf{d}_{(i)} \leftarrow \text{order}(\mathbf{d}^{(r)})[i : K]$ 
5:    $\sigma_{(i)}^2 \leftarrow \text{var}(\mathbf{d}_{(i)})$ 
6:   if  $\sigma_{(i+1)}^2 / \sigma_{(i)}^2 < \theta$  then
7:      $j \leftarrow j + 1$ 
8:   else if  $i = 1$  then
9:     break
10:  else
11:     $\text{outliers} \leftarrow \text{order}(\mathbf{d}^{(r)})[1 : j]$ 
12:    return outliers
13:  end if
14: end while

```

K graph density values $\mathbf{d}^{(r)}$ for each parameter r . Next, we check the extent to which the largest element occupies a total sample variance in $\mathbf{d}^{(r)}$, and when the largest element exceeds a criterion of an outlier judgment formula, it is judged as an outlier. The pseudocode of an *outlier-judgment-method* is shown in Algorithm 1. Here, $\sigma_{(i+1)}^2 / \sigma_{(i)}^2 < \theta$ is the outlier judgment formula, and θ ($0 \leq \theta \leq 1$) is a threshold value. In addition, the $\text{order}()$ function returns a permutation that rearranges its first argument into descending order, and the $\text{var}()$ function returns a sample variance.

Up until now, we have been able to calculate a graph density value from traffic data X and determine outliers. In the next section, we describe the steps for calculating the above steps sequentially and issuing alerts in real time.

Algorithm 2 *Dark-GLASSO* with Online Processing

Input: $X, \beta, M, r \in \mathbb{R} (= \{r_1, r_2, \dots, r_s\}), \mathbf{d}^{(r)}, K, \theta$

Output: alerts or none (per while-loop)

```

1: while  $X$  is updated newly do
2:   preprocess  $X$ ; and create  $D$  from  $X$ ; and compute  $S$  from  $D$ 
3:   for  $r$  in  $R$  do
4:     compute  $(\hat{\Sigma}^{-1})^{(r)}$  using the glasso ( $S, r$ )
5:     compute  $d^{(r)}$  from  $(\hat{\Sigma}^{-1})^{(r)}$ 
6:     add  $d^{(r)}$  to  $\mathbf{d}^{(r)}$ 
7:     if  $\text{length}(\mathbf{d}^{(r)}) = K$  then
8:       call outlier-judgment-method ( $\mathbf{d}^{(r)}, K, \theta$ )
9:       if there are outliers then
10:        collect alert information from outliers
11:        return alerts
12:        delete all graph densities at the same time slots as the outliers from  $\mathbf{d}^{(r)}$ 
13:      else
14:        delete the oldest value from  $\mathbf{d}^{(r)}$ 
15:      end if
16:    end if
17:  end for
18: end while

```

3.4.5 Procedure for Online Processing

The pseudocode for *Dark-GLASSO* with online processing is described in Algorithm 2. Here, the $\text{length}()$ function obtains the length of the vectors. We execute the process each time when new traffic data X is updated. Next, we obtain a graph density value $d^{(r)}$ for each r . Thereafter, $d^{(r)}$ is appended to an arrangement of the past graph density value $\mathbf{d}^{(r)}$, and if the length of $\mathbf{d}^{(r)}$ equals K , an outlier detection is carried out.

Datasets judged as *outliers* are output as an alert and are removed from graph densities in

the same time slots as the *outliers* from $\mathbf{d}^{(r)}$ so that they will not be referred to in subsequent outlier detection. In addition, if no dataset is determined as an outlier, the oldest value is removed from $\mathbf{d}^{(r)}$. When the time has passed and a new X is updated, by repeating the above steps, it is possible to process *Dark-GLASSO* sequentially while maintaining the number of datasets used for outlier detection. From the above, *Dark-GLASSO* makes it possible to perform online processing and detect time slots in which there is abnormally high synchronization among source hosts in real time and issue an alert.

3.5 Parameter Tuning

The *Dark-GLASSO* has five important parameters, and we introduce a heuristic way to set these five parameters.

3.5.1 Length T of Time Slot

The length of time slot T (*sec.*) means the length of the entire observation time series of the data used for one model learning. First of all, we consider upper bounds of T . The computational cost of the *Dark-GLASSO* depends significantly on the number of source hosts N . As T becomes longer, the number of source hosts generally also increases, the processing time increases polynomially in N , and real-time processing becomes difficult. Next, we consider lower bounds. It is sufficient if we can make in-depth observations such that all of one campaign of a network scan is observed in T seconds. From our experience, one campaign of many network scans was observed at the maximum observation scale of the darknet sensor used in the *Dark-GLASSO* ends within 5 minutes (subnet /17). Accordingly, we set it to $T = 600$ (*sec.*), which can be properly processed in real time, and make in-depth observations.

3.5.2 The Number of Time Series Samples M

The symbol M denotes the number of time series samples used for one model learning, which is obtained by dividing the time slot length T into M sections. In other words, it has

a meaning of adjusting the length of the sampling interval T/M (sec.). Here, when T/M is small, the statistical model to denote the number of packets in the time slot is complex and powerful to describe the data. In contrast, when T/M is large, the model is too simple to describe the data properly. It means that there are certain appropriate values of M . We could solve this issue as a problem of statistical model selection. However, we solve the problem in another way. Note that the regularization coefficient r that is discussed in the next section has the similar function as M . That is, unless M is set to an extreme number, it can be covered with the regularization coefficient r . Based on our experience, if the sampling interval T/M is about 50 seconds, we can measure a sufficient time series of the number of packets, so we set $M = 12$.

3.5.3 Regularization Coefficient r

The regularization coefficient r (≥ 0) is an input parameter used in the graphical lasso algorithm. It is a hyperparameter to determine how much dependence is considered and adjusts the sparsity of the estimate of precision matrix Σ^{-1} . In other words, by tuning r , it is possible to scrape-down weakly related dependencies and reduce the number of false positives of dependency estimation. Based on this perspective, we see that r and M have similar roles. As a feature of r , $d^{(r)}$ generally becomes closer to 1 as r becomes closer to 0, and $d^{(r)}$ becomes closer to 0 as r becomes larger. In addition, as r is increased, the number of zero elements of a precision matrix estimated by the graphical lasso algorithm increases, so that the calculation time is shortened. Finally, r can be tried multiple times with R ($= \{r_1, r_2, \dots, r_s\} \subset [0, \infty)$), but the processing time of the *Dark-GLASSO* increases by the number of trials.

We set the *Dark-GLASSO*'s parameter as $T = 600$ (sec.), $M = 12$ and examine which value of r is appropriate. First, even when we try with varying values from six decimal places to two decimal places, the graph density value does not change significantly. For example, our experience shows that time slot datasets, which are alerted with $r = 0.55$, are judged alert by $r = 0.5$ or 0.6 in all cases. From this, we discover that it is sufficient to try while changing the value to one decimal place. In addition, there are many zero matrices from $r \geq 1$, which often

result in $d^{(r)} = 0$. The processing time also takes longer for $r < 0.4$, where a sufficiently sparse precision matrix is not estimated, and outliers do not emerge in most cases. From the above, we finally decided to set $r \in \mathbb{R}$ ($= \{0.4, 0.5, 0.6, 0.7, 0.8, 0.9\}$).

3.5.4 The Number of Graph Densities K and Threshold θ for Alert Judgment

The number of graph density values K and the threshold θ are used for alert judgment. In general, the precision of outlier detection improves as the number of data increases. However, in the case of time-series data that changes rapidly depending on the time, the precision may decrease as the number of data increases. That is, it is necessary to select K appropriately, as well as the relationship between M and r , unless K is set to an extreme number, it can be covered with θ . The θ ($0 \leq \theta \leq 1$) is the threshold of the outlier judgment formula $\sigma_{(i+1)}^2 / \sigma_{(i)}^2 < \theta$. As θ approaches 1, the outlier is judged loosely, and as it is closer to 0, the outlier is judged strictly. Based on our experience, we determined to use three days' data for outlier detection ($K = 432$), and θ is now set to 0.98 after trial and error.

3.6 Performance Evaluation

The alert issued by *Dark-GLASSO* is when the time patterns of packets sent from multiple source hosts for some reason are strongly synchronized. What happens when the source hosts observed in the darknet appear to be synchronized? Among them are cyberthreats that launched large-scale attacks/scanning. In this section, we give an overview of the demonstration experiment we performed, analyze the alerts detected by *Dark-GLASSO*, and evaluate a quantitative detection accuracy of cyberthreats. Here, the conventional method, ChangeFinder [7], is also tried and evaluated comparatively with our method.

3.6.1 Demonstration Experiment

In this section, we present the process of demonstrating the feasibility of *Dark-GLASSO*. We implemented a prototype of *Dark-GLASSO* using R language and a Linux shell running on

two machines, a 16 GB of memory and a 3.3GHz Intel-Xeon-E3-1230. We used eight different darknet sensors in parallel and operated *Dark-GLASSO* for one month in October 2018 in real time for each sensor. The input parameters of *Dark-GLASSO* used for the operation were set to $T = 600$ (sec.), $M = 12$, $K = 432$, $\theta = 0.98$, $r \in \mathbb{R}$ ($= \{0.4, 0.5, 0.6, 0.7, 0.8, 0.9\}$); details of this parameter selection can be found in the paper [13].

Next, we describe the preprocessing of darknet traffic performed in this demonstration experiment. In order to stabilize the performance of *Dark-GLASSO*, we filtered out potentially adverse packets to the engine from traffic data X in the preprocessing stage. First, we excluded destination TCP ports that constantly received a large number of packets and source hosts for a long time. Then, we excluded TCP ports 22, 23, 80, 81, 445, 1433, 2323, 3389, 5555, 8080, and 52869. Second, we regularly excluded the packets addressed to the TCP ports that are intensively observed as alerts. If such packets, the first and second cases, are included in the model learning of *Dark-GLASSO*, they will massively increase the indicator of synchronization among the source hosts to be estimated, and there is concern that smaller but essential synchronization may be overlooked. Such uninteresting packets that have an adverse effect on model learning and that have already attracted much attention should be regarded as noise and should be excluded. These procedures should take place regularly.

Finally, in order to reduce the number of source hosts N and gain a global understanding of cyberthreats, we ignored the lower 16-bit of the source host's IPv4 addresses, which meant that the hosts with identical upper 16-bit of IPv4 address were regarded as one host. In addition, when the number of source hosts N exceeded the threshold β , β source hosts were selected at random from the input X in order to limit N . Because the complexity of the graphical lasso is $O(N^3)$, in order to run this engine in real time, it is necessary to limit N according to the hardware specifications. All of the preprocessing described here is not compulsory and can be adapted depending on the situation. However, limiting the number of source hosts to β can degrade the accuracy of the learning model, which we discuss in Section 3.7.1.

Table 3.1: The number of observed IP addresses (IPs), alerts, and unique ports for each darknet sensor

Sensor ID	#IPs (/Mask)	#Alerts	#Ports
A	29,182 (/17)	118	33
B	14,593 (/18)	195	43
C	4,098 (/20)	111	42
D	4,096 (/20)	336	40
E	8,188 (/19)	187	44
F	16,384 (/18)	112	49
G	2,044 (/21)	66	32
H	2,045 (/21)	168	36
<i>Dark-GLASSO (ALL)</i>	80,630	1,293	152

3.6.2 Analysis of Alert Result

Table 3.1 shows the number of observed IP addresses (IPs), alerts and the number of unique ports for each darknet sensor. The individual sensors are distributed and installed in several countries in the world, where “*Dark-GLASSO (ALL)*” means the aggregated sensor consisting of all the sensors. Since one time slot was 10 minutes in this experiment, there are 4,464 time slots in one month. Because it had been demonstrated with eight sensors, 35,712 time slots were calculated. Consequently, 1,293 time slots were obtained as alerts. An alert identifies a target TCP port and source IP addresses that are considered to have been used as cyberthreats in the time slots determined to be outliers by *Dark-GLASSO*. Here, the target TCP port is the destination TCP port in which the number of source hosts that sent packets to the port in that time slot is 20 or more, and the ratio is 10% or more. One alert has multiple target TCP ports and a list of source hosts for each target TCP port. Finally, from the results of all sensors, we had 1,293 alerts and 152 unique destination TCP port among alerts.

In this study, in order to facilitate analysis and evaluation, only the target TCP ports are investigated without considering time information of alerts. It means if there is at least one

alert for a correct target TCP port during the entire dataset, it will be counted as a true positive. The reason is explained in Section 3.7.2. As a result of the analysis, 152 target TCP ports are classified into the following three types.

1. *Cyberthreats*: network scans that indiscriminately attempt attacks and intrusions through vulnerable TCP ports in order for large-scale hosts already infected with malware to search for the next infection targets—e.g., IoT malware, bots, fast-moving worms, automated exploit tools, etc.
2. *Survey scans*: network scans that indiscriminately attempt surveys and academic research on TCP ports using multiple hosts by universities or organizations such as Shodan, Censys, etc. This is a network scanning, but not a cyberthreat.
3. *Sporadically-focused traffic*: a phenomenon in which packets are suddenly concentrated from multiple source hosts to one darknet destination IP address and a TCP port. Most destination TCP ports are 5-digit unknown ports. We consider it to misconfiguration traffic.

We are able to clearly divide all the target TCP ports into the above three types.

First, the sporadically-focused traffic can be easily identified by whether the destination IP address is concentrated at one point. Second, the survey scan type has a smaller host size than large-scale cyberthreats, and more than 50% of the hosts in the alert occupies known universities and organizations that conduct survey scans. In the case of cyberthreats, only about 20% of hosts perform survey scans included in alerts. Consequently, out of the 152 target TCP ports, 34 were cyberthreats, 2 were survey scans, and 116 were sporadically-focused traffic.

The survey scan type is classified by looking at how many known scan organizations are included in hostnames of source hosts which are estimated to be synchronized. Also, they sometimes release scanning details on the organization's web site. It is a heuristic method based on our experience. However, the alerts of the two ports that are considered to be survey scans were acquired only once for 53/TCP from sensor A and once for 11211/TCP from sensor

Table 3.2: Ground truth list of TCP ports used for cyberthreats observed on our darknets in October 2018

Cyberthreat Type	TCP Port
IoT Malware (16 TCP ports)	82,83,84,85,88,444,2480,5358,5984, 7547,8000,8010,8088,8443,8888,9000
Router Vulnerability (12 TCP ports)	21,25,110,443,5431,8001,8081, 8181,8291,23023,37215,65000
Other Vulnerability (7 TCP ports)	1701,2004,5379,5900,6379,7379,49152

F, which are large observation scales of the sensor. In those two alerts, the number of synchronized source hosts was 20, 21, and more than half of them showed the hostnames of known scan organizations. At that time, no cyberthreat was observed in these two ports. Probably, the number of hosts for which scans were observed to 53, 11211/TCP coincidentally increased in spaces where observation scales of the sensor are large. As a countermeasure, the number of source hosts for alert judgment is currently set to 20 for all sensors, but it can be avoided by setting it for each sensor observation scale.

3.6.3 Detection Performance

In this section, we quantitatively evaluate the performance of *Dark-GLASSO* in detecting cyberthreats and compare it with a conventional method, ChangeFinder. First, we prepared the ground truth for quantitative evaluation. Table 3.2 shows the list of TCP ports used for cyberthreats observed on our darknets in October 2018, as confirmed by a darknet analysis expert in our organization. This list was created in January 2020 by comprehensively examining information such as changes in traffic volume, SYN packet characteristics, threat intelligence services, vulnerability reports, security reports and data from honeypots. As a result, 35 TCP ports are classified into three types of cyberthreats: IoT malware, router vulnerabilities, and other vulnerabilities. Conventional darknet analysis studies only showed case studies, but this

Table 3.3: Evaluation results of the detection performance of *Dark-GLASSO* and ChangeFinder (confusion matrix)

Sensor ID	#Ports	#Noises	#TPs	#FPs	#FNs	#TNs
A	33	4	29	0	6	-
B	43	14	29	0	6	-
C	42	17	25	0	10	-
D	40	15	25	0	10	-
E	44	17	27	0	8	-
F	49	23	26	0	9	-
G	32	13	19	0	16	-
H	36	17	19	0	16	-
<i>Dark-GLASSO</i> (ALL)	152	118	34	0	1	-
ChangeFinder (ALL)	25	1	24	0	11	-

work is the first attempt to create and quantitatively evaluate a long-term ground truth. In order to improve the reliability of this ground truth, we provide details in Section 3.6.4. Note that zero-day threats may become apparent in the future, and more ports may be added to the list.

Next, we describe the setting of ChangeFinder in the comparative experiment. ChangeFinder is a change point detection method that runs in an online manner. In an environment equivalent to *Dark-GLASSO* demonstration experiment, ChangeFinder was applied to time-series data of the numbers of unique source hosts and the numbers of packets every 10 minutes with eight sensors. The parameters of ChangeFinder were set to 2 for the order of AR model, 0.005 for the forgetting parameter, 10 and 5 for the smoothing range in 2 steps, and 3 for the change point judgment threshold. For details such as the meaning of these parameters, see [7]. This setting was determined based on our experience of using ChangeFinder for many years at the NICTER [8]. As a result, we obtained 1,501 change points. We processed the obtained change points in the same way as the target TCP port acquisition manner in Section 3.6.2, and obtained 25 unique ports.

Table 3.4: Evaluation results of the detection performance of *Dark-GLASSO* and ChangeFinder (4 major metrics)

Sensor ID	Accuracy [%]	Precision [%]	Recall [%]	F-measure [%]
A	82.86	100.00	82.86	90.63
B	82.86	100.00	82.86	90.63
C	71.43	100.00	71.43	83.33
D	71.43	100.00	71.43	83.33
E	77.14	100.00	77.14	87.10
F	74.29	100.00	74.29	85.25
G	54.29	100.00	54.29	70.37
H	54.29	100.00	54.29	70.37
<i>Dark-GLASSO</i> (ALL)	97.14	100.00	97.14	98.55
ChangeFinder (ALL)	68.57	100.00	68.57	81.36

Tables 3.3 and 3.4 shows the evaluation results of the detection performance of *Dark-GLASSO* and ChangeFinder. The evaluation of *Dark-GLASSO* for each sensor and all the sensors are shown. The evaluation of ChangeFinder for all sensors is also shown. Here, #Ports is the number of detected target TCP ports, #Noises is the number of TCP ports by survey scan or sporadically-focused traffic, #TPs, #FPs, #FNs, and #TNs indicate the number of true positives, false positives, false negatives, and true negatives, respectively. Currently, we do not know how many true negatives there are, so we did not care about them. Therefore, the usual accuracy is $(\#TPs + \#TNs) / (\#TPs + \#FPs + \#FNs + \#TNs)$, but here we assume #TNs were zero. Precision, Recall, and F-measure are obtained using the usual evaluation indices as follows: $\#TPs / (\#TPs + \#FPs)$, $\#TPs / (\#TPs + \#FNs)$, and $2\#TPs / (2\#TPs + \#FPs + \#FNs)$. Consequently, *Dark-GLASSO* achieved 97.14% accuracy and 98.55% F-measure with 34 TPs, 0 FPs, and 1 FN. If we assume that we could not separate the survey scan and the cyberthreat in this experiment, #FPs will increase from zero to two, and the accuracy will become 91.89% from 97.14%. On the other hand, ChangeFinder achieved 68.57% accuracy

and 81.36% F-measure with 24 TPs, 0 FPs, and 11 FNs, showing a lower level of performance than *Dark-GLASSO*.

Below we summarize a tendency seen from the results. Since there were no false positives, all precision were 100%. ChangeFinder had little noise but missed 11 ports. *Dark-GLASSO* missed only TCP port 444, while ChangeFinder missed TCP ports 444, 1701, 2004, 2480, 5358, 5379, 5984, 6379, 7379, 7547, and 8010. Because TCP ports 2480 and 5358 are constant threats, ChangeFinder seems to have missed the change points. There are two reasons why ChangeFinder often missed the change points: 1) Looking at the other nine ports individually, spikes are clearly present, but in most cases, they are small; and 2) the smoothing function cannot respond to momentary events. Among them, 444/TCP, which *Dark-GLASSO* overlooked, had overwhelmingly few source hosts compared to other ports. Our darknet analysts used the NICTER's 300,000-scale darknet, but this time *Dark-GLASSO* used the 80,000-scale darknet, so our method seems to have missed the small event, 444/TCP. In contrast, there were events that analysts missed (human error), in which TCP ports 25 and 1701 were detected by *Dark-GLASSO* and added to the ground truth (Table 3.2).

3.6.4 Details of Detected Cyberthreat

We describe the details of the detected cyberthreats according to the three attack types (IoT malware, router vulnerabilities, and other vulnerabilities) in Table 3.2. We refer to security vendor articles related to threats on each TCP port. If there is no reference, we provide an explanation based on the observed features and the transition graph of the number of source hosts per one hour (Fig. 3.4 to 3.9) to improve the reliability.

IoT Malware: Threats from Mirai, Hajime, and HNS were observed on the 16 TCP ports of the IoT malware type in Table 3.2. First, Mirai has a feature where a sequence number denotes a destination IP address converted to decimal in SYN packets. Figures 3.4, 3.5, and 3.6 show the TCP ports where such Mirai features were monitored. The 10 TCP ports in Fig. 3.4 were regularly scanned from approximately 1,000 hosts, but on October 22, the number of source hosts rapidly increased to approximately 6,000 and gradually decreased. In

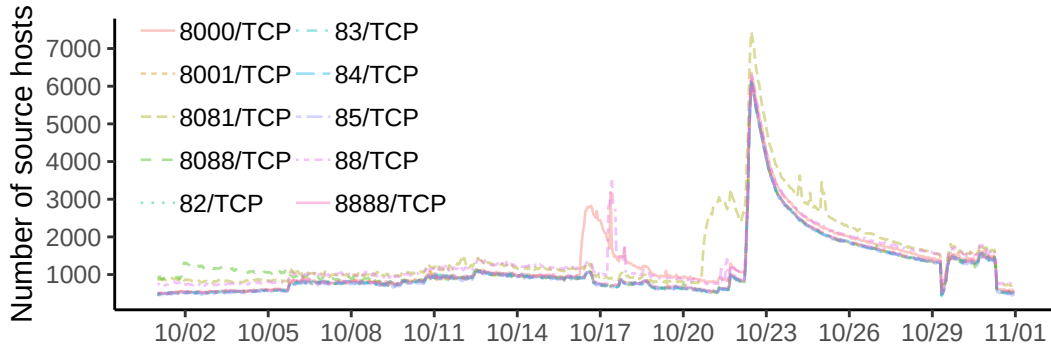


Figure 3.4: The number of source hosts per one hour in October 2018 at TCP ports where Mirai-related cyberthreats were observed. (sensor A, part I)

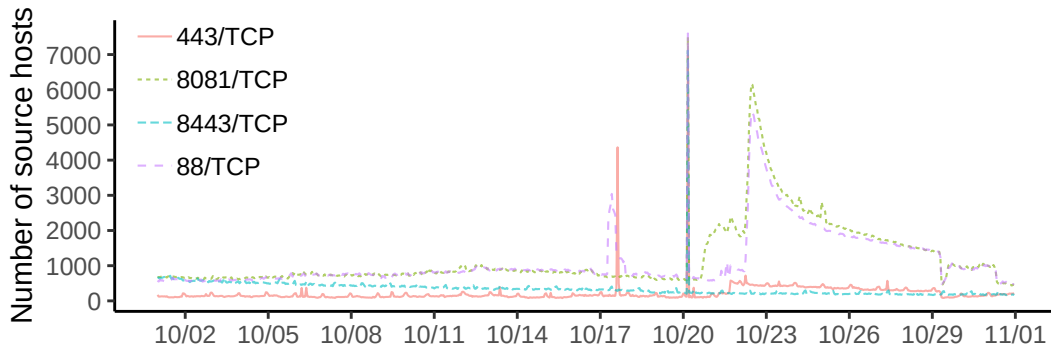


Figure 3.5: The number of source hosts per one hour in October 2018 at TCP ports where Mirai-related cyberthreats were observed. (sensor B, part II)

Fig. 3.5, we observed an abrupt scan from about 7,000 hosts to four TCP ports on October 20. It was different from the event in Fig. 3.4 that many hosts originated in China and had a fixed window size of 14100. Figure 3.5 shows the results for sensor B because this threat could not be observed with sensor A.

Figure 3.6 shows three TCP ports, which had relatively few hosts, but carried Mirai features. Regarding 7547/TCP, many hosts from Egypt targeted the vulnerability that was published on October 20, 2016 [62]. The event towards 444/TCP, mainly by hosts from Greece on October 23, was the only threat that *Dark-GLASSO* overlooked. As described in the pre-

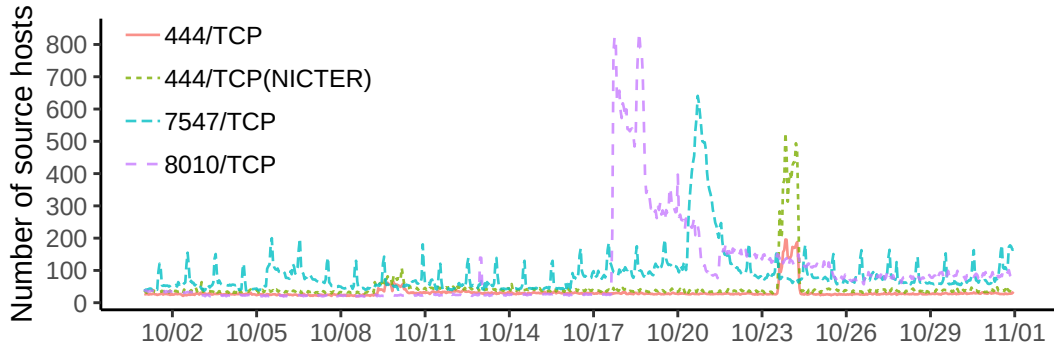


Figure 3.6: The number of source hosts per one hour in October 2018 at TCP ports where Mirai-related cyberthreats were observed. (sensor A and NICTER, part III)

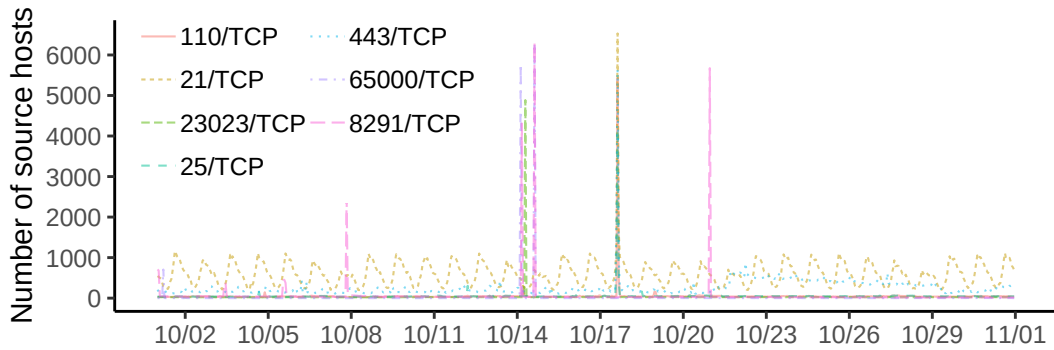


Figure 3.7: The number of source hosts per one hour in October 2018 at TCP ports where router vulnerability-related cyberthreats were observed. (sensor A)

vious section, approximately 500 source hosts observed with 444/TCP when monitored from the whole NICTER, but only about 200 hosts were observed from sensor A, and the number of source hosts was small compared to other threats.

Next, Hajime regularly scanned TCP ports 5358, 9000, and other ports [63]. In Hajime, the window size is fixed at 14600, and one byte before and after the sequence number is 0. Finally, HNS scanned TCP ports 23, 80, 8080, 2480, 5984, and other random ports [64].

Router Vulnerability: The 12 TCP ports of router vulnerability types in Table 3.2 were divided into threats against vulnerabilities presented in the router products from five manufac-

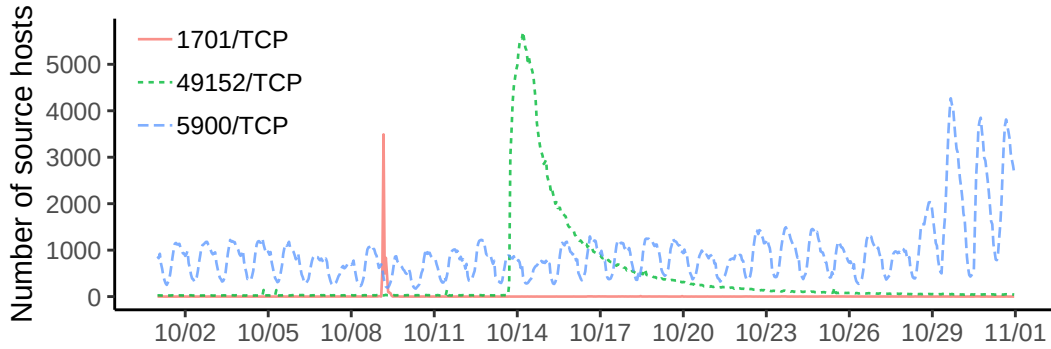


Figure 3.8: The number of source hosts per one hour in October 2018 at TCP ports where other vulnerability-related cyberthreats were observed. (sensor A, part I)

turers. Figure 3.7 shows multiple scans from devices that seem to be manufacturer A’s router products, and they scanned the ports that are open by default [65]. The window size was fixed at 1024 in all cases.

Second, 37215/TCP was scanned from router products of manufacturer B [66], 8181/TCP was scanned from router products of manufacturer C [67], and 8001, 8081/TCP (Fig. 3.4) were scanned from the router products of manufacturer D. The characteristics of Mirai were seen from the scanning activity to these four ports. When we accessed the observed source hosts through the HTTP, many login screens for each router company were displayed. Finally, using the vulnerability of manufacturer E’s Universal Plug and Play (UPnP), multiple router products using manufacturer E’s UPnP were hijacked, and network scans were observed from the infected routers (5431/TCP) [68].

Other Vulnerability: The seven TCP ports of the other vulnerability types in Table 3.2 were broken down into five threats to vulnerabilities of applications and services. The three TCP ports 1701, 49152, 5900 in Fig. 3.8 are scans targeting services such as L2TP VPN (Layer 2 Tunneling Protocol Virtual Private Network), Supermicro BMC (Baseboard Management Controller), and VNC (Virtual Network Computing).

1701/TCP was suddenly scanned from China on October 9. On October 14, Mirai-featured scans from Egypt were observed for 49152/TCP. 5900/TCP gradually increased the

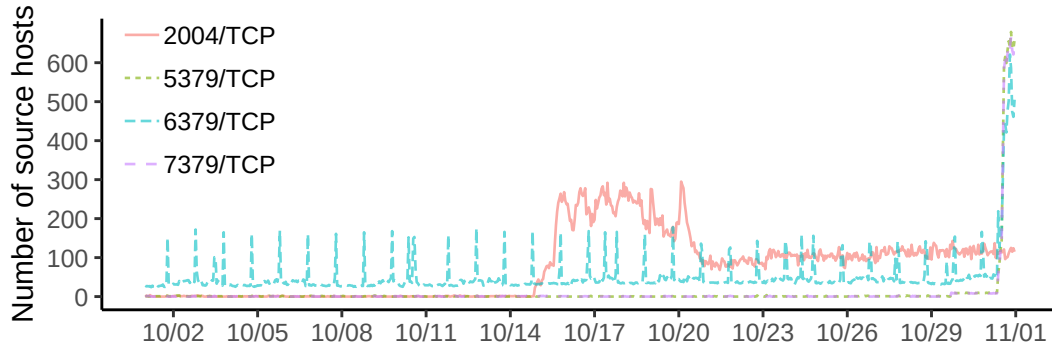


Figure 3.9: The number of source hosts per one hour in October 2018 at TCP ports where other vulnerability-related cyberthreats were observed. (sensor A, part II)

number of source hosts and peaked on October 29. The window size of these scan packets was fixed at 8192, and many hosts considered to be Windows OS were observed. Next, although there are relatively few hosts in Fig. 3.9, scans targeting NoSQL database service vulnerabilities were observed on October 31 (TCP ports 5379, 6379, and 7379). In 2004/TCP, the number of source hosts increased from October 15, and the window size was 14600 or 29200. Many of the hosts were running WordPress.

Many of the cyberthreats in October 2018 were attacks using known malware or known vulnerabilities that were observed continuously or regularly before that time. In particular, Mirai-related threats were regular and complicated. On the other hand, many of the ports without references above are zero-day threats scanned from a large number of source hosts such as manufacturers A, D, and some other vulnerabilities. From this perspective, *Dark-GLASSO* has shown the possibility of detecting zero-day threats at an appropriate time.

3.7 Discussion

In this section, we discuss the adverse effect of limiting the number of source hosts and describe the technical limitations of our method.

Table 3.5: Average performance when *Dark-GLASSO* was run five times for each β , using a confusion matrix

Sensor ID	β	#Unique Alert-times	#TPs	#FPs	#FNs	#TNs
C	300	49	4.8	5.6	13.2	984.4
	500	44	9.2	4.6	8.8	985.4
	600	20	14.6	1	3.4	989
	700	18	18	0	0	990
D	300	38	4.2	6.4	7.8	989.6
	500	31	8.4	7.2	3.6	988.8
	600	15	10.4	2.2	1.6	993.8
	700	14	12	1.2	0	994.8

3.7.1 Limiting the Number of Source Host: β

Among the preprocessing of *Dark-GLASSO* performed in the demonstration experiment, we investigate how limiting the number of source hosts to β affects the performance. We set the β to 300, 500, 600, and 700 for darknet data of sensors C and D for one week from October 1 to 7, 2018, and ran *Dark-GLASSO* five times, respectively. Here, *Dark-GLASSO* was set to the same conditions as in the demonstration experiment. The minimum number of source hosts in one time slot of sensors C and D was 497, 460, the median was 579, 549, and the maximum was 1358, 1260 respectively. Further, we used the alerts obtained without the host restriction of β as an answer label and examined a change in performance. Consequently, 18 alerts were obtained for sensor C and 12 for sensor D.

Tables 3.5 and 3.6 show the average performance when *Dark-GLASSO* was run five times for each β , using a confusion matrix and five evaluation metrics. Also, it shows the number of unique alert-times obtained in five trials for each threshold β . The total number of samples in the confusion matrix was 1,008 because there were 10-minute time slots for one week. The metrics here are same as Tables 3.3 and 3.4, but this time #TNs can be counted, so the accuracy

Table 3.6: Average performance when *Dark-GLASSO* was run five times for each β , using five evaluation metrics

Sensor ID	β	Accuracy [%]	Precision [%]	Recall [%]	F-measure [%]	FPR [%]
C	300	98.13	46.15	26.67	33.80	0.57
	500	98.67	66.67	51.11	57.86	0.46
	600	99.56	93.59	81.11	86.90	0.10
	700	100.00	100.00	100.00	100.00	0.00
D	300	98.59	39.62	35.00	37.17	0.64
	500	98.93	53.85	70.00	60.87	0.72
	600	99.62	82.54	86.67	84.55	0.22
	700	99.88	90.91	100.00	95.24	0.12

is $(\#TPs + \#TNs) / (\#TPs + \#FPs + \#FNs + \#TNs)$, and the false positive rate (FPR) is $\#FPs / (\#FPs + \#TNs)$. As a result, we can see from Tables 3.5 and 3.6 that if a large number of hosts were removed, the performance was significantly reduced, and the number of unique alert-times increased and varied greatly. On the other hand, when a small number of hosts were limited, the performance was improved, and the dispersion between five trials was reduced.

In other words, we conclude that it is better not to limit the number of hosts. Table 3.7 offers summaries on the number of source hosts and runtime per time slot for each sensor (Min, Median, Mean, and Max) and the setting of β in the demonstration experiment. The number of source hosts and the runtime is calculated per time slot, and the runtime refers to the time from the start to the end of algorithm 2. Sensors C, D, E, G, and H, which have small observation scales, could be processed in most cases without randomly selecting a host, and the performance is not considered to have dropped significantly as a consequence.

We face a problem about sensors A, B, and F, which have a large observation scale. While such sensors sometimes did not exceed the threshold β , in many cases the source hosts were randomly selected. Note that in the demonstration experiment in Table 3.7, the β was set to 1100 to 1300, but in the experiment in Tables 3.5 and 3.6, β was set to 300 to 700, and strict

Table 3.7: Summaries on the number of source hosts and runtime per time slot for each sensor (Min, Median, Mean, Max), and the setting of β in the demonstration experiment

Sensor ID	#IPs	β	#Hosts, Runtime [sec]			
			Min	Median	Mean	Max
A	29,182	1100	450, 48.88	1784, 309.22	1753, 286.49	5131, 839.56
B	14,593	1100	290, 16.86	962, 247.93	1011, 233.74	2999, 816.98
C	4,098	1300	144, 7.60	608, 51.21	565, 94.61	2475, 618.37
D	4,096	1300	74, 5.06	576, 29.09	541, 59.63	2457, 543.64
E	8,188	1300	212, 9.17	889, 95.58	871, 164.46	3403, 697.05
F	16,384	1100	307, 16.75	1307, 231.09	1270, 212.48	4177, 737.40
G	2,044	1300	90, 5.18	382, 14.47	359, 25.66	1774, 523.42
H	2,045	1300	93, 4.83	372, 14.19	342, 20.85	1242, 482.96

evaluation was performed. As the sampling rate increases, synchronizations that can be statistically estimated increase, and the reliability of random sampling improves. Therefore, it is considered that the performance has not dropped dramatically in the demonstration experiment as compared to when β in Tables 3.5 and 3.6 is 300. Moreover, looking at the detection accuracy of each sensor in Tables 3.3 and 3.4, the sensor with the larger observation scale shows better detection accuracy. The reason is that the larger the observation scale, the wider the range of cyberthreats that can be observed. This random sampling is indispensable to maintain

real-time processing. Since the accuracy cannot be guaranteed by random sampling, running the same process several times in parallel can be considered as one measure.

3.7.2 Limitation

We describe two limitations that were difficult to evaluate in this study. First, it is difficult to evaluate whether the detected alert time is correct or not. There is no quantitative way to certainly prepare correct labels for this study as well as for studies that detect anomalies through time-series data. Then, we evaluated the performance in a relatively easy way by examining only information on targeted TCP ports without considering when alerts were detected. It means if there were at least one alert for a correct target TCP port during an entire dataset, it would be counted as a true positive. In this work, we demonstrated the cases in which new threats that occurred during the evaluation period could be detected at an appropriate time, so we confirmed the possibility of detecting cyberthreats promptly.

Second, it is difficult to assess whether the hosts with the estimated synchronicity were actually infected the same and behaved in the same way. By extracting and analyzing those hosts that are assumed to be synchronized, more complex knowledge may be obtained. The above two limitations are related to the darknet constraints. Because our darknet does not return a response, only the first SYN packet of TCP arrives. Therefore, we can guess the threat, but we do not completely know the intention of the scan on the darknet. One promising measure to evaluate the above two limitations is to use a darknet and a honeypot together. For example, verifying that scans of the same features as alerts obtained by *Dark-GLASSO* have also been observed in the honeypot will improve the reliability of *Dark-GLASSO*. In order to achieve this goal, it is necessary to consider what kind of honeypots should be prepared and how to integrate and analyze the darknet and honeypot information.

3.8 Conclusion

Early detection of cyberthreats is important to use the Internet safely, and the darknet is suitable for observing global cyberthreats. As a conventional method, there is a method to find the change point of the traffic volume, but technical limitations have come in recent years. The proposed *Dark-GLASSO* automatically detects a time slot where the synchronization among source hosts is abnormally high in real time. Using darknet data, the ground truth of cyberthreats during the test period is created, and a conventional method, ChangeFinder algorithm, is compared with *Dark-GLASSO*. As a result, the detection accuracy of cyberthreats of our method is about 29% better than the conventional method. Through this work, it is possible to quickly grasp global cyberthreats regardless of new types and variants of malware, and this state-of-the-art engine can substantially streamline security operations and reduce the burden of network security operators and incident response teams. The datasets we used in this chapter are available online at <https://csdataset.nict.go.jp/darknet>.

Chapter 4

Dark-NMF: Real-time Detection of Malware Activities Using NMF

4.1 Introduction

Malware that exploits vulnerabilities in cyberspace to launch attacks is evolving and becoming more sophisticated. In particular, with the popularization of Internet of Things (IoT) devices in the recent years, malware targeting these devices have diversified and increased. Many vulnerable IoT devices are infected with IoT malware and exploited for large-scale distributed denial of service attacks, causing serious damage [3]. Moreover, unlike conventional malware, IoT malware performs a higher infection spread speed with their aggressive scanning, e.g., parallel scans to multiple ports [26]. The emergence of diverse and sophisticated IoT malware has further complicated cyber threats.

Malware, which has a self-propagating mechanism via a network, indiscriminately searches for more vulnerable targets through infected victims and spreads the infection. To prevent damage caused by such indiscriminate scanning attacks, it is important to detect a potential malware activities accurately at an early stage before the infection of the malware spreads. However, accurately distinguishing potential threats from a myriad of scanning and probing is like finding a needle in a haystack. In this study, efforts were made to tackle the challenge of

identifying indiscriminate threats quickly and accurately with few false positives and no false negatives.

A darknet that passively monitors with an unreachable dark IP address block is an effective system for observing indiscriminate global cyber threats via the Internet. In addition, since darknet sensors are simple, they can be easily installed on a large scale and can capture a large amount of the first packets of indiscriminate scanning and probing. In the conventional method for analyzing darknet traffic, the change point is detected by ChangeFinder algorithm [7] from the time series data of the traffic volume or the number of unique source hosts to detect cyber threats [8]. However, with the increasing complexity of cyber threats, in many cases, threats cannot be detected by ChangeFinder. Hence, a new approach is required.

Devices infected with similar malware that share a scan module tend to scan in a similar spatiotemporal pattern to search for next infection targets [10]. The spatial pattern refers to the source host feature space and the destination port feature space in network traffic. Such a tendency is also observed in the darknet [9]. In a previous study [16], attempts were made to detect potential malware activities by determining the source hosts that are highly synchronized with the spatiotemporal patterns on darknet traffic. In this chapter, we call this method “Dark-NMF.” Dark-NMF employs nonnegative matrix factorization (NMF) to decompose darknet traffic into latent temporal and spatial patterns to determine the spatiotemporal features that present anomalous synchronization.

Contribution: The previous study [16] demonstrated an applicability of NMF for detecting new malware activities. However, it needs modifications for use in actual operations in real-time. Moreover, its performance should be evaluated using long-term traffic data observed in an actual network to demonstrate its effectiveness and reliability. In this study, we developed Dark-NMF into a real-time processing method. We also validated a practicality of malware activity detection with hyperparameter tuning and quantitative evaluation.

For the evaluation, we prepared the human-labeled ground truth of malware activities in a certain period and compared the detection results of Dark-NMF and existing methods, ChangeFinder [7] and Dark-GLASSO [12]. In conventional Dark-NMF, only the source host feature space is considered as a spatial feature; in contrast, in our modified Dark-NMF, the

destination port feature space too is treated as a spatial feature for improving the detection accuracy of malware activities. Therefore, Dark-NMF can detect all malware activities that should be detected in the ground truth without false negatives; however, compared to the existing methods, more false positives are obtained. In malware activity detection, although fewer false positives are desired, a system without false negatives is highly valued.

Further, Dark-NMF has the advantages such as low calculation cost, no past data required for detection. Dark-NMF can be operated immediately in real-time, and we provide a guideline on selecting an algorithm suitable for practical operation. Thus, information on the timestamp, destination port number, and source IP address of the detected threat can be supplied to organizations managing the security incident response, allowing them to take prompt measures such as detailed malware activities analysis and cause investigation.

4.2 Related Work

An approach similar to ours was adopted by BotSniffer [10] that detects botnets based on spatiotemporal correlation. However, BotSniffer is not suitable for large-scale darknet traffic because it analyzes only specific protocols for small networks and requires response traffic, including payload. On the other hand, Dark-NMF does not impose limiting protocols or require response traffic; it estimates synchronizations at high speed only by using spatiotemporal features and determines the anomalous spatial feature group. Since large-scale darknet observation sensors are used in Dark-NMF, the analysis is based on a much larger number of scanning threats compared to the analysis method for small networks.

Next, we introduce related works that analyze large-scale darknet traffic in a manner similar to Dark-NMF. The ChangeFinder algorithm [7] was used to detect a point of rapid change in darknet traffic with low computational cost, regardless of synchronization. A similar solution, Dark-GLASSO [12] examines the synchronization in darknet traffic. Dark-GLASSO treats the packet arrival time pattern of one source host as one variable and estimates the conditional independence between all variables, assuming that the variables follow a multivariate normal distribution. A situation in which the number of conditional independent variables at a certain

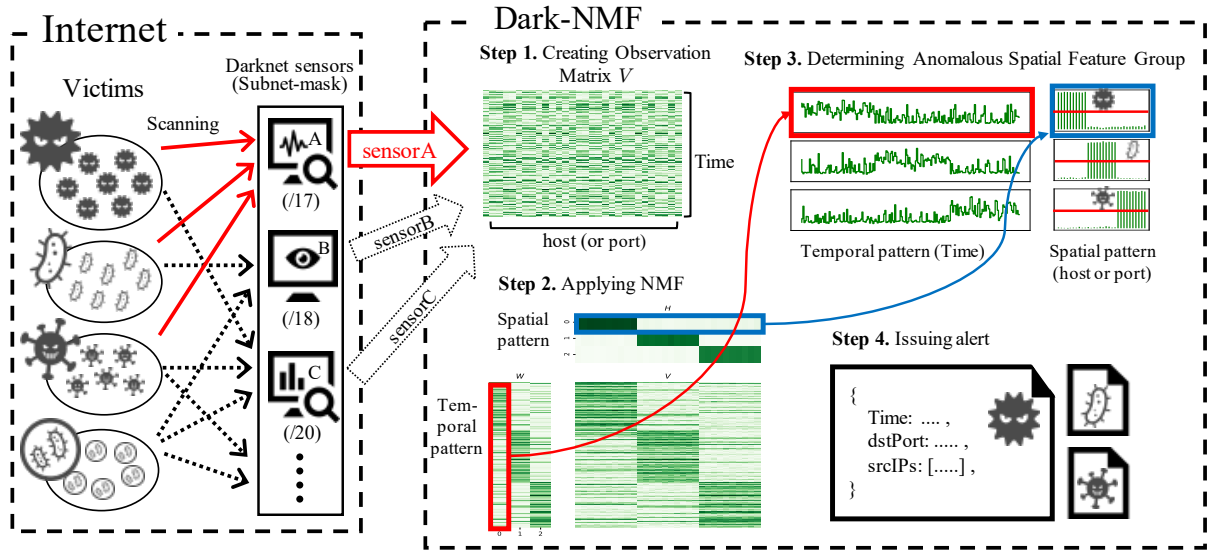


Figure 4.1: Overview of Dark-NMF.

time becomes smaller than that at other time periods implies that there are many synchronized source hosts, and an alert is issued. We performed a quantitative comparative evaluation of Dark-NMF, ChangeFinder, and Dark-GLASSO; the results are provided in Section 4.5.

4.3 System Overview: Dark-NMF

Here, we describe how Dark-NMF applies NMF to darknet traffic data and detects malware activities. Fig. 4.1 shows an overview of Dark-NMF. Note that the system overview has not changed from that in our previous study [16]. Hence, we briefly introduce the process flow below.

4.3.1 Overview

As shown in Fig. 4.1, Dark-NMF involves four steps. Here, the source host feature space is considered an example of spatial features. (1) Create a matrix V representing spatiotemporal features in a unit time from darknet traffic. (2) Observation matrix V is decomposed by NMF

into two matrices W and H that represent the time pattern and spatial pattern, respectively. (3) Determine an abnormal host group from matrix H that represents the spatial pattern. (4) Aggregate the information of the abnormal host group, and issue an alert. In step 4, if many source hosts that are identified as strongly synchronized sent many packets to the same destination port, it is determined that the source hosts are carrying out a scan attack targeting the destination port service. Similarly, when applying a destination port feature space to a spatial feature, if many identical source hosts scan the destination ports identified as strongly synchronized, it is determined that the source hosts are carrying out a scan attack targeting the destination ports.

Dark-NMF is based on the ability of NMF to decompose observation matrix V into several groups with similar time patterns and spatial patterns. Dark-NMF can identify, from among these patterns, the spatial features that are strongly synchronized from matrix H . An examination of the synchronization of the spatial features can eliminate noise communication, such as misconfigured communication. As an example, suppose that scans from victims infected with three types of malware in Fig. 4.1 are simultaneously observed by darknet sensor A. The observed darknet traffic data are decomposed into three groups by NMF, and the information of each victim is issued as an alert.

4.3.2 Step 1: Creating Observation Matrix V

We prepare darknet traffic data for T seconds and assume that N source hosts (or destination ports) are observed in that darknet traffic data. Next, the number of packets is counted for each source host (or destination port) at a sampling interval of T/M seconds. Here, M denotes the number of observation data samples. Thus, observation matrix $V \in \mathbb{N}_0^{M \times N}$ representing the spatiotemporal features is created ($\mathbb{N}_0 = \{0, 1, 2, \dots\}$).

4.3.3 Step 2: Applying NMF

NMF approximates nonnegative observation matrix $V \in \mathbb{N}_0^{M \times N}$ to the product of two non-negative matrices $W \in \mathbb{R}^{M \times r}$ and $H \in \mathbb{R}^{r \times N}$ ($V \approx WH$). Here, r is the number of basis vectors

Algorithm 3 Multiplicative Update Algorithm [17] on Dark-NMF**Input:** Data matrix $V \in \mathbb{N}_0^{M \times N}$, rank parameter $r < \min(N, M)$, threshold stopping criterion ϵ, δ **Output:** $W \in \mathbb{R}^{M \times r}$ and $H \in \mathbb{R}^{r \times N}$ ($V \approx WH$)

- 1: $\ell \leftarrow 0$
- 2: initialize $W^{(\ell)}, H^{(\ell)}$ by singular value decomposition
- 3: **while** $\|V - WH\|^2 < \epsilon$ or $\ell \geq \delta$ **do**
- 4: $H^{(\ell+1)} \leftarrow H^{(\ell)} \frac{[(W^{(\ell)})^T V]}{[(W^{(\ell)})^T W^{(\ell)} H^{(\ell)}]}$
- 5: $W^{(\ell+1)} \leftarrow W^{(\ell)} \frac{[(V H^{(\ell+1)})^T]}{[W^{(\ell)} H^{(\ell+1)} (H^{(\ell+1)})^T]}$
- 6: $\ell \leftarrow \ell + 1$
- 7: **end while**
- 8: $W \leftarrow W^{(\ell)}$
- 9: $H \leftarrow H^{(\ell)}$

and the number of latent patterns to decompose ($r < \min(N, M)$). Since the observation matrix V has no negative value, we adopted the constrained NMF in which no negative values appear even after decomposition. Although several approximate decomposition algorithms of NMF have been proposed, we used the most popular multiplicative update algorithm proposed by Lee et al. [17]. Algorithm 3 shows the procedure of the multiplicative update algorithm on Dark-NMF. Here, ϵ, δ denote thresholds of stopping criterion of Algorithm 3. We set $\epsilon = 0.00001$ and $\delta = 3000$.

4.3.4 Step 3: Determining Anomalous Spatial Feature Group

The values of matrices W and H decomposed by NMF are not uniquely determined. Therefore, we first normalize the scale of the values of matrices W and H . We normalize $W = W\Lambda$, $H = \Lambda^{-1}H$ using diagonal matrix $\Lambda \in \mathbb{R}^{r \times r}$, in which the reciprocal of each column's sum is the diagonal component so that the sum of each column of the matrix W becomes 1. Thus, the value of normalized matrix H has the same scale as the actual number of observed packets for a spatial feature.

We consider spatial features with matrix H value of less than 1 as inactive noise features in each basis vector. In contrast, spatial features with matrix H value equal to or greater than 1 are regarded *active features* in each basis vector. In other words, there is either no *active feature* or one *active feature* in one basis vector. In addition, among the *active features*, those that are strongly synchronized are called *anomalous active features*. When more than β spatial features exceed α (%) of the maximum value (maximum number of packets) of the element in each basis vector, it is determined as an *anomalous active features*.

4.3.5 Step 4: Issuing Alert

Finally, if an *anomalous active feature* exists in a basis vector of matrix H , an alert is issued for each basis vector. An alert is issued by aggregating the time, destination port number, and source host information when many source hosts send packets to a specific destination port among *active features* or *anomalous active features*.

4.4 Implementation and Dataset

In this section, we briefly introduce the real-time processing implementation method of Dark-NMF and describe the details of hyperparameter tuning and the dataset used for the evaluation in this study. In the previous study [16], the conventional Dark-NMF did not have a mechanism for real-time online processing, and only some case studies were described, without a quantitative evaluation of malware activity detection. In this study, we quantitatively evaluated an online processing method for Dark-NMF. Dark-NMF processes T seconds of darknet traffic at a time. In this implementation, we applied the sequential processing of T seconds of darknet traffic every t seconds ($t < T$). In addition, real-time processing was realized by processing Dark-NMF when darknet traffic data for T seconds were newly observed.

4.4.1 Hyperparameter Tuning

Here, we describe the hyperparameters that are set when implementing Dark-NMF. First, we introduce the hyperparameters that are set to fixed values. When creating observation matrix V , period T for observing darknet traffic was set to 1,800 s, and the number of samples M was set to 30. That is, the sampling interval T/M of the observation matrix V was 60 s. The number of basis vectors r of NMF was set from 1 to 10. These hyperparameter settings were selected based on the experience of the previous study [16]. Finally, time interval t for online processing was set to 300 s.

Next, we introduce four hyperparameters to be tuned. Since these hyperparameters have a large effect on the alert output, we investigated changes in the detection accuracy of malware activities while adjusting these values.

- $u \in \{1, 2, \dots, 32\}$ (bit): We adjust a source host's observation unit by separating the upper bits of the source IPv4 address. That is, the source host feature space is analyzed based on the network class.
- $f \in \{0, 1\}$: It indicates whether the information required for issuing an alert is collected from *active features* ($f = 0$) or *anomalous active features* ($f = 1$).
- $\alpha \in [0, 100]$ (%): It refers to the ratio of the maximum number of packets, which is the threshold value when determining *anomalous active features* from *active features*.
- $\beta \in \mathbb{N}$: It refers to the threshold value of the number of spatial features when determining *anomalous active features* from *active features*.

The setting and tuning of these hyperparameters is described later in Section 4.5.

4.4.2 Dataset and Ground Truth

In this section, we describe the details of darknet traffic used in the experiment and the ground truth of malware activities in that data. First, we used eight darknet sensors installed worldwide; the observable malware activities may differ depending on the geography of the

sensor. The observation scale of darknet sensors used in this implementation varied from about 30,000 IPv4 addresses (subnet-mask /17) to about 2,000 IPv4 addresses (subnet-mask /21). In all, about 80,000 IPv4 addresses were observed via these eight darknet sensors.

For the observation period of darknet traffic used in this implementation, the data for one month, i.e., October 2018 was used. We analyze TCP-SYN traffic on darknets where indiscriminate scanning attacks are observed. This dataset is from the same period as that used in the previous related study [12] and has been published at <https://csdataset.nict.go.jp/darknet/>. When creating observation matrix V from this dataset, we excluded the following TCP ports that were known and constantly threatened to highlight unknown malware activities: 22, 23, 80, 81, 445, 1433, 2323, 3389, 5555, 8080, and 52869.

Next, the ground truth of malware activities prepared for the evaluation experiment is described below. This ground truth was created by manually investigating the malware activities observed on our darknet by TCP ports in October 2018 based on the results of expert analysts and Dark-GLASSO [12]. The following 35 TCP ports indicate the correct answer table. The malware activities caused by IoT malware and the vulnerabilities related to router manufacturers and various service protocols were clarified.

Ground truth (35 TCP ports): 21, 25, 82, 83, 84, 85, 88, 110, 443, 444, 1701, 2004, 2480, 5358, 5379, 5431, 5900, 5984, 6379, 7379, 7547, 8000, 8001, 8010, 8081, 8088, 8181, 8291, 8443, 8888, 9000, 23023, 37215, 49152, 65000.

Since it is difficult to accurately prepare the ground truth for the time and source host information of malware activities, we did not consider such information in this evaluation. The purpose of this ground truth was to evaluate that Dark-NMF successfully detected “TCP ports where malware activities were observed.” Our goal was to find the hyperparameter sets that minimize false negatives, even with a few false positives. With regard to false detection, there were no false negatives in malware activity detection after Dark-NMF was so tuned; hence, we regarded it best to secondarily investigate whether the events detected by Dark-NMF were correct (true positive) or not (false positive). In the future, we aim to automate this secondary investigation as well.

4.5 Performance Evaluation

The evaluation of the malware activity detection performance of Dark-NMF based is described in this section. We conducted two experiments, one using the source host feature space and the other using the destination port feature space as the spatial features of the observation matrix V in Dark-NMF. We also compared Dark-NMF with the existing approaches, namely, ChangeFinder [7] and Dark-GLASSO [12].

The evaluation indexes used in this experiment are the number of destination ports detected correctly (i.e., the number of true positives, #TPs) and the number of destination ports detected incorrectly (i.e., the number of false positives, #FPs). The number of false negatives is $(35 - \text{\#TPs})$, so we neglect it. Also, the number of true negative may be treated as $2^{16} - (\text{\#TPs} + \text{\#FPs})$ for all TCP ports, but we do not consider it as it is not an essential index in this evaluation.

4.5.1 Experiment 1: Anomalous Synchronization Detection of Source Host Feature Space

First, we used the source host feature space as the spatial feature and experimented with detecting TCP ports targeted by the source hosts that were abnormally synchronized. We conducted experiments on eight sensors A–H, changing the values of the four hyperparameters $\{u, f, \alpha, \beta\}$. We adjusted u with 16 bits and 32 bits. A u of 16 bits means that the source host is analyzed in network class B units, and a u of 32 bits means that one IPv4 address was used for analyzing the source host. We adjusted f with values of 0 and 1, and set α and β at $\{50, 65, 80\}(\%)$ and $\{2, 5\}$ respectively. Among the experimental results of the eight sensors, the results of sensors A (/17), E (/19), and G (/21) are listed in Table 4.3. Since the results of other sensors showed the same tendency, they are omitted. For example, when $\{u, f, \alpha, \beta\} = (16, 1, 50, 2)$ on sensor E, all 35 TCP ports were correctly detected (true positives), including 415 false positives.

The experiment showed that f has considerable influence on the result. When $f = 1$, #TPs was close to all correct answers, but #FPs was over ten times greater than #TPs. On the other

Table 4.1: Results of Experiment 2 for each α and each sensor. Integration is the result of integrating all eight sensors. (#TPs, #FPs)

Sensor \ α	80	90	95	98
A (/17)	9, 91	8, 36	8, 12	5, 1
B (/18)	9, 143	8, 51	7, 15	7, 2
C (/20)	7, 85	6, 27	2, 10	2, 2
D (/20)	6, 72	5, 15	3, 6	3, 0
E (/19)	6, 92	5, 25	4, 9	3, 1
F (/18)	12, 110	8, 23	8, 2	8, 1
G (/21)	7, 104	5, 28	3, 4	2, 0
H (/21)	7, 148	4, 45	4, 20	4, 2
Integration (80K #IPs)	13, 498	12, 177	11, 58	10, 9

hand, when $f = 0$, #FPs was close to 0, but #TPs was less than that corresponding to $f = 1$. The tendency of the result of u differed with f . When $f = 0$, the tendency differed slightly depending on the sensor observation scale, but the change in #TPs was small regardless of $u = 16, 32$. When $f = 1$, $u = 32$ had less #TPs than that for $u = 16$ regardless of the sensor observation scale. Finally, we found that larger the α and β , the smaller are both #TPs and #FPs. In particular, the smaller the sensor observation scale, the faster was the decrease in #TPs.

4.5.2 Experiment 2: Anomalous Synchronization Detection of Destination Port Feature Space

We used the destination TCP port feature space as the spatial feature and attempted to detect the destination port group in which the pattern of the number of packets scanned to destination ports were abnormally synchronized. Dark-NMF issues an alert when the detected destination port group is scanned by many of the same source hosts. We did not perform this

experiment separately; instead, we conducted Experiment 2 to suppress #FPs and increase #TPs by combining the alert results from Experiment 1. Specifically, using the alert time and TCP port detected in $\{u, f, \alpha, \beta\} = (16, 0, 50, 2)$ of Experiment 1 as a trigger, in Experiment 2, we investigated whether any other TCP ports were abnormally synchronized with the TCP port at the alert time. In Experiment 2, we evaluated how many additional 14 TCP ports that were false negatives in $\{u, f, \alpha, \beta\} = (16, 0, 50, 2)$ of Experiment 1 could be detected.

In the hyperparameter setting in Experiment 2, it is unnecessary to set u because the spatial feature is the TCP port feature space. In addition, since only alerts due to *anomalous active features* are required in this experiment, we fixed $f = 1$ and $\beta = 2$ and adjusted only α in the range of $\{80, 90, 95, 98\}$. The experimental results obtained with eight sensors are summarized in Table 4.1. It was found that the larger the α value, the fewer were #TPs, and the more significantly fewer were #FPs. In particular, when $\alpha = 98$ was obtained by integrating the results of all eight sensors, 10 #TPs of the target 14 TCP ports could be realized with low #FPs.

An example of the alert result in this experiment is shown in Fig. 4.2. This alert result was detected from sensor A at 18:30 on October 31, 2018 (JST). In Experiment 1, TCP port 5900 was detected, and in Experiment 2, 5379, 6379, and 7379 were detected as TCP ports that were abnormally synchronized with TCP port 5900 as a trigger. Fig. 4.2 shows the number of packets in the three dimensions of time, source IP address, and destination TCP port number in 1-min units at the alert time. If the source host IP addresses match between multiple ports in 1-min units, the plot is red, and if they do not match, the plot is green. It was found that the source host IP addresses of ports 5379, 6379, and 7379 were almost duplicated, and in fact, scan attacks targeting the same service were seen on these ports.

4.5.3 Comparative Evaluation with Related Works

This section compares and evaluates the malware activity detection results of Dark-NMF Experiments 1 and 2 and the related research ChangeFinder [7] and Dark-GLASSO [12]. The results are listed in Table 4.2. The results of related works are available in literature [12].

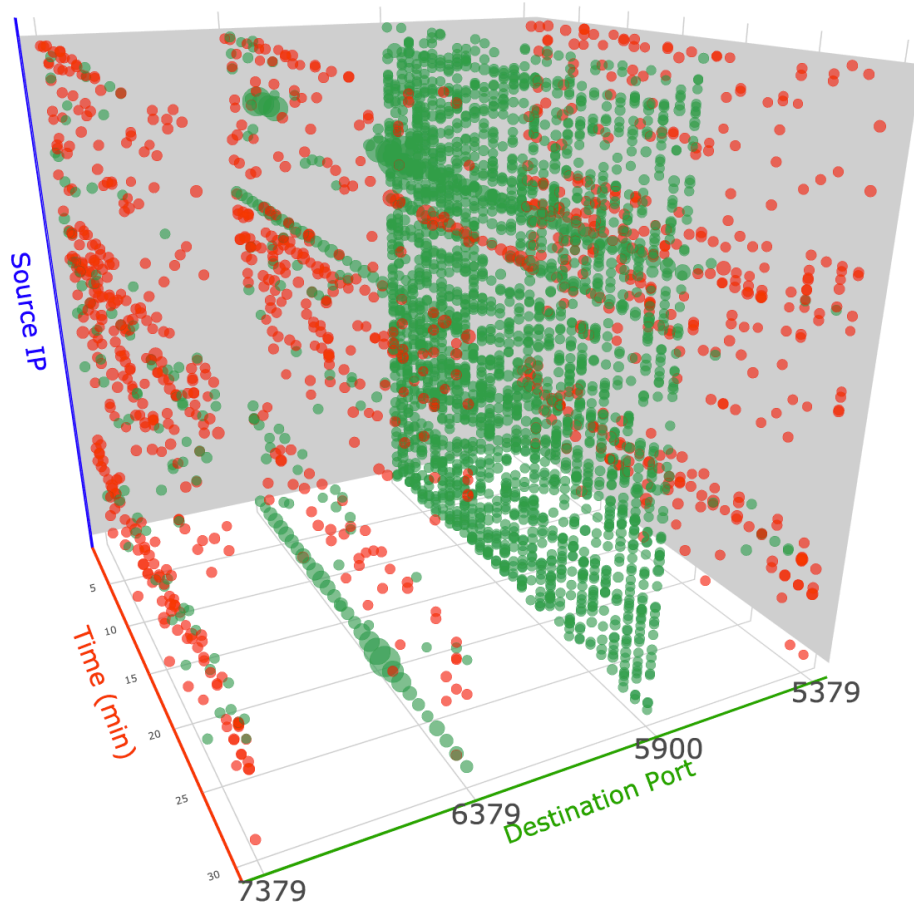


Figure 4.2: Three-dimensional plot of the number of packets per minute as an example of the alert result in Experiment 2.

For comparison, Dark-NMF results obtained with the following three hyperparameter settings were considered:

SET1 $\{u, f, \alpha, \beta\} = (16, 0, 50, 2)$ of Experiment 1.

SET2 SET1 + $\alpha = 98$ of Experiment 2.

SET3 $\{u, f, \alpha, \beta\} = (16, 1, 50, 2)$ of Experiment 1.

The integration results of ChangeFinder, Dark-GLASSO, and Dark-NMF SET1 detected 24, 34, and 21 #TPs, respectively, while suppressing #FPs to 0. The integration result of

Table 4.2: Malware activity detection results of ChangeFinder, Dark-GLASSO, and Dark-NMF for each sensor. Integration is the result of integrating all eight sensors. (#TPs, #FPs)

Sensor	Change-Finder	Dark-GLASSO	Dark-NMF		
			SET1	SET2	SET3
A (/17)	22, 0	29, 0	19, 0	24, 1	33, 466
B (/18)	19, 0	29, 0	20, 0	27, 2	33, 470
C (/20)	22, 0	25, 0	19, 0	21, 2	28, 348
D (/20)	21, 0	25, 0	20, 0	23, 0	30, 516
E (/19)	21, 0	27, 0	21, 0	24, 0	35, 415
F (/18)	22, 0	26, 0	20, 0	28, 1	31, 371
G (/21)	22, 0	19, 0	20, 0	22, 0	29, 459
H (/21)	20, 0	19, 0	20, 0	24, 2	30, 457
Integration (80K #IPs)	24, 0	34, 0	21, 0	31, 9	35, 1074

Dark-NMF SET2 drastically increased #TPs while suppressing the increase in #FPs in comparison with SET1; this was made possible by combining SET1 and the result of Experiment 2. Finally, the integration result of Dark-NMF SET3 could detect all 35 TCP ports where malware activities were observed, but with many #FPs.

The results of ChangeFinder and Dark-GLASSO contained few #FPs because the number of source hosts required to issue alerts was limited to 20 or more. On the other hand, SET3 issues an alert if two or more source hosts are abnormally synchronized. Therefore, it is possible to detect TCP ports with a small infected-host scale of the observed malware activity, but there is a trade-off relationship in which #FPs increases. Meanwhile, the results of each method other than SET3 showed a tendency to overlook TCP ports with a small infected source host scale of the observed malware activity. Furthermore, the results of ChangeFinder and SET1 tended to overlook persistent malware activities that constantly kept the scale of infected hosts to a certain level.

4.6 Discussion: Dark-NMF vs. Dark-GLASSO

ChangeFinder does not consider synchronization and its detection performance is relatively poor, so we do not discuss it in this section. Both Dark-NMF and Dark-GLASSO eliminate misconfigured communication and uncorrelated communication that accidentally reach the darknet by estimating synchronization. In other words, Dark-NMF and Dark-GLASSO are robust to noise in darknet traffic. Table 4.2 shows that Dark-GLASSO provides less #FPs than Dark-NMF. However, Dark-NMF has several advantages over Dark-GLASSO. They are described below.

Low calculation cost. The biggest drawback of Dark-GLASSO is that it requires cubic time complexity $O(N^3)$, where N is the spatial feature in the observation matrix V . For Dark-GLASSO to perform processing in real-time, it is necessary to sample the source host feature space randomly. In recent years, the amount of observed traffic has been increasing in the darknet. Hence, in the future Dark-GLASSO should perform analysis after randomly sampling many source host feature spaces. It is known that the detection accuracy of Dark-GLASSO decreases as more random sampling is performed. On the other hand, Dark-NMF functions in linear time $O(N)$ and processing can be completed in real-time without sampling.

No past data required for detection. Dark-GLASSO requires long-term past data to determine whether the input data are abnormal, but Dark-NMF recognizes abnormal patterns on the input data. Therefore, Dark-GLASSO takes a long time to obtain the detection result every time the hyperparameters are adjusted, but Dark-NMF can immediately obtain the detection result.

Decomposing spatiotemporal features. Dark-NMF uses the NMF algorithm to decompose spatiotemporal features into frequent latent patterns and recognize anomalies within those patterns. Therefore, it is possible to detect malware activities with a small scale of infection. On the other hand, since Dark-GLASSO estimates the synchronization between all features without decomposing features, it may not be possible to capture malware activities with a small scale of infection.

Low influence of the darknet observation scale. Dark-GLASSO certainly yields few #FPs

and has good detection accuracy. However, as seen from Table 4.2, #TPs of Dark-GLASSO differ greatly depending on the darknet sensor observation scale. On the other hand, SET3 yielded high #TPs regardless of the sensor observation scale.

Application of spatial feature. When the destination port feature space is used as the spatial feature, the size of observation matrix V becomes enormous when all destination ports are scanned. This matrix can be analyzed by Dark-NMF because the calculation is completed in linear time, but the calculation is difficult by Dark-GLASSO, which requires cubic time complexity.

Thus, Dark-NMF has many advantages over Dark-GLASSO. As a method of utilizing and operating Dark-NMF, SET2 and Dark-GLASSO are assumed to be used together based on SET3. Although SET3 has good performance for #TPs, using SET3 alone for analysis is inefficient because it yields many #FPs. Therefore, by using SET2 and Dark-GLASSO together, reliable #TPs can be extracted from the many events detected by SET3.

The human-labeled ground truth used for the evaluation in this study is imperfect. It is possible that the many #FPs in SET3 actually contain TCP ports where malware activities were observed. There is no doubt that the events found by Dark-NMF are highly synchronized. When we manually checked some #FPs, we found many TCP ports that were synchronized by survey-scanning organizations such as Shodan¹ and Censys². Such survey scanners need to be distinguished from malware activities. As a future work, we will consider a method for analyzing the actual truth or falsehood of #FPs.

4.7 Conclusion

We quantitatively evaluated Dark-NMF, a method for detecting potential malware activities in real-time. Dark-NMF focuses on synchronizing the spatiotemporal features seen during the spread of malware infections from the enormous number of indiscriminate scanning attacks observed on the darknet. Using the human-labeled ground truth, we tuned four hyperparam-

¹<https://www.shodan.io/>

²<https://censys.io/>

ters in Dark-NMF and compared its performance with the performances of ChangeFinder and Dark-GLASSO. We found that Dark-NMF has a hyperparameter set that can detect all malware activities that should be detected in the ground truth, including malware activities with a small infection scale. Finally, we described the advantages of Dark-NMF over Dark-GLASSO and the effective use of Dark-NMF. We expect that Dark-NMF will contribute to developing rapid responses to malware activities.

Table 4.3: Results of Experiment 1 with four tuned hyperparameters $\{u, f, \alpha, \beta\}$. (#TPs, #FPs)

Hyperparameters		sensor A (/17)				sensor B (/18)			
α	u	16		32		16		32	
	$\beta \setminus f$	0	1	0	1	0	1	0	1
50	2	19, 0	33, 466	19, 0	31, 345	20, 0	31, 493	21, 3	29, 350
	5	19, 0	33, 464	19, 0	31, 345	20, 0	31, 478	21, 3	29, 348
65	2	19, 0	33, 448	19, 0	31, 347	20, 0	33, 493	21, 3	28, 376
	5	19, 0	33, 436	19, 0	31, 341	20, 0	33, 470	21, 3	28, 374
80	2	19, 0	33, 390	19, 0	31, 342	20, 0	31, 439	21, 3	28, 373
	5	19, 0	33, 362	19, 0	31, 328	20, 0	31, 382	20, 2	28, 363
		sensor C (/20)				sensor D (/20)			
50	2	19, 0	28, 449	22, 20	29, 498	20, 0	30, 573	25, 20	30, 652
	5	19, 0	28, 444	22, 19	29, 490	20, 0	30, 569	25, 19	30, 647
65	2	19, 0	28, 424	22, 19	29, 496	20, 0	30, 543	25, 20	28, 642
	5	18, 0	28, 409	22, 12	29, 481	20, 0	30, 516	25, 16	28, 631
80	2	19, 0	28, 390	22, 17	28, 456	20, 0	29, 435	25, 19	28, 600
	5	17, 0	28, 348	20, 4	28, 411	18, 0	29, 382	19, 4	28, 555
		sensor E (/19)				sensor F (/18)			
50	2	21, 0	35, 415	21, 5	27, 453	20, 0	31, 488	21, 3	29, 354
	5	21, 0	33, 410	21, 5	27, 449	20, 0	31, 475	21, 3	29, 351
65	2	21, 0	32, 386	21, 5	27, 546	20, 0	31, 495	21, 3	29, 372
	5	21, 0	32, 377	21, 4	27, 540	20, 0	31, 480	21, 3	29, 368
80	2	21, 0	31, 356	21, 4	27, 528	20, 0	31, 404	21, 3	29, 366
	5	20, 0	31, 315	20, 1	27, 498	19, 0	31, 375	21, 1	29, 358
		sensor G (/21)				sensor H (/21)			
50	2	20, 0	29, 459	25, 34	29, 519	20, 0	30, 484	24, 25	29, 535
	5	20, 0	29, 441	24, 32	29, 506	20, 0	30, 462	24, 23	29, 527
65	2	20, 0	29, 429	25, 32	29, 490	20, 0	29, 464	24, 25	29, 509
	5	18, 0	29, 390	21, 27	28, 448	20, 0	29, 415	19, 13	29, 480
80	2	20, 0	29, 396	25, 30	19, 348	20, 0	29, 421	23, 22	23, 401
	5	17, 0	29, 314	13, 9	18, 231	17, 0	27, 330	10, 4	21, 249

Chapter 5

FCSG: Investigating Behavioral Differences between IoT Malware

5.1 Introduction

With the widespread use of IoT devices, IoT malware that infects vulnerable IoT devices is spreading. Mirai, which is a family of IoT malware, has formed a large-scale botnet and caused considerable damage, including the world's largest 623 Gbps DDoS (Distributed Denial-of-Service) attack [3]. Malware whose source code has been released or leaked, e.g., Mirai and Bashlite, has become more easily exploited and causes different variants to proliferate. More than 70,000 new IoT malware were identified in the first quarter of 2020, and this number has been increasing year over year [69]. Therefore, accurate analysis of malware functionality and quick responses are required to prevent damage.

Reusing a code base or sharing code logic is common in malware [6]. In addition, malware variants evolve as attackers gradually change their behaviors. Thus, it is important for malware variants to see what behaviors have been added or removed. Antivirus software analyzes malware specimens to determine which malware family a specimen belongs to. However, it is very difficult to identify what behaviors each malware specimen can invoke and what behavioral differences exist between malware specimens even in an identical malware family. For

example, a specimen identified as Mirai could send UDP packets for DDoS attacks, whereas another specimen of Mirai would not possess that functionality.

In a previous study, Iwamoto et al. [70] analyzed Windows malware statically, generated directed graphs by extracting application programming interface (API) calls, and then clustered malware by calculating the graph similarity between malware specimens. Their method draws an API call graph from static analysis and visualizes API call relationships; however, they only focused on clustering. That is, their method is insufficient to know specific behaviors of each malware in a cluster. In addition, they did not consider user-defined functions and the effects of other environmental factors, e.g., instruction set architecture (ISA) and compilers. We have set out to expand the visualization of malware specimens' behavior in API call graphs and consider a method to identify behavioral differences using signatures rather than a malware clustering technique.

To investigate the actual conditions of malware, especially IoT malware, we attempt to automatically estimate behavioral differences between large-scale malware specimens by focusing on which functions (e.g., `open`, `read`, `printf`, and `close` in C programming) are called and their sequence. Some of those function calls are used to implement specific behaviors of malware. Here, a behavioral difference refers to changes in attack types and malware behaviors.

We propose a method to generate a function-call-sequence graph (FCSG) that considers both API names and names of called user-defined functions from the disassembled code of malware specimens. In addition, we implement a method to avoid environmental problems in IoT malware, e.g., ISA and compilers, by applying normalization on function names or removing nodes in FCSGs. Next, we generate the FCSGs from the disassembly result or the released source code of a specific IoT malware family, and the characteristic behaviors of the malware are created manually as signature-FCSGs. When a signature-FCSG is matched to the FCSGs of a malware specimen, the malware specimen can be considered to have a behavior that corresponds to the signature-FCSG.

The proposed method is used to specify malicious behaviors of malware binaries captured by honeypot systems for investigating functional behaviors between malware, assuming that

the honeypot systems do not capture any benign software (i.e., the proposed method does not target any benign software in this paper). Also, an FCSG and a signature-FCSG can handle malware on different ISAs because it focuses only on function calls. Moreover, an analyst can confirm function calls briefly, and signatures can be created easily from the source code or disassembled code.

Contributions: To the best of our knowledge, this is the first work to extract behaviors contained in IoT malware specimens in a graph, confirm the behavioral differences between specimens using graph-based signatures, and discover specimens containing activities from multiple malware families. In this study, we used FCSGs to investigate behaviors of 24,126 in-the-wild IoT malware specimens running on Linux of IoT devices and confirmed that ISA-independent signature matching is possible. Our primary contributions are summarized as follows.

- We propose generating a FCSG from the disassembled code of a malware specimen and estimating the behavioral differences among large-scale malware specimens regardless of ISAs or malware families (Sections 5.3 and 5.4).
- We identify mixed variants with multiple malware family features that match some signature-FCSGs generated from the source code of different malware families (Section 5.5.3).
- We identify malware variants that may have been removed from some behaviors and specialized in some attacking behaviors (Section 5.5.4).

5.2 Preliminaries

Here, we explain some important elements; instruction set architecture and disassemble by IDA Pro.

5.2.1 Instruction Set Architecture

There are multiple CPU types (e.g., ARM, MIPS, and Intel x86) used in IoT devices, and each CPU has its own unique instruction set architecture (ISA). This fact makes us difficult to identify the behavior of IoT malware as follows.

Binary files with different ISAs cannot be executed on the same Linux OS (e.g., a specimen of ARM does not run on Linux OS built for MIPS); therefore, many IoT malware prepares binary files compiled for multiple ISAs in advance and executes the binary that is suitable for a target IoT device to be infected. This results in that different binaries can be generated by compilers even from the same source code if the target ISA is distinct.

Such malware binaries pose this challenge: *how well can we identify a specific behavior (e.g., sending UDP packets for DDoS attacks) shared between given binaries with different ISAs?* If we simply compare those binaries byte by byte, this result is to show that the malware binaries greatly differ from each other due to the different ISAs, whether or not they are generated by reusing any part of the same source code. Identifying behaviors of IoT malware directly leads to capturing behavioral differences between IoT malware, and so we tackle this problem in this study with a graph-based method focusing on function calls, which is proposed in Section 5.3.

Note that we use a reverse engineering tool to extract function calls without executing malware specimens (i.e., in a static analysis manner) for our method. We can also choose dynamic analysis; however, there are some major drawbacks of dynamic analysis. One is that executing each specimen is time consuming, and another drawback is that IoT malware would not reveal malicious behaviors unless they receive commands through their C2 (Command and Control) servers.

5.2.2 Disassemble by IDA Pro

IDA Pro [71] is a reverse-engineering software tool working for multiple ISAs (e.g., ARM and MIPS). This tool can disassemble a binary file to provide its basic blocks, where a basic block of a binary file means a set of multiple machine codes that are divided by branch and

return instructions. The tool also provides a flow chart (i.e., the control flow) of the processing order of the basic blocks.

With IDA Pro, we can obtain function calls in the processing order from that control flow because basic blocks include function calls embedded in a binary file and they are sorted in the processing order. Those function calls are used for generating the FCSGs introduced in Section 5.3.2. In addition, IDA Pro shows good performance against IoT malware [72], which also supports us to choose IDA Pro as the reverse-engineering tool we use in this study.

5.3 Proposed Method

In this section, we propose a method for identifying behaviors of IoT malware. This leads to investigating behavioral differences between IoT malware as is demonstrated in Section 5.5.

5.3.1 Overview

We assume that a specific *behavior* of IoT malware is achieved by a set of functions including system calls (e.g., `socket`, `bind`, `listen`, and `exec`). Examples of such behaviors include *accessing to a C2 (Command and Control) server* and *executing a Linux command*.

Figure 5.1 shows an overview of the proposed method for identifying behaviors of a given specimen. Our method focuses on the FCSG (Function Call Sequence Graph) explained in Section 5.3.2 in detail. Intuitively, the FCSG is a directed graph of functions with the processing order. Given a specimen, the whole binary file can be represented as an FCSG, and a specific behavior is encapsulated by a sub-graph in the FCSG. The key idea behind our method is to previously extract sub-graphs corresponding to behaviors as signatures.

For identification of behaviors, we manually create signatures (as *Generating Signature-FCSGs* in Fig. 5.1). We then automatically generate FCSGs of target malware specimens with tools (as *Generating FCSGs*). Finally, we also automatically conduct a signature matching against those specimens to be investigated (as *Signature Matching*). If a signature-FCSG is matched to an FCSG of a specimen as a sub-graph, this specimen is considered to perform the

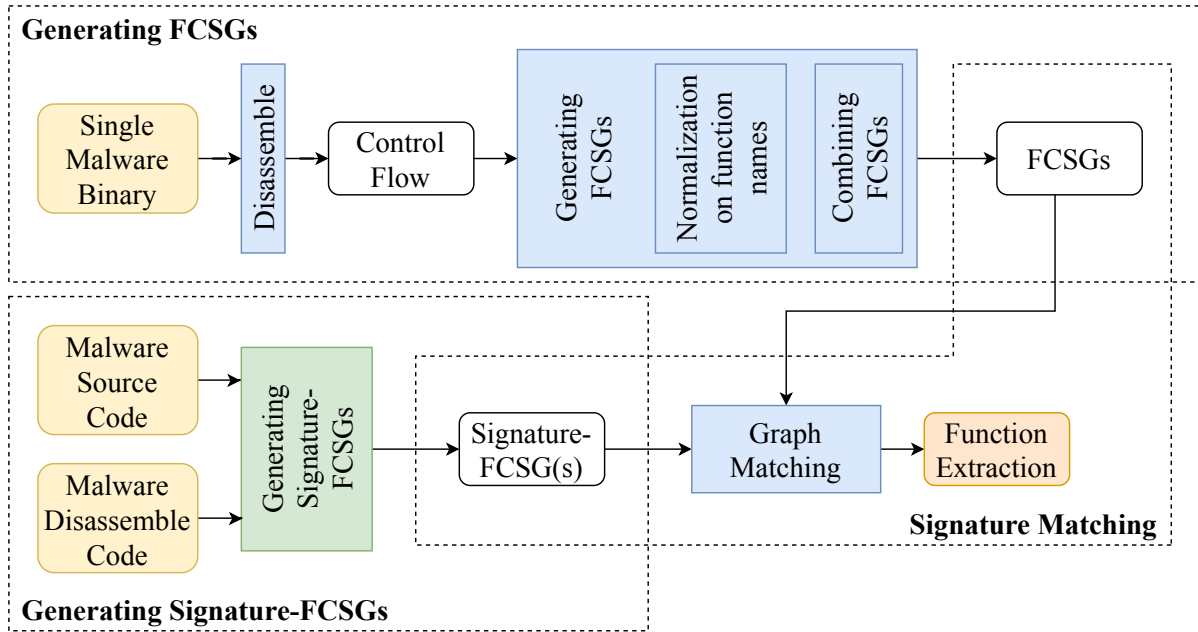


Figure 5.1: Overview of proposed approach (yellow: inputs; orange: outputs; blue: automatic process; green: manual process)

behavior corresponding to the signature.

There is a problem of the ISA differences that we aim to solve, as is explained in Section 5.2.1. We have to match a signature with IoT malware specimens even if they are binaries with different ISAs. The signature of FCSG can absorb the ISA differences because it only focuses on functions as nodes of FCSG. This is based on the fact that those functions will be called in the processing order predefined in source code without depending on ISAs, and so the FCSG and the signature are robust against ISA differences and will work well.

For details, *Generating FCSGs* is explained in Section 5.3.2, and *Generating Signature-FCSGs* and *Graph Match* are explained in Section 5.3.3.

5.3.2 Function Call Sequence Graph

The FCSG, a directed graph representing the sequence of function calls, is extracted from a binary file as follows. A binary file is disassembled to obtain its disassembly code, and

Program 5.1: Sample program

```

int get_random() {
    int ret; int fd = open("/dev/random", O_RDONLY);
    read(fd, &ret, sizeof(ret)); close(fd);
    return ret;
}

int main() {
    for(int i = 0; i < 4; i++) {
        int res = get_random(); printf("%d\n", res);
    }
    return 0;
}

```

all function calls invoked by the binary are recognized. For each function (e.g., `main` and `get_random`), the control flow of the basic blocks is created. Given a control flow, the corresponding FCSG is then generated by extracting function calls from the control flow with preserving the processing order. In FCSGs, a *node* is a function call labeled with its function name, and a directed *edge* is linked from a function to next function to be called according to the control flow. The *root node* of an FCSG is the function name for which the corresponding control flow is created (e.g., `main` and `get_random`).

For better understanding FCSG, Fig. 5.2 shows the control flow of `main` function of Program 5.1 compiled by the GCC compiler for ARM (named `arm-linux-gnueabi-gcc v9.3.0`), and Fig. 5.3 shows that of `get_random` function of Program 5.1. The corresponding FCSGs to `main` and `get_random` are shown in Fig. 5.4. As `get_random` $\rightarrow$ `printf`, a directed edge is linked from a function to next function to be called. The root node `main` is also linked to `get_random`, which is called first in the control flow of `main`.

An FCSG is combined with another FCSG if any node in an FCSG is the root node in another FCSG. For example, in Fig. 5.4, `get_random` node in the left FCSG is the root node

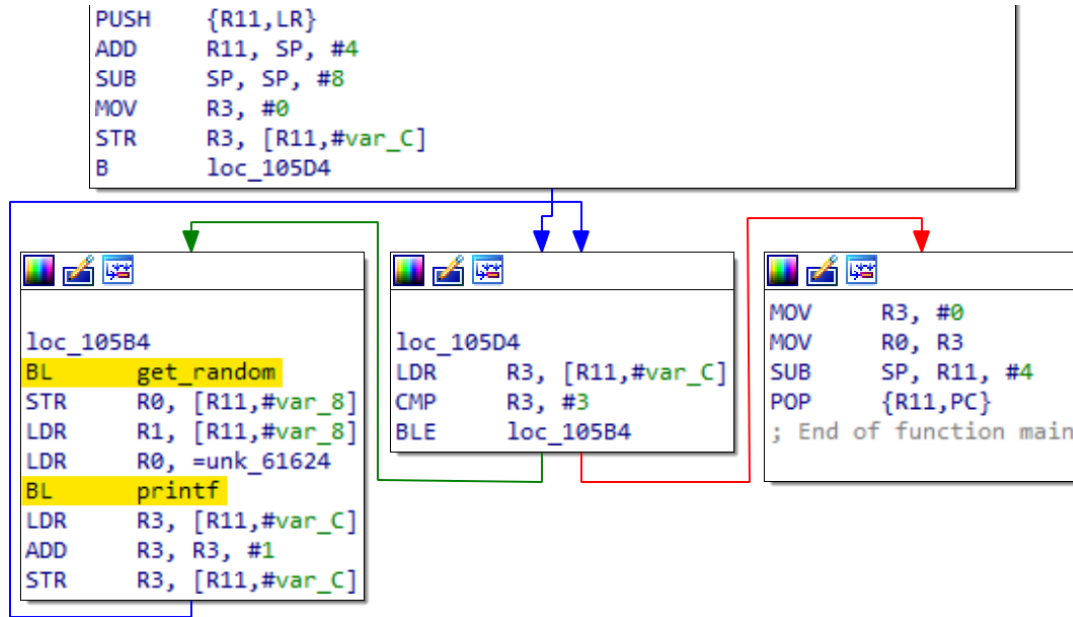
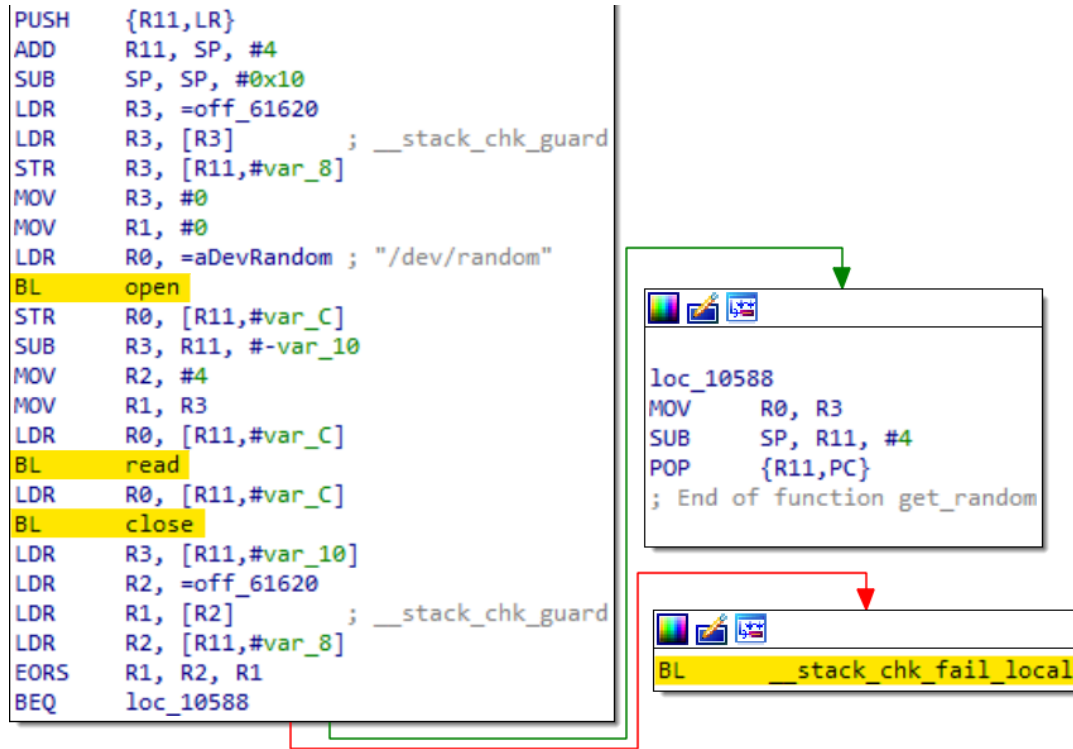


Figure 5.2: The control flow of main function from ARM binary (in Program 5.1)

in the right FCSG, and then those FCSGs are combined to the united FCSG shown in Fig. 5.7. In this manner, the FCSGs for all functions are combined as a united FCSG of a specimen, except for standard functions (e.g., `open` and `read` in C) including Linux system calls. The FCSG for a standard function could be also generated from the functions that consist of that standard function; however, we do not generate any FCSGs of standard functions. This is because standard functions do not conduct malicious tasks by themselves. We only generate FCSGs of use-defined functions and combine them.

As a case study for showing robustness of FCSG against different ISAs, Figures 5.5 and 5.6 show the control flows obtained with IDA Pro after compiling Program 5.1 by the GCC compiler for MIPS (mips-linux-gnu-gcc v9.3.0). Comparing Fig. 5.5 (compiled for MIPS) with Fig. 5.2 (compiled for ARM), it can be seen that the structures of basic blocks of `main` are quite different. On the contrary, there is no difference in the sequence of function calls. This can be also seen comparing Fig. 5.3 with Fig. 5.6. By using FCSGs, it is possible to extract the behavior even in different ISAs.

Figure 5.3: The control flow of `get_random` function from ARM binary (in Program 5.1)

5.3.3 Signature Matching

Signatures of FCSG are manually generated as follows. We obtain the source code of an IoT malware specimen or the disassembly code. We then analyze the codes to learn specific behaviors of the specimen and the part of the code corresponding to each behavior. After that we generate a FCSG from the corresponding part of the code for a behavior. We call this FCSG a *signature-FCSG* for that behavior. For identifying behaviors of a given specimen, we check if a signature-FCSG is matched to the united FCSG of the specimen as a sub-graph. If they match, we determine that the specimen has the behavior corresponding to the matched signature. For example, when the signature-FCSG (`open` → `read` → `close`) in Fig. 5.8 is applied the combined FCSG in Fig. 5.7, the combined FCSG's dashed nodes are matched. This signature represents the reading of a file.

IDA Pro cannot obtain function names from specimens in the following case: Mirai speci-

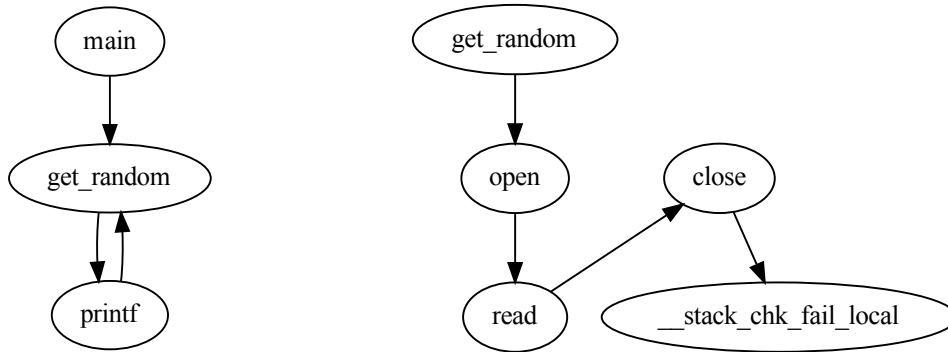


Figure 5.4: FCSGs for each control flow (in Fig. 5.2 and Fig. 5.3)

mens may call user-defined functions through a function-descriptor table previously embedded by malware authors in the specimens. For this, the specimen first obtains the address of a function from the descriptor table. The obtained address is then stored in a CPU register (e.g., `r8` in Intel x86_64), and via this register the function is called (e.g., `call r8`). In this case, IDA Pro just identifies the register name used for use-defined function calls. To treat with this problem, we take the register name (e.g., `r8`) as the name of a node in FCSGs to generate FCSGs of the specimen. Because IDA Pro does not identify which function is called via the CPU register, we do not combine FCSGs based on any register names.

5.4 Implementation

This section introduces an implementation of the proposed method.

5.4.1 Outline of Proposed Method

Algorithm 4 describes instructions in pseudo code that are followed after manually generating signature-FCSGs. In the algorithm, function *GenerateFCSGs* takes as input a malware binary *M* to output FCSGs of *M*, and function *SignatureMatching* conducts signature match-

```

loc_400888:
jal    get_random
nop
lw     $gp, 0x20+var_10($fp)
sw     $v0, 0x20+var_4($fp)
lw     $a1, 0x20+var_4($fp)
lui    $v0, 0x46 # 'F'
addiu  $a0, $v0, (aD - 0x460000) # "%d\n"
la     $v0, printf
move   $t9, $v0
bal    printf
nop
lw     $gp, 0x20+var_10($fp)
lw     $v0, 0x20+var_8($fp)
addiu  $v0, 1
sw     $v0, 0x20+var_8($fp)

```

Figure 5.5: The partial control flow of main function from MIPS binary (in Program 5.1)

ing against M with each of the input signature-FCSGs $[S_1, S_2, \dots]$. The output of Algorithm 4 is a signature-match vector V . This vector represents which signatures are matched to M , and the malware is then considered to have the specific behaviors corresponding to the matched signatures.

In the rest of this section, Section 5.4.2 explains how we manually generated signature-FCSGs. Sections 5.4.3 and 5.4.4 explain functions *GenerateFCSGs* and *SignatureMatching* in detail, respectively. Those functions are automated with software tools and scripts.

5.4.2 Generating Signature-FCSGs

In this study, we select Mirai and Bashlite because they are very popular IoT malware samples and the source code of theirs is often reused. Their released source codes can be download from GitHub [73, 74]. After obtaining the source code, we selected some basic behavior and behavior for attacking. We then generated a signature based on the file name, program structure, function name, and comments. Consequently, we generated 13 signatures for Bashlite and 15 signatures for Mirai, and these signatures are listed in Tables 5.1 and 5.2,

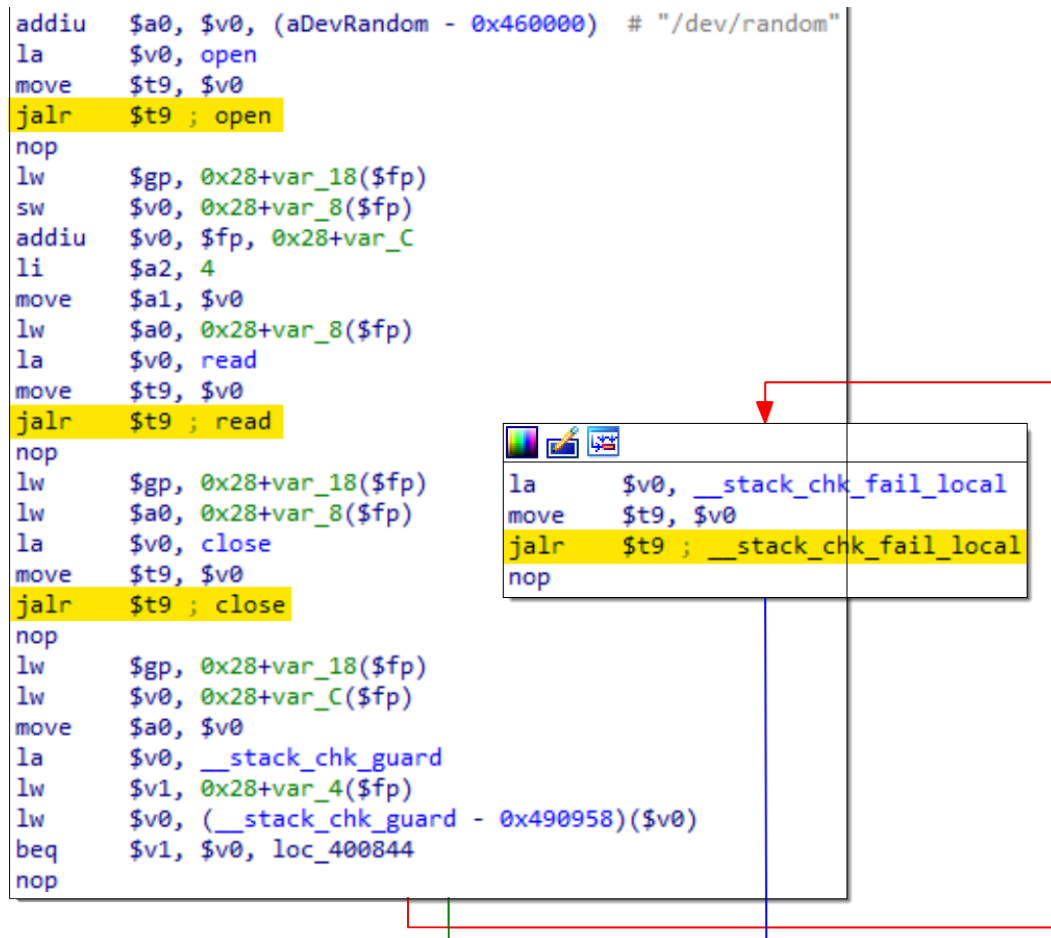


Figure 5.6: The partial control flow of `get_random` function from MIPS binary (in Program 5.1)

respectively. The Bashlite source code comprises a single file, and the Mirai source code is divided into several files; thus, regarding Mirai's signature in Table 5.2, the file name is listed prior to the signature name.

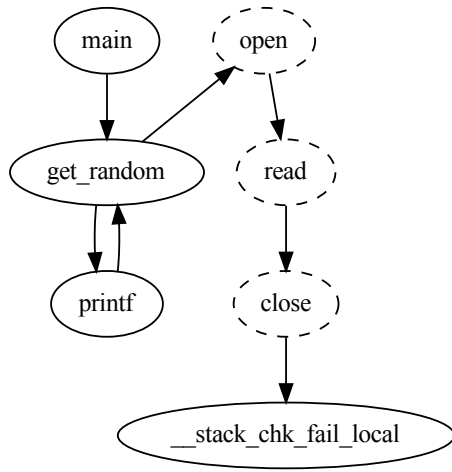


Figure 5.7: Combined FCSG from Fig. 5.4 based on
main function

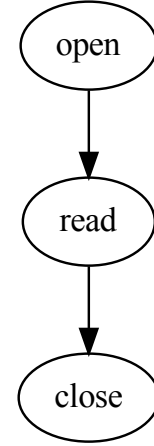


Figure 5.8: Sample signature-FCSG

5.4.3 GenerateFCSGs

Generating FCSGs

As a plug-in of IDA Pro 7.4, we create a Python script to retrieve the control flow for each function and extract function call sequences from the control flow. This script searches the basic block in the control flow while considering branch instructions, and it finds call instructions. If the instruction calls a function name, the script records the called function name with the function name called immediately before in the same control flow. This script targets ISAs of Intel x86_64, x86, ARM MC 68000, MIPS, and Sparc, and it considers `call`, `BL`, `bal`, `jal`, `jalr`, and `bsr.l` as call instructions.

Next, we use Python 3.8 and NetworkX [75] 2.4 to generate a directed graph from function call sequences obtained by the script. Note that some function names are affected by fluctuations in notation and evident compiler influence; therefore the following processing is required.

Algorithm 4 Outline of the proposed method**Input:** Malware Binary M , Signature-FCSGs $[S_1, S_2, \dots, S_m]$ **Output:** Signature match vector $[v_1, v_2, \dots, v_m]$ ($v_j \in \{0, 1\}$)

```

1:  $[G_1, G_2, \dots, G_n] \leftarrow \text{GenerateFCSGs}(M)$        $\{G_1$ :united FCSG,  $G_2, \dots, G_n$ : not-combined
   FCSGs}
2:  $V \leftarrow [0] * m$                                 {Init the signature match vector}
3: for  $i = 1$  to  $n$  do
4:   for  $j = 1$  to  $m$  do
5:     if  $\text{SignatureMatching}(S_j, G_i)$  then
6:        $V[j] = 1$       {If  $S_j$  matches  $G_i$  as a sub-graph, return true otherwise return false}
7:     end if
8:   end for
9: end for

```

Normalization on function names

When combining FCSGs, the label of nodes (i.e., function names) are used to match a node in a FCSG and the root node in another FCSG. Some compilers add specific prefixes to the names of standard library functions prior to generating binaries; therefore, to absorb the differences between compilers, we remove the beginning part of the function name. If the specific prefix containing underscore exists (such as `_GI_`, `_libc_` and `_uClibc_`), it is removed.

We then delete the following nodes (i.e., functions) from malware's FCSGs. ARM-based CPUs do not support the modulo operation at the machine language level. If a modulo operation exists in the source code, it is replaced with the `modsi3` function, which obtains similar operational results using a compiler for ARM. Second, when initializing an array, the `memset` function may or may not be used. Finally, the compiler adds the `__errno_location` function so the program can handle the error. Thus, all `modsi3`, `memset`, and `__errno_location` functions are deleted.

Table 5.1: List of created Bashlite signatures

Signature Name	Behavior
access_to_c2	Check if the C2 server is accessible
echo_build_info	Send its own binary information to the C2 server
copy_process	Process replication
ping_pong	Reply ‘PONG’ when ‘PING’ is sent from the C2 server
exec_command	Execute the command sent from the C2 server
get_parameter	Get the parameters sent from the C2 server
get_ip	Get the IP address of the current device
get_mac	Get the MAC address of the current device
get_random_str	Generate a random string
sendHOLD	Perform a HOLD attack
sendJUNK	Perform a JUNK attack
sendTCP	Perform a TCP attack
sendUDP	Perform a UDP attack

Combining FCSGs

We define *user-defined functions* as functions that are not listed in the man page provided by the JM Project [76]. However, in some Mirai malware, there are some functions that are called by the functions’ address, not functions’ name. Therefore, we combined FCSGs as much as possible; however, we also attempted signature matching for FCSGs that could not be combined.

5.4.4 SignatureMatching

We adopt the VF2 [77] graph matching algorithm, which is implemented on NetworkX. By default, the VF2 algorithm only checks a graph’s shape; however, it is possible to check matching node names by defining a feasibility rule. Some compilers change the name of

Table 5.2: List of created Mirai signatures

(File name) Signature Name	Behavior
(main) init_argcheck	Check command line arguments
(main) init_hide_argv0	Hide command line arguments
(main) init_hide_pname	Hide process name
(main) init_print_system_exec	Get other running processes
(main) initializer	Various initialization processes
(main) init_connect	Establish a connection to the C2 server
(scanner) scanner_init_init	Initialization of other device scanners
(scanner) scanner_init_main	Execution of other device scanners
(attack) attack_parse	Get attack parameters
(attack) attack_start	Create a process and start an attack
(attack) add_attack	Registration of attack function
(attack_app) attack_app_http	Attack on HTTP
(attack_app) attack_app_cfnul	Attack on HTTP (large POST)
(attack_gre) attack_gre_ip	Attack on GRE IP
(attack_gre) attack_gre_eth	Attack on GRE Ethernet

functions (e.g., `strstr` to `strstr_r` or `_strstr`). To treat such functions as the same function (e.g., `strstr` and `_strstr` are treated as the same function), we define the following feasibility rule: after unifying function names to the lowercase, nodes are matched if the function name on the signature side is partially included in the specimen's FCSGs.

An example of actually generated FCSG from a Bashlite specimen is shown in Fig. 5.9. The parts where signature-FCSGs matched are marked in red. In addition, Figures 5.10 and 5.11 show examples of the parts of Figure 5.9 that match signature-FCSGs (marked in pink).



Figure 5.9: Generated FCSG from a Bashlite specimen

5.5 Investigation

In this section, we show the results obtained by our investigation with the proposed method. It was conducted against the dataset containing in-the-wild IoT malware. This section begins with the outline of our dataset, followed by the discussion on the investigation results with some interesting findings.

5.5.1 Dataset

Using the IoTPOT [78], 54,544 specimens were collected from October 2, 2016 to January 20, 2018. We disassembled specimens and used 24,126 cases whose main function was identified. For specimens whose main function was unidentified, it is highly probable that the function name has been deleted or the file has been corrupted. Therefore, such specimens were excluded from the analysis.

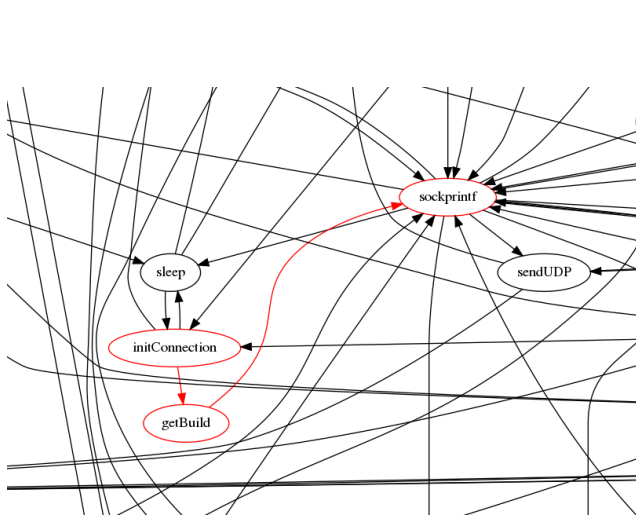


Figure 5.10: Matched signature-FCSG on Fig. 5.9's FCSG (part 1)

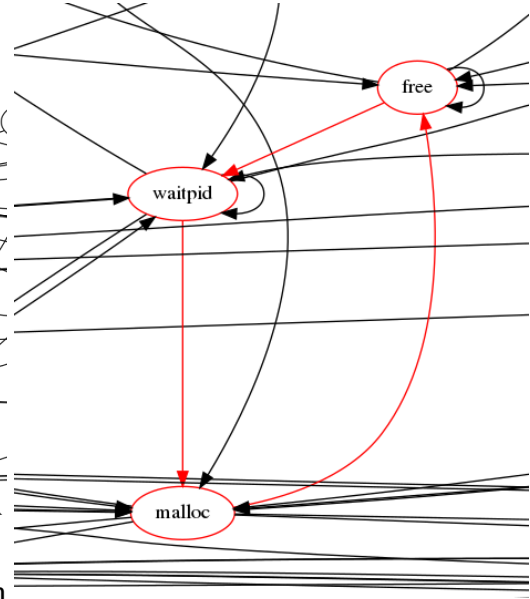


Figure 5.11: Matched signature-FCSG on Fig. 5.9's FCSG (part 2)

Table 5.3 shows a breakdown of the malware family names and ISAs (Instruction Set Architectures) acquired by the Linux `readelf` command [79] and AVClass [46]. Here, specimens whose malware family name is NaN were not uploaded to VirusTotal [80] during data acquisition. In addition, SINGLETON is a specimen whose AVClass could not determine a malware family name for various reasons, e.g., all of VirusTotal's labels are generic.

We generated signature-FCSGs from source code of Bashlite and Mirai for this investigation, and extracted FCSGs from all specimens. The order and size of those FCSGs are summarised in Table 5.4, where *order* and *size* mean the number of nodes and that of edges in an FCSG. Table 5.4 shows Min., Max., Avg., and Std. denote minimum, maximum, average, and standard deviation, respectively. In the table, *All* means those of all specimens (i.e., 24,126 specimens). In addition to all specimens, Table 5.4 only shows the values for the Bashlite and Mirai specimens and omits the specimens categorized to the other families (e.g., Tsunami and Lightaidra). This is because the other families had the tendency similar to Bashlite and Mirai.

Table 5.3: Breakdown of dataset by ISA and malware family

Malware family	ISA						Total
	x64	ARM	x86	MC 68000	MIPS	Sparc	
Bashlite	2154	4871	4468	2069	4431	2033	20026
Mirai	1	56			64	1	122
Tsunami	49	110	94	44	118	38	453
Lightaidra		1		3			4
SINGLETON	1			1	3		5
NaN	372	909	729	365	795	346	3516
Total	2577	5947	5291	2482	5411	2418	24126

5.5.2 Signature Matching Results

Table 5.5 shows the number of Bashlite signature matches (in Table 5.1) and a breakdown of the malware family. Among the analysis targets, 20,026 malware were determined to be Bashlite by AVClass, and there existed 19,954 Bashlite specimens that matched one or more signatures. Similarly, Table 5.6 shows the number of Mirai signature matches (in Table 5.2) and a breakdown of the malware family. Here, 122 specimens were determined to be Mirai by AVClass, and one or more signatures matched 121 specimens. In addition, Tables 5.7 and 5.8 show the number of matched specimens for each signature.

5.5.3 Investigation 1: Specimens Mixed with Malware Families

As shown in Tables 5.5 and 5.6, we found that some specimens matched multiple signatures from different malware families. These are mixed variants created by adding some behaviors of specimens with different malware families.

Bashlite with Mirai behavior: As shown in Table 5.6, 119 Bashlite specimens matched the Mirai signature. These specimens matched two of Mirai's signatures, i.e., (main) `init_hide`

Table 5.4: Order and size of Signature-FCSGs and FCSGs

			Min.	Max.	Avg.	Std.
Signature-FCSGs	Bashlite	order	2	7	3.9	1.5
		size	1	7	3.5	1.6
	Mirai	order	2	14	6.0	4.0
		size	1	21	6.5	6.0
FCSGs (per a specimen)	All	order	125	3534	538.2	189.1
		size	255	6723	949.6	243.6
	Bashlite	order	125	2599	530.6	178.4
		size	257	3753	944.1	230.3
	Mirai	order	308	3534	556.7	302.3
		size	586	6723	878.7	557.1

`_argv0`, `(main) init_hide_pname`, or only `(main) init_hide_argv0` (Table 5.8). When we decompiled and checked some of these specimens, their structure of basic blocks was that of Bashlite. Thus, these Bashlite specimens may have been generated with part of the Mirai source code to conceal the process name and argument.

Mirai with Bashlite behavior: As shown in Table 5.5, there was a single Mirai specimen that matched a Bashlite signature, and it was the signature `copy_process` in Table 5.7. However, the corresponding signature was not matched to the other Mirai specimens. Therefore, this single specimen is considered a Mirai variant, and it is possible that process duplication processing was added from the Bashlite source code.

In such cases, an analyst with prior knowledge may analyze the existence of multiple malware families' behavior. Therefore, it is expected that analysis can be performed while referencing the information of multiple malware families.

Table 5.5: How many Bashlite signatures out of 13 signatures were matched to specimens in the dataset (e.g., five Bashlite signatures were matched to 40 Tsunami specimens). The signatures are listed in Table 5.1.

	Malware familiy						Total
	Bashlite	Mirai	Tsunami	Lightaidra	SINGLETON	NaN	
Signature match counts	0	57	121	5		68	251
	1	13	1	268		61	343
	2	193		94		52	339
	3	274		10	1	37	322
	4	254		36	2	73	365
	5	575		40	2	119	736
	6	650			2	167	819
	7	587			2	169	758
	8	1327				452	1779
	9	3277				584	3861
	10	3410				528	3938
	11	4432				628	5060
	12	4798				548	5346
	13	179				30	209

5.5.4 Investigation 2: Specialized Specimens for Some behaviors

As shown in Table 5.5, there were specimens whose Bashlite's signature was only partially matched. The following signatures were not matched when we conducted signature matching against those specimens.

- get_parameter
- get_mac
- sendHOLD
- get_ip
- exec_command
- sendJUNK

We decompiled those specimens and found that corresponding behaviors had been removed. From the above, it has been confirmed that there are simplified Bashlite variants pro-

Table 5.6: How many Mirai signatures out of 15 signatures were matched to specimens in the dataset. The signatures are listed in Table 5.2.

	Malware familiy						Total
	Bashlite	Mirai	Tsunami	Lightaidra	SINGLETON	NaN	
0	19907	1	453	4	2	3404	23771
1	1	52				22	75
2	118				3	60	181
3		2				1	3
4		13				21	34
5						1	1
6		1					1
7		23				1	24
8		12				3	15
9		15				2	17
12		2				1	3
13		1					1
14							0
15							0

duced by deleting some behaviors. In addition, the proposed method can detect the deletion of behavior in a malware specimen. In such cases, the analyst may concentrate on analyzing existing behavior. Therefore, it is expected that rapid analysis will become possible.

5.5.5 Investigation 3: Specimens Matched with All Signatures

Some specimens matched all Bashlite signatures and most Mirai signatures. It is presumed that these specimens have nearly the same behavior group as the GitHub source code referenced when generating the signature. In such cases, nearly all source code is diverted when these specimens were created. Therefore, it is possible to respond quickly to malware based

Table 5.7: The number of specimens matching each Bashlite signature

Signature name	Malware familiy						Total
	Bashlite	Mirai	Tsunami	Lightaidra	SINGLETON	NaN	
access_to_c2	17630			1	5	2540	20176
echo_build_info	11982			1		1195	13178
copy_process	19721	1	447	4	5	3366	23544
ping_pong	19480			4	5	3295	22784
exec_command	5206					919	6125
get_parameter	13270			4	4	2265	15543
get_ip	17920		147			3019	21086
get_mac	17658		1			2925	20584
get_random_str	19224		40	4	2	3195	22465
sendHOLD	8225		33			1191	9449
sendJUNK	8194		33			1181	9408
sendTCP	18454		76	4		3110	21644
sendUDP	19021		53	4		3173	22251

on the source code or existing analysis results.

5.5.6 Investigation 4: Specimens Unmatched with Any Signatures

Although specimens were determined to be Bashlite by AVClass, there were 57 specimens that did not match any Bashlite signatures. By investigating these specimens, it was confirmed that the program structure (i.e., the structure of basic blocks) differed from the released Bashlite source code [73]. As a result, the order of function calls changed significantly; therefore, although these specimens were determined as Bashlite, it is probable that the signatures did not match.

We also investigated one specimen determined to be Mirai by AVClass; however, none of the Mirai signatures were matched. When we checked the disassembled result, we found

Table 5.8: The number of specimens matching each Mirai signature (None of the signatures were matched to Tsunami and Ligtaidra)

Signature name	Malware familiy				Total
	Bashlite	Mirai	SINGLETON	NaN	
init_argcheck		67		29	96
init_hide_argv0	119	53	3	67	242
init_hide_pname	118	53	3	67	241
init_print_system_exec		67		30	97
initializer		67		30	97
init_connect		1			1
scanner_init_init		3		1	4
scanner_init_main		3		1	4
attack_parse		31		6	37
attack_start		54		8	62
add_attack		121		52	173
attack_app_http		17		3	20
attack_app_cfnnull		2		1	3
attack_gre_ip		3			3
attack_gre_eth		3			3

similar parts to both source codes (Bashlite [73] and Mirai [74]), but the structure for which the signature was created was different. Therefore, it appeared that the antivirus software detected the structural part of Mirai, and although the specimen was determined to be Mirai by AVClass, none of the signatures were matched.

Since it can be estimated that the specimens, as mentioned above, have a different structure from the referenced source code, it is necessary to analyze them as new specimens. However, once signatures are generated from such modified specimens, those signatures can be applied to other specimens whose structure is the same as or similar to that of those modified

specimens.

5.5.7 Investigation 5: Specimens in Other Malware Families

Table 5.7 shows that there existed Tsunami and Lightaidra specimens matching some Bashlite signatures. This indicates that they had the behaviors corresponding to the matched signatures of Bashlite. Table 5.8 also shows that Bashlite and SINGLETON were the only specimens that matched Mirai signatures. Considering this result, the source code of Mirai could not be often used to implement malware specimens except for Bashlite, whereas some Bashlite behaviors were implemented in multiple malware families.

Furthermore, it can be estimated that SINGLETONs, which could not be named by AV-Class in this investigation, had both Bashlite and Mirai behavior. Therefore, even for such unknown specimens, the proposed method was able to identify some of their specific behaviors.

5.6 Related Work

Relative to the function call sequence, Iwamoto et al. [70] extracted the call sequence of API functions after static analysis of Windows malware. Then they created a directed graph and calculated the similarity of the graph. Kikuchi et al. [81] generated API call graphs in the same manner reported in the literature [70], where common subgraphs were extracted between graphs. They proposed a method to measure the similarity between API call graphs and common subgraphs in order to cluster them and determine whether the specimens are unknown malware. In this study, we extend their method [70] to include API calls and user-defined functions in the graph. We also add function name processing to make the influence of ISA less susceptible.

Focusing on malware signatures, Feng et al. [82] vectorized the control flow obtained by static analysis and collected the vectors generated from multiple ISA binaries. They then proposed a method to determine whether an unknown binary file contains a vulnerability by vec-

torizing the vulnerability part into a signature and matching vectors at high speed. Xu et al. [42] improved processing speed using a neural network based on the method [82]. Sathyanarayan et al. [83] extracted Windows API calls via static analysis and calculated the appearance frequency of important APIs in API calls. They proposed a signature creation method that can detect entire malware without being bound by the malware family. The above methods generate signatures automatically by some specific calculation; however, associating signatures with malware behavior is difficult. In the current study, we hope to create a signature that the analyst can easily associate with a behavior; therefore, the FCSG is adopted.

5.7 Conclusion

In this study, we proposed a method for identifying behaviors of IoT malware across different ISAs. The key idea behind our method to absorb ISAs differences is to use signature-FCSG, based on the fact that the processing order of function calls is not affected by ISA differences. We then investigated 24,126 in-the-wild malware specimens captured by an IoT POT system to show behavioral differences. We confirmed that some specimens were generated by reusing part of source code of Mirai and that of Bashlite, which was one of our key findings in this study. For efficient malware analysis, analysts should keep it in mind that a target specimen to be analyzed could belong to more than one IoT malware family; they could behave like Mirai and sometimes like Bashlite, for example.

As a drawback of the proposed method, it is necessary to manually generate signature-FCSGs. In our future work, we plan to consider a way to support generating signature-FCSGs. As another future work, we aim to investigate evolutionary relationships of malware variants. To this end, we plan to apply FCSGs to a phylogenetic tree of malware generated by the method of [84]. Currently, method [84] can divide malware specimens into clusters based on the generated tree; however, it does not show behavioral differences between malware specimens in the same cluster. With FCSGs, we hope that their behavioral differences will be uncovered in detail.

Chapter 6

Conclusion

Against cyber attacks on diversifying and advanced IoT malware, this thesis has achieved the goal of automating the analysis of malware attack behavior and deriving early warning information before it becomes a significant incident. First, as the first issue, we proposed Dark-GLASSO and Dark-NMF, which are methods for real-time detection of malware activities by darknet analysis. As a conventional method on the darknet analysis, there is a method to find the change point of the traffic volume, but technical limitations have come in recent years. Dark-GLASSO automatically detects a time slot where the synchronization among source hosts is abnormally high in real time. Dark-NMF focuses on synchronizing the spatiotemporal features seen during the spread of malware infections from the enormous number of indiscriminate scanning attacks observed on the darknet. Dark-GLASSO and Dark-NMF take a similar approach that focuses on malware synchronization. Therefore, these methods are resistant to noise; packets that arrive accidentally in a darknet space due to misconfigurations will not show synchronization. However, there are differences in the calculation process and the results obtained. Dark-GLASSO detects anomalies by comparing the quantified degree of synchronization with the numerical values of other periods, while Dark-NMF recognizes anomalous patterns in the decomposed spatial patterns without the need to compare the results with other periods.

Using the human-labeled ground truth, we tuned four hyperparameters in Dark-NMF and

compared its performance with the performances of ChangeFinder and Dark-GLASSO. As a result, the detection accuracy of cyberthreats of Dark-GLASSO was about 29% better than ChangeFinder. In addition, we found that Dark-NMF has a hyperparameter set that can detect all malware activities that should be detected in the ground truth, including malware activities with a small infection scale. Finally, we described the advantages of Dark-NMF over Dark-GLASSO and the effective use of Dark-NMF. As a tendency of the two proposed engines found by the result of quantitative evaluation, Dark-GLASSO is good at detecting large-scale malware activities without false detection. Dark-NMF can detect small-scale malware activities, but it tends to have more false positives than Dark-GLASSO. Through this work, it is possible to quickly grasp global cyberthreats regardless of new types and variants of malware, and these state-of-the-art engines can substantially streamline security operations and reduce the burden of network security operators and incident response teams. The datasets we used in this chapter are available online at <https://csdataset.nict.go.jp/darknet>.

Future works for Dark-GLASSO and Dark-NMF are described below. The two proposed systems can quickly capture malware activities and are useful for security operations, but we do not know what kind of malware activities the detected event refers to. It may be false-positive events such as survey scans that have nothing to do with malware activities. In the future, we would like to consider a model that automatically analyzes the detected events. Specifically, we extract features from the traffic header related to the detected events and classify the events. By classifying, malware activities and survey scans can be distinguished, and it can be expected that more detailed behavior of malware activities can be captured.

Secondly, since the ground truth used in this thesis includes only TCP ports where malware activities were observed, it has not been possible to evaluate whether the detection time of the captured event is appropriate. In the future, we can expect to be able to assert the possibility of early detection of malware activities even more strongly by creating a ground truth that includes information on malware activities' timing. Finally, with the current proposal systems, the detected event ends only once without tracking. In the future, we will focus on the port where malware activity was once detected and consider a model that captures pattern classification and changes by tracking darknet traffic information on that port.

Next, as the second task, we proposed FCSG, which investigates the functional differences between a large number of malware variants by malware analysis. We proposed a method for identifying behaviors of IoT malware across different ISAs. The key idea behind our method to absorb ISAs differences is to use signature-FCSG, based on the fact that the processing order of function calls is not affected by ISA differences. We then investigated 24,126 in-the-wild malware specimens captured by an IoTPOT system to show behavioral differences. We confirmed that some specimens were generated by reusing part of source code of Mirai and that of Bashlite, which was one of our key findings in this study. For efficient malware analysis, analysts should keep it in mind that a target specimen to be analyzed could belong to more than one IoT malware family; they could behave like Mirai and sometimes like Bashlite, for example.

Future works for FCSG are described below. Currently, it takes a lot of time and effort to create a signature-FCSG. We want to consider a mechanism that generates a program dependence graph similar to FCSG from the released malware source code and automatically extracts the control flow corresponding to the characteristic function. If signatures can be automatically generated, the practicality of this method will increase. As the next task, we would like to improve the signature matching method. At present, signature matching is performed when the node name and shape of FCSG exactly match, but if the function or function name changes slightly while maintaining the attack function, signature matching becomes impossible. In order to avoid this issue, it is conceivable to vectorize the signature-FCSG and the FCSG obtained from the specimen more abstractly by using graph embedding, and to perform signature matching for the parts with high similarity. The third challenge is to expand the range of malware binaries that can be analyzed. Currently, FCSG cannot be extracted from stripped binaries with debug symbol information removed, obfuscated binaries, and binaries that cannot be parsed by IDA Pro [71]. FCSG can be analyzed if the control flow and function name are known by disassembling, so we would like to devise ways to analyze the above unanalyzable binaries in the future. Finally, we aim to study evolutionary relationships of malware variants by utilizing a phylogenetic tree of malware using the hierarchical clustering method [84] and combining it with FCSG. Malware specimens that are close to each other in the phylogenetic

tree are expected to have similar behaviors.

References

- [1] M. Burhan, R. Rehman, B. Khan, and B. Kim, “IoT elements, layered architectures and security issues: A comprehensive survey,” *Sensors*, vol. 18, no. 9, p. 2796, Aug. 2018, <https://doi.org/10.3390/s18092796>.
- [2] Cybersecurity Laboratory, National Institute of Information and Communications Technology, “NICTER analysis report 2019,” 2019, https://www.nict.go.jp/cyber/report/NICTER_report_2019.pdf [Visited on Dec. 01, 2020].
- [3] M. Antonakakis, T. April, M. Bailey, M. Bernhard, E. Bursztein, J. Cochran, Z. Durumeric, J. A. Halderman, L. Invernizzi, M. Kallitsis, D. Kumar, C. Lever, Z. Ma, J. Mason, D. Menscher, C. Seaman, N. Sullivan, K. Thomas, and Y. Zhou, “Understanding the Mirai botnet,” in *26th USENIX Security Symposium*, 2017, pp. 1093–1110.
- [4] D. Goodin, “Hackers infect 500,000 consumer routers all over the world with malware: VPNFilter can survive reboots and contains destructive kill function,” 2018, <https://arstechnica.com/information-technology/2018/05/hackers-infect-500000-consumer-routers-all-over-the-world-with-malware/> [Visited on Dec. 01, 2020].
- [5] A. Štěda and J. Neduchal, “Hide ’N Seek botnet continues infecting devices with default credentials, building a P2P network and more,” 2018, <https://blog.avast.com/hide-n-seek-botnet-continues> [Visited on Dec. 01, 2020].
- [6] Y. Xiao, B. Chen, C. Yu, Z. Xu, Z. Yuan, F. Li, B. Liu, Y. Liu, W. Huo, W. Zou, and

- W. Shi, “MVP: detecting vulnerabilities using patch-enhanced vulnerability signatures,” in *29th USENIX Security Symposium*, 2020, pp. 1165–1182.
- [7] J. Takeuchi and K. Yamanishi, “A unifying framework for detecting outliers and change points from time series,” *IEEE Trans. Knowl. Data Eng.*, vol. 18, no. 4, pp. 482–492, 2006.
- [8] D. Inoue, K. Yoshioka, M. Eto, M. Yamagata, E. Nishino, J. Takeuchi, K. Ohkouchi, and K. Nakao, “An incident analysis system NICTER and its analysis engines based on data mining techniques,” in *Neural Information Processing - 15th International Conference (ICONIP)*, vol. 5506, 2008, pp. 579–586.
- [9] M. Bailey, E. Cooke, F. Jahanian, A. Myrick, and S. Sinha, “Practical darknet measurement,” in *40th Annual Conference on Information Sciences and Systems*, 2006, pp. 1496–1501.
- [10] G. Gu, J. Zhang, and W. Lee, “BotSniffer: detecting botnet command and control channels in network traffic,” in *Proceedings of the Network and Distributed System Security Symposium (NDSS)*, 2008.
- [11] M. Akiyama, T. Kawamoto, M. Shimamura, T. Yokoyama, Y. Kadobayashi, and S. Yamaguchi, “A proposal of metrics for botnet detection based on its cooperative behavior,” in *2007 International Symposium on Applications and the Internet - Workshops (SAINT)*, 2007, <https://doi.org/10.1109/SAINT-W.2007.14>.
- [12] C. Han, J. Shimamura, T. Takahashi, D. Inoue, J. Takeuchi, and K. Nakao, “Real-time detection of global cyberthreat based on darknet by estimating anomalous synchronization using graphical lasso,” *IEICE Transactions on Information and Systems*, vol. E103.D, no. 10, pp. 2113–2124, Oct. 2020, <https://doi.org/10.1587/transinf.2020edp7076>.
- [13] C. Han, J. Shimamura, T. Takahashi, D. Inoue, M. Kawakita, J. Takeuchi, and K. Nakao, “Real-time detection of malware activities by analyzing darknet traffic using graphical

- lasso,” in *18th IEEE International Conference On Trust, Security And Privacy In Computing And Communications (TrustCom)*, 2019, pp. 144–151, <https://doi.org/10.1109/TrustCom/BigDataSE.2019.00028>.
- [14] C. Han, K. Kono, S. Tanaka, M. Kawakita, and J. Takeuchi, “Botnet detection using graphical lasso with graph density,” in *Neural Information Processing - 23th International Conference (ICONIP)*, 2016, pp. 537–545, https://doi.org/10.1007/978-3-319-46687-3_59.
- [15] J. Friedman, T. Hastie, and R. Tibshirani, “Sparse inverse covariance estimation with the graphical lasso,” *Biostatistics*, vol. 9, no. 3, pp. 432–441, 2007, <https://doi.org/10.1093/biostatistics/kxm045>.
- [16] S. Yamauchi, M. Kawakita, and J. Takeuchi, “Botnet detection based on non-negative matrix factorization and the MDL principle,” in *Neural Information Processing - 19th International Conference (ICONIP)*, vol. 7667, 2012, pp. 400–409.
- [17] D. Lee and H. S. Seung, “Algorithms for non-negative matrix factorization,” in *Proceedings of the 13th International Conference on Neural Information Processing Systems (NIPS)*, 2000, p. 535–541.
- [18] R. Kawasoe, C. Han, R. Isawa, T. Takahashi, and J. Takeuchi, “Investigating behavioral differences between IoT malware via function call sequence graphs,” in *Proceedings of the 36th ACM/SIGAPP Symposium on Applied Computing*. ACM, 2021.
- [19] Cybersecurity Laboratory, National Institute of Information and Communications Technology, “NICTER analysis report 2016,” 2016, http://www.nict.go.jp/en/cyber/4otfsk00003rpx3e-att/NICTER_report_2016.pdf [Visited on Dec. 01, 2020].
- [20] —, “NICTERWEB,” <https://www.nicter.jp/> [Visited on Dec. 01, 2020].
- [21] M. Bailey, E. Cooke, F. Jahanian, and J. Nazario, “The internet motion sensor - A distributed blackhole monitoring system,” in *Proceedings of the Network and Distributed System Security Symposium (NDSS)*, 2005, pp. 167–179.

- [22] S. Choi, J. Song, S. Kim, and S. Kim, “A model of analyzing cyber threats trend and tracing potential attackers based on darknet traffic,” *Security and Communication Networks*, vol. 7, no. 10, pp. 1612–1621, May 2013, <https://doi.org/10.1002/sec.796>.
- [23] K. Nakao, D. Inoue, M. Eto, and K. Yoshioka, “Practical correlation analysis between scan and malware profiles against zero-day attacks based on darknet monitoring,” *IEICE Transactions on Information and Systems*, vol. E92-D, no. 5, pp. 787–798, 2009, <https://doi.org/10.1587/transinf.e92.d.787>.
- [24] S. H. A. Ali, S. Ozawa, T. Ban, J. Nakazato, and J. Shimamura, “A neural network model for detecting DDoS attacks using darknet traffic features,” in *2016 International Joint Conference on Neural Networks (IJCNN)*. IEEE, Jul. 2016, <https://doi.org/10.1109/ijcnn.2016.7727577>.
- [25] T. Ban, L. Zhu, J. Shimamura, S. Pang, D. Inoue, and K. Nakao, “Detection of botnet activities through the lens of a large-scale darknet,” in *Neural Information Processing*. Springer International Publishing, 2017, pp. 442–451, https://doi.org/10.1007/978-3-319-70139-4_45.
- [26] R. Tanabe, T. Tamai, A. Fujita, R. Isawa, K. Yoshioka, T. Matsumoto, C. Gañán, and M. van Eeten, “Disposable botnets: examining the anatomy of IoT botnet infrastructure,” in *The 15th International Conference on Availability, Reliability and Security (ARES)*, 2020, pp. 7:1–7:10.
- [27] S. García, A. Zunino, and M. Campo, “Survey on network-based botnet detection methods,” *Security and Communication Networks*, vol. 7, no. 5, pp. 878–903, Jun. 2013, <https://doi.org/10.1002/sec.800>.
- [28] M. Feily, A. Shahrestani, and S. Ramadass, “A survey of botnet and botnet detection,” in *2009 Third International Conference on Emerging Security Information, Systems and Technologies*. IEEE, 2009, <https://doi.org/10.1109/securware.2009.48>.

- [29] G. Gu, R. Perdisci, J. Zhang, and W. Lee, “Botminer: Clustering analysis of network traffic for protocol- and structure-independent botnet detection,” in *Proceedings of the 17th USENIX Security Symposium*, 2008, pp. 139–154.
- [30] W. Strayer, R. Walsh, C. Livadas, and D. Lapsley, “Detecting botnets with tight command and control,” in *Proceedings. 2006 31st IEEE Conference on Local Computer Networks*. IEEE, Nov. 2006, <https://doi.org/10.1109/lcn.2006.322100>.
- [31] S. García, A. Zunino, and M. Campo, “Botnet behavior detection using network synchronism,” in *Privacy, Intrusion Detection and Response*. IGI Global, pp. 122–144, <https://doi.org/10.4018/978-1-60960-836-1.ch005>.
- [32] X. Carreras and L. Màrquez, “Boosting trees for anti-spam email filtering,” *CoRR*, vol. cs.CL/0109015, 2001, <https://arxiv.org/abs/cs/0109015>.
- [33] A. L. Buczak and E. Guven, “A survey of data mining and machine learning methods for cyber security intrusion detection,” *IEEE Communications Surveys & Tutorials*, vol. 18, no. 2, pp. 1153–1176, 2016, <https://doi.org/10.1109/comst.2015.2494502>.
- [34] Y. Mirsky, T. Doitshman, Y. Elovici, and A. Shabtai, “Kitsune: An ensemble of autoencoders for online network intrusion detection,” in *25th Annual Network and Distributed System Security Symposium, (NDSS)*. The Internet Society, 2018.
- [35] W. U. Hassan, S. Guo, D. Li, Z. Chen, K. Jee, Z. Li, and A. Bates, “Nodoze: Combating threat alert fatigue with automated provenance triage,” in *26th Annual Network and Distributed System Security Symposium, (NDSS)*. The Internet Society, 2019.
- [36] J. Ma, L. K. Saul, S. Savage, and G. M. Voelker, “Learning to detect malicious URLs,” *ACM Transactions on Intelligent Systems and Technology*, vol. 2, no. 3, pp. 1–24, Apr. 2011, <https://doi.org/10.1145/1961189.1961202>.
- [37] T. Nelms, R. Perdisci, M. Antonakakis, and M. Ahamad, “Webwitness: Investigating, categorizing, and mitigating malware download paths,” in *24th USENIX Security Symposium*. USENIX Association, 2015, pp. 1025–1040.

- [38] ———, “Towards measuring and mitigating social engineering software download attacks,” in *25th USENIX Security Symposium*. USENIX Association, 2016, pp. 773–789.
- [39] Y. Fang, C. Huang, L. Liu, and M. Xue, “Research on malicious JavaScript detection technology based on LSTM,” *IEEE Access*, vol. 6, pp. 59 118–59 125, 2018, <https://doi.org/10.1109/access.2018.2874098>.
- [40] M. Sharif, J. Urakawa, N. Christin, A. Kubota, and A. Yamada, “Predicting impending exposure to malicious content from user behavior,” in *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*. ACM, Jan. 2018, <https://doi.org/10.1145/3243734.3243779>.
- [41] T. van Ede, R. Bortolameotti, A. Continella, J. Ren, D. J. Dubois, M. Lindorfer, D. R. Choffnes, M. van Steen, and A. Peter, “Flowprint: Semi-supervised mobile-app fingerprinting on encrypted network traffic,” in *27th Annual Network and Distributed System Security Symposium, NDSS*. The Internet Society, 2020.
- [42] X. Xu, C. Liu, Q. Feng, H. Yin, L. Song, and D. Song, “Neural network-based graph embedding for cross-platform binary code similarity detection,” in *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*. ACM, Oct. 2017, <https://doi.org/10.1145/3133956.3134018>.
- [43] F. Zuo, X. Li, P. Young, L. Luo, Q. Zeng, and Z. Zhang, “Neural machine translation inspired binary code similarity comparison beyond function pairs,” in *26th Annual Network and Distributed System Security Symposium, (NDSS)*. The Internet Society, 2019.
- [44] Z. L. Chua, S. Shen, P. Saxena, and Z. Liang, “Neural nets can learn function type signatures from binaries,” in *26th USENIX Security Symposium*. USENIX Association, 2017, pp. 99–116.
- [45] B. E. Strom, A. Applebaum, D. P. Miller, K. C. Nickels, A. G. Pennington, and C. B. Thomas, “Mitre att&ck: Design and philosophy,” *Technical report, MITRE Corporation*,

- 2018, <https://www.mitre.org/sites/default/files/publications/pr-18-0944-11-mitre-attack-design-and-philosophy.pdf> [Visited on Dec. 01, 2020].
- [46] M. Sebastián, R. Rivera, P. Kotzias, and J. Caballero, “AVclass: A tool for massive malware labeling,” in *Research in Attacks, Intrusions, and Defenses*, 2016, pp. 230–253.
- [47] A. Beaunon, P. Chifflier, and F. Bach, “ILAB: an interactive labelling strategy for intrusion detection,” in *Research in Attacks, Intrusions, and Defenses - 20th International Symposium (RAID)*, ser. Lecture Notes in Computer Science, vol. 10453. Springer, 2017, pp. 120–140.
- [48] V. Yegneswaran, P. Barford, and D. Plonka, “On the design and use of internet sinks for network abuse monitoring,” in *Recent Advances in Intrusion Detection: 7th International Symposium (RAID)*, vol. 3224, 2004, pp. 146–165, https://doi.org/10.1007/978-3-540-30143-1_8.
- [49] D. Moore, “Network telescopes: Tracking denial-of-service attacks and internet worms around the globe,” in *Proceedings of the 17th Conference on Systems Administration (LISA)*, 2003.
- [50] J. Friedman, T. Hastie, and R. Tibshirani, “Graphical lasso: Estimation of gaussian graphical models,” 2018, <https://cran.r-project.org/web/packages/glasso/glasso.pdf> [Visited on Dec. 01, 2020].
- [51] C. M. Bishop, *Pattern recognition and machine learning*, ser. Information science and statistics. Springer, 2006, <https://cds.cern.ch/record/998831> [Visited on Dec. 01, 2020].
- [52] T. Idé, A. C. Lozano, N. Abe, and Y. Liu, “Proximity-based anomaly detection using sparse structure learning,” in *Proceedings of the SIAM International Conference on Data Mining, SDM*, 2009, pp. 97–108, <https://doi.org/10.1137/1.9781611972795.9>.
- [53] A. P. Dempster, “Covariance selection,” *Biometrics*, vol. 28, no. 1, p. 157, Mar. 1972, <https://doi.org/10.2307/2528966>.

- [54] N. Meinshausen and P. Bühlmann, “High-dimensional graphs and variable selection with the lasso,” *The Annals of Statistics*, vol. 34, no. 3, pp. 1436–1462, Jun. 2006, <https://doi.org/10.1214/009053606000000281>.
- [55] O. Banerjee, L. E. Ghaoui, and A. d’Aspremont, “Model selection through sparse maximum likelihood estimation for multivariate gaussian or binary data,” *J. Mach. Learn. Res.*, vol. 9, pp. 485–516, 2008.
- [56] R. Tibshirani, “Regression shrinkage and selection via the lasso: a retrospective,” *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, vol. 73, no. 3, pp. 273–282, Apr. 2011, <https://doi.org/10.1111/j.1467-9868.2011.00771.x>.
- [57] C. Fachkha and M. Debbabi, “Darknet as a source of cyber intelligence: Survey, taxonomy, and characterization,” *IEEE Communications Surveys and Tutorials*, vol. 18, no. 2, pp. 1197–1227, 2016, <https://doi.org/10.1109/COMST.2015.2497690>.
- [58] A. Dainotti, K. Benson, A. King, k. claffy, M. Kallitsis, E. Glatz, and X. Dimitropoulos, “Estimating internet address space usage through passive measurements,” *SIGCOMM Comput. Commun. Rev.*, vol. 44, no. 1, pp. 42–49, 2014, <https://doi.org/10.1145/2567561.2567568>.
- [59] Z. Durumeric, M. Bailey, and J. A. Halderman, “An internet-wide view of internet-wide scanning,” in *Proceedings of the 23rd USENIX Security Symposium*, 2014, pp. 65–78.
- [60] C. Fachkha, E. Bou-Harb, A. Keliris, N. D. Memon, and M. Ahamad, “Internet-scale probing of CPS: inference, characterization and orchestration analysis,” in *24th Annual Network and Distributed System Security Symposium (NDSS)*, 2017.
- [61] E. Ahmed, A. J. Clark, and G. M. Mohay, “A novel sliding window based change detection algorithm for asymmetric traffic,” in *IFIP International Conference on Network and Parallel Computing (NPC)*, 2008, pp. 168–175, <https://doi.org/10.1109/NPC.2008.81>.
- [62] FOX-IT, “Recent vulnerability in Eir D1000 router used to spread updated version of Mirai DDoS bot,” 2016, <https://blog.fox-it.com/2016/11/28/recent-vulnerability-in-eir->

- d1000-router-used-to-spread-updated-version-of-mirai-ddos-bot/ [Visited on Dec. 01, 2020].
- [63] S. Herwig, K. Harvey, G. Hughey, R. Roberts, and D. Levin, “Measurement and analysis of Hajime, a peer-to-peer IoT botnet,” in *26th Annual Network and Distributed System Security Symposium (NDSS)*, 2019.
- [64] Netlab 360, “HNS botnet recent activities,” 2018, <https://blog.netlab.360.com/hns-botnet-recent-activities-en/> [Visited on Dec. 01, 2020].
- [65] —, “7,500+ MikroTik routers are forwarding owners’ traffic to the attackers, how is yours?” 2018, <https://blog.netlab.360.com/7500-mikrotik-routers-are-forwarding-owners-traffic-to-the-attackers-how-is-yours-en/> [Visited on Dec. 01, 2020].
- [66] Huawei, “Security notice - statement on remote code execution vulnerability in Huawei HG532 product,” 2017, <https://www.huawei.com/en/psirt/security-notices/huawei-sn-20171130-01-hg532-en> [Visited on Dec. 01, 2020].
- [67] Tenable Research, “Tenable research advisory: Peekaboo critical vulnerability in nuuo network video recorder,” 2018, <https://www.tenable.com/blog/tenable-research-advisory-peekaboo-critical-vulnerability-in-nuuo-network-video-recorder> [Visited on Dec. 01, 2020].
- [68] Netlab 360, “BCMPUPnP Hunter: a 100k botnet turns home routers to email spammers,” 2018, https://blog.netlab.360.com/bcmpupnp_hunter-a-100k-botnet-turns-home-routers-to-email-spammers-en/ [Visited on Dec. 01, 2020].
- [69] McAfee Labs, “McAfee Labs COVID-19 threats report, july 2020,” 2020, <https://www.mcafee.com/enterprise/en-us/assets/reports/rp-quarterly-threats-july-2020.pdf> [Visited on Dec. 01, 2020].
- [70] K. Iwamoto and K. Wasaki, “Malware classification based on extracted API sequences using static analysis,” in *Proceedings of the Asian Internet Engineering Conference*, ser. AINTEC ’12, 2012, p. 31–38.

- [71] Hex-Rays SA, “IDA Pro – Hex Rays,” 2020, <https://www.hex-rays.com/products/ida/> [Visited on Dec. 01, 2020].
- [72] Sri Shaila G, A. Darki, M. Faloutsos, N. Abu-Ghazaleh, and M. Sridharan, “IDAPro for IoT malware analysis?” in *12th USENIX Workshop on Cyber Security Experimentation and Test (CSET 19)*, Aug. 2019.
- [73] J. Gamblin, “GitHub - jgamblin/Mirai-Source-Code: leaked Mirai source code for research/ioc development purposes,” 2019, <https://github.com/jgamblin/Mirai-Source-Code> [Visited on Dec. 01, 2020].
- [74] F. Ding, “IoT Malware,” 2017, <https://github.com/ifding/iot-malware> [Visited on Dec. 01, 2020].
- [75] A. Hagberg, P. Swart, and D. S Chult, “Exploring network structure, dynamics, and function using NetworkX,” Los Alamos National Lab.(LANL), Los Alamos, NM (United States), Tech. Rep., 2008.
- [76] JM Project, “JM Index (LDP man-pages) (Japanese),” 2019, <https://linuxjm.osdn.jp/INDEX/ldp.html> [Visited on Dec. 01, 2020].
- [77] L. P. Cordella, P. Foggia, C. Sansone, and M. Vento, “A (sub)graph isomorphism algorithm for matching large graphs,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 26, no. 10, pp. 1367–1372, 2004.
- [78] Y. M. P. Pa, S. Suzuki, K. Yoshioka, T. Matsumoto, T. Kasama, and C. Rossow, “IoT-POT: Analysing the rise of IoT compromises,” in *9th USENIX Workshop on Offensive Technologies (WOOT 15)*, Aug. 2015.
- [79] M. Kerrisk, “readelf(1) - Linux manual page,” 2019, <http://man7.org/linux/man-pages/man1/readelf.1.html> [Visited on Dec. 01, 2020].
- [80] VirusTotal, “VirusTotal,” 2020, <https://www.virustotal.com/gui/> [Visited on Dec. 01, 2020].

- [81] H. Kikuchi and T. Okubo, “Malware classification method using common API call graph,” *Symposium on Cryptography and Information Security*, 2019.
- [82] Q. Feng, R. Zhou, C. Xu, Y. Cheng, B. Testa, and H. Yin, “Scalable graph-based bug search for firmware images,” ser. CCS ’16, 2016, p. 480–491.
- [83] V. S. Sathyanarayan, P. Kohli, and B. Bruhadeshwar, “Signature generation and detection of malware families,” in *Information Security and Privacy*, 2008, pp. 336–349.
- [84] T. He, C. Han, R. Isawa, T. Takahashi, S. Kijima, J. Takeuchi, and K. Nakao, “A fast algorithm for constructing phylogenetic trees with application to IoT malware clustering,” in *Neural Information Processing*. Springer International Publishing, 2019, pp. 766–778, https://doi.org/10.1007/978-3-030-36708-4_63.

Publications

Refereed Journal Papers

1. C. Han, J. Shimamura, T. Takahashi, D. Inoue, J. Takeuchi, and K. Nakao, "Real-time Detection of Global Cyberthreat Based on Darknet by Estimating Anomalous Synchronization Using Graphical Lasso," *IEICE Transactions on Information and Systems*, Vol.E103-D, No.10, pp.2113-2124, Oct. 2020.

Refereed International Conference Papers

2. [Under review] C. Han, J. Takeuchi, T. Takahashi, and D. Inoue, "Real-time Detection of Malware Activities on the Internet Using NMF," *IEEE International Conference on Communications (ICC)*, 2021.
3. R. Kawasoe, C. Han, R. Isawa, T. Takahashi, and J. Takeuchi, "Investigating Behavioral Differences between IoT Malware via Function Call Sequence Graphs," *Proceedings of the 36th ACM/SIGAPP Symposium on Applied Computing (SAC)*, 2021.
4. T. He, C. Han, R. Isawa, T. Takahashi, S. Kijima, J. Takeuchi, and K. Nakao, "A Fast Algorithm for Constructing Phylogenetic Trees with Application to IoT Malware Clustering," *Neural Information Processing - 26rd International Conference (ICONIP)*, 2019.
5. C. Han, J. Shimamura, T. Takahashi, D. Inoue, M. Kawakita, J. Takeuchi, and K. Nakao, "Real-Time Detection of Malware Activities by Analyzing Darknet Traffic Using Graph-

ical Lasso,” *IEEE International Conference on Trust, Security and Privacy in Computing and Communications (TrustCom): Security Track*, 2019.

6. C. Han, K. Kono, M. Kawakita, and J. Takeuchi, “Botnet Detection Using Graphical Lasso with Graph Density,” *Neural Information Processing - 23rd International Conference (ICONIP)*, vol.9947, pp.537-545, 2016.

Non-refereed Academic Magazine (Japanese)

7. T. Takahashi, K. Furumoto, and C. Han, “Cyber Wars:2. Boosting Cybersecurity with Machine Learning Techniques,” *IPSJ Magazine*, vol.61(7), pp.672-677, 2020.

Non-refereed Domestic Conference Papers (Japanese)

8. C. Han, J. Takeuchi, T. Takahashi, and D. Inoue, “Evaluation of Malware Activity Detection Method Using Nonnegative Matrix Factorization,” *Symposium on Cryptography and Information Security (SCIS)*, 2021.
9. R. Kawasoe, C. Han, R. Isawa, T. Takahashi, and J. Takeuchi, “A Feasibility Study on Investigating Functional Differences between IoT Malware,” *Computer Security Symposium (CSS)*, 2020.
10. R. Kawasoe, C. Han, R. Isawa, T. Takahashi, and J. Takeuchi, “A Study on Function Estimation of IoT Malware Focusing on Function Call Sequence,” *Symposium on Cryptography and Information Security (SCIS)*, 2020.
11. T. He, C. Han, R. Isawa, T. Takahashi, S. Kijima, J. Takeuchi, and K. Nakao, “A Fast Algorithm for Constructing Phylogenetic Trees and Its Evaluation,” *IEICE Tech. Rep.*, vol.119, no.288, ICSS2019-61, pp.7-12, 2019.

12. Y. Nakagawa, C. Han, J. Shimamura, T. Takahashi, A. Fujita, K. Yoshioka, and D. Inoue, "An Investigation of Constant Internet-Wide Scanning through Analysis of Darknet Traffic," *Computer Security Symposium (CSS)*, 2019.
13. C. Han, J. Shimamura, T. Takahashi, D. Inoue, J. Takeuchi, and K. Nakao, "Evaluation of Cyber Attack Detection Method based on Darknet Traffic Analysis," *Symposium on Cryptography and Information Security (SCIS)*, 2019.
14. C. Han, J. Shimamura, T. Takahashi, D. Inoue, M. Kawakita, J. Takeuchi, and K. Nakao, "A Study on Malware Activity Detection Based on Real-time Analysis of Darknet Data Using Graphical Lasso," *IEICE Tech. Rep., vol.117, no.481, ICSS2017-51*, pp.1-6, 2018.
15. C. Han, S. Matsumoto, J. Kawamoto, and K. Sakurai, "Classified by the API call record and personal information leakage detection of Android malware," *Hinokuni - Land of Fire Information Processing Symposium, 3A-4*, 2016.

Non-refereed Oral Presentation

16. Chansu Han, "Real-time Detection of Anomalous Cooperation based on Darknet using Graphical Lasso," *The 1st International Joint Conference on AI & Data Science*, Suwon, Korea, Nov. 2019.