PAPER

Real-Time Detection of Global Cyberthreat Based on Darknet by Estimating Anomalous Synchronization Using Graphical Lasso[*]

Chansu HAN[†,††a)], ***Member***, **Jumpei SHIMAMURA**[†††], ***Nonmember***, **Takeshi TAKAHASHI**[†], **Daisuke INOUE**[†], **Jun'ichi TAKEUCHI**[††], ***Members***, ***and*** **Koji NAKAO**[†], ***Fellow***

SUMMARY With the rapid evolution and increase of cyberthreats in recent years, it is necessary to detect and understand it promptly and precisely to reduce the impact of cyberthreats. A darknet, which is an unused IP address space, has a high signal-to-noise ratio, so it is easier to understand the global tendency of malicious traffic in cyberspace than other observation networks. In this paper, we aim to capture global cyberthreats in real time. Since multiple hosts infected with similar malware tend to perform similar behavior, we propose a system 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. In our evaluation, we perform our proof-of-concept implementation of the proposed engine to demonstrate its feasibility and effectiveness, and we detect cyberthreats with an accuracy of 97.14%. This work is 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.

key words: *cyberthreat, malware, darknet, network security, synchronization, outlier detection, real-time detection*

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 scanning 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 [2], [3], because there are no legitimate hosts there. Other than misconfiguration, packets destined for the darknet are almost always malicious [4]. In general, it is suitable for monitoring explosions, not small events [5]. 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 [3], 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 [6]. 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 the *GLASSO engine*[**], which estimates a synchronization of the source hosts in real time by using a sparse structure learning algorithm called the *graphical lasso* [8]. 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

Manuscript received April 3, 2020.
Manuscript revised June 5, 2020.
Manuscript publicized June 25, 2020.

[†]The authors are with National Institute of Information and Communications Technology, Koganei-shi, 184–8795 Japan.
[††]The authors are with Kyushu University, Fukuoka-shi, 819–0395 Japan.
[†††]The author is with clwit Inc., Tokyo, 140–0001 Japan.

[*]This paper was partly presented at the 18th IEEE International Conference on Trust, Security and Privacy in Computing and Communications (TrustCom), 2019 [1]. Some experiments have been updated in this paper.

a) E-mail: han@nict.go.jp

DOI: 10.1587/transinf.2020EDP7076

[**]The engine is named after the library "`glasso`" of `R` [7].

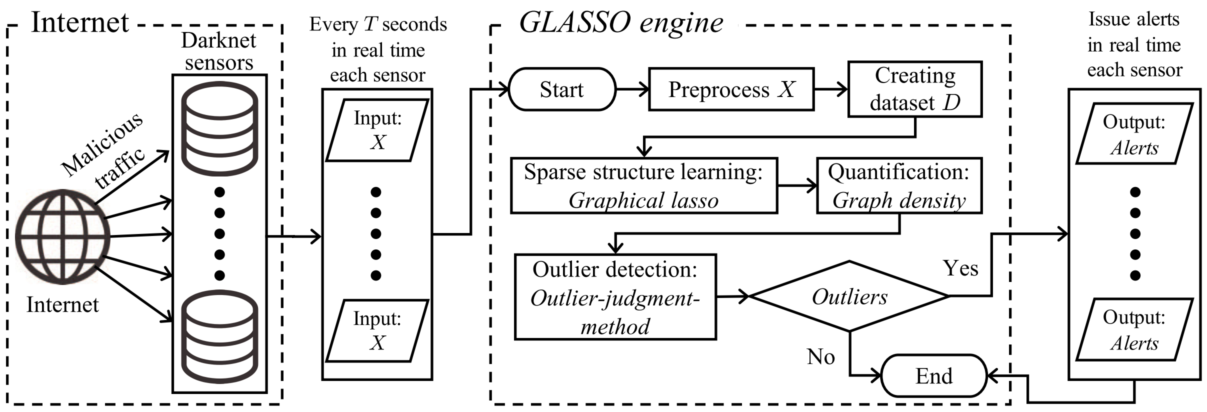
Internet
Darknet sensors
Malicious traffic
Internet
Every T seconds in real time each sensor
Input: X
Input: X
GLASSO engine
Start
Preprocess X
Creating dataset D
Sparse structure learning: Graphical lasso
Quantification: Graph density
Outlier detection: Outlier-judgment-method
Outliers
Yes
No
End
Issue alerts in real time each sensor
Output: Alerts
Output: Alerts

Fig. 1 An overview of the *GLASSO engine*

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 the *GLASSO engine* with a demonstration 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 paper, 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 the *GLASSO engine* using the list as the ground truth. As a result, we confirm that the *GLASSO engine* can detect cyberthreats with somewhat massive host-scale infection with high accuracy, and which shows its effectiveness and usefulness. In other words, the *GLASSO engine* can successfully identify global cyberthreats in real time in the darknet that could lead to serious security incidents.

This paper offers the following contributions:

1. We propose the *GLASSO engine*, as shown in Fig. 1, which statistically estimates a synchronization between the source hosts from traffic data and detects anomalies. (Discussed in Sect. 3)
2. We perform a demonstration experiment of the *GLASSO engine* 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 Sect. 4)
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 Sect. 5)

This paper 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 Sect. 2 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 the *GLASSO engine*. In addition, we publish the datasets used for demonstration experiment in this paper on the web[†].

2. Related Work

In this section, we review previous research about botnet detection, darknet data analysis, including our achievement so far. The BotSniffer [9] and the BotMiner [10] adopt C&C traffic detection method based on the spatial-temporal correlation approach, which is a similar approach to this paper. 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, the *GLASSO engine* 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,

[†]https://csdataset.nict.go.jp/darknet

and the scale of observed scans is overwhelmingly large. The *GLASSO engine* 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 [11]. In [2]–[5], 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 [12]. Durumeric et al. analyzed the large-scale darknet to investigate scanning activities and identify patterns in sizable horizontal scanning operations [13]. 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 [14]. 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 [11], 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 [15]–[17]. In Network Incident analysis Center for Tactical Emergency Response (`NICTER`) project [15] conducted by NICT, they used a ChangeFinder algorithm [18] 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 [15]–[17]. We have started to study the *GLASSO engine*, which is a method to detect cyberthreats using an approach that can avoid the problem of change-point detection methods, as mentioned above. The *GLASSO engine* detects anomalies based on more abundant information, such as the synchronization of source hosts, than the change point detection method. In addition, the *GLASSO engine* 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. *GLASSO Engine*

In this section, we describe the *GLASSO engine* 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, the *GLASSO engine* can automatically detect not only known cyberthreats but also zero-day cyberthreats quickly and accurately.

Figure 1 shows an overview of the *GLASSO engine*, and Algorithm 2 shows the pseudocode of the *GLASSO engine*. First, we prepare T seconds of traffic data X every darknet sensor. Here, the *GLASSO engine* can analyze regardless of a protocol, but this paper 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 the *GLASSO engine*, preprocess traffic data according to the situation[†], 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 the *GLASSO engine* in real time is introduced.

3.1 Creating Dataset D

The *GLASSO engine* 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 Appendix A 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

[†]Detailed settings for preprocessing are described in Sect. 4.1.

Algorithm 1 *outlier-judgment-method*

```
Input: $\boldsymbol{d}^{(r)} \in \mathbb{R}^K,\ K,\ \theta$
Output: outliers or none
 1: $i,\ j \leftarrow 0$
 2: while TRUE do
 3:     $i \leftarrow i + 1$
 4:     $\boldsymbol{d}_{(i)} \leftarrow \mathrm{order}(\boldsymbol{d}^{(r)})[i : K]$
 5:     $\sigma^2_{(i)} \leftarrow \mathrm{var}(\boldsymbol{d}_{(i)})$
 6:     if $\sigma^2_{(i+1)}/\sigma^2_{(i)} < \theta$ then
 7:         $j \leftarrow j + 1$
 8:     else if $i = 1$ then
 9:         break
10:     else
11:         outliers $\leftarrow \mathrm{order}(\boldsymbol{d}^{(r)})[1 : j]$
12:         return outliers
13:     end if
14: end while
```

T/M (sec.). We then convert from the X to a dataset

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

Here, $\boldsymbol{x}^{(m)}$ 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, $\mathbb{N}_0 = \{0, 1, 2, \cdots\}$.

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 (Appendix B), one of a sparse structure learning algorithm, for estimating dependencies and synchronizations among variables from the dataset D.

3.2 Sparse Structure Learning: Graphical Lasso

As described in Appendix A, 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 Appendix B 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 $r \in R$ $(= \{r_1, r_2, \cdots, r_s\} \subset [0, \infty))$ 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.

Algorithm 2 The *GLASSO engine* with online processing

```
Input: $X,\ \beta,\ M,\ r \in R\ (= \{r_1, r_2, \cdots, r_s\}), \boldsymbol{d}^{(r)},\ K,\ \theta$
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 $\boldsymbol{d}^{(r)}$
 7:         if length$(\boldsymbol{d}^{(r)}) = K$ then
 8:             call outlier-judgment-method$(\boldsymbol{d}^{(r)},\ K,\ \theta)$
 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 $\boldsymbol{d}^{(r)}$
13:             else
14:                 delete the oldest value from $\boldsymbol{d}^{(r)}$
15:             end if
16:         end if
17:     end for
18: end while
```

3.3 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 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 K graph density values $\boldsymbol{d}^{(r)}$ for each parameter r. Next, we check the extent to which the largest element occupies a total sample variance in $\boldsymbol{d}^{(r)}$, 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, $\sigma^2_{(i+1)}/\sigma^2_{(i)} < \theta$ is the outlier judgment formula, and θ $(0 \leq \theta \leq 1)$ is a threshold value. In addition, the order( ) function returns a permutation that rearranges its first argument into descending order, and the 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.

3.5 Procedure for Online Processing

The pseudocode for the *GLASSO engine* with online processing is described in Algorithm 2. Here, the 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 $\boldsymbol{d}^{(r)}$, and if the length of $\boldsymbol{d}^{(r)}$ 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 $\boldsymbol{d}^{(r)}$ 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 $\boldsymbol{d}^{(r)}$. When the time has passed and a new X is updated, by repeating the above steps, it is possible to process the *GLASSO engine* sequentially while maintaining the number of datasets used for outlier detection. From the above, the *GLASSO engine* 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.

4. Performance Evaluation

The alert issued by the *GLASSO engine* 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 the *GLASSO engine*, and evaluate a quantitative detection accuracy of cyberthreats. Here, the conventional method, ChangeFinder [18], is also tried and evaluated comparatively with our method.

4.1 Demonstration Experiment

In this section, we present the process of demonstrating the feasibility of the *GLASSO engine*. We implemented a prototype of the *GLASSO engine* using R language and a Linux shell running on two machines, a 16 GB of memory and a 3.3GHz Intel-Xenon-E3-1230. We used eight different darknet sensors in parallel and operated the *GLASSO engine* for one month in October 2018 in real time for each sensor. The input parameters of the *GLASSO engine* used for the operation were set to $T = 600$ (sec.), $M = 12$, $K = 432$, $\theta = 0.98$, $r \in R$ (= {0.4, 0.5, 0.6, 0.7, 0.8, 0.9}); details of this parameter selection can be found in the paper [1].

Next, we describe the preprocessing of darknet traffic performed in this demonstration experiment. In order to stabilize the performance of the *GLASSO engine*, 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 the *GLASSO engine*, 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 Sect. 5.1.

4.2 Analysis of Alert Result

Table 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 "*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

Table 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
GLASSO (ALL)	80,630	1,293	152

addresses that are considered to have been used as cyberthreats in the time slots determined to be outliers by the *GLASSO engine*. 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 paper, 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 Sect. 5.2. 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 type 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 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.

Table 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

4.3 Detection Performance

In this section, we quantitatively evaluate the performance of the *GLASSO engine* in detecting cyberthreats and compare it with a conventional method, ChangeFinder. First, we prepared the ground truth for quantitative evaluation. Table 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 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 Sect. 4.4. 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 the *GLASSO engine* 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 [18]. This setting was determined based on our experience of using ChangeFinder for many years at the `NICTER` [15]. 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 Sect. 4.2, and obtained 25 unique ports.

Table 3 Evaluation results of the detection performance of the *GLASSO engine* and ChangeFinder

Sensor ID	#Ports	#Noises	#TPs	#FPs	#FNs	#TNs	Accuracy [%]	Precision [%]	Recall [%]	F-measure [%]
A	33	4	29	0	6	-	82.86	100.00	82.86	90.63
B	43	14	29	0	6	-	82.86	100.00	82.86	90.63
C	42	17	25	0	10	-	71.43	100.00	71.43	83.33
D	40	15	25	0	10	-	71.43	100.00	71.43	83.33
E	44	17	27	0	8	-	77.14	100.00	77.14	87.10
F	49	23	26	0	9	-	74.29	100.00	74.29	85.25
G	32	13	19	0	16	-	54.29	100.00	54.29	70.37
H	36	17	19	0	16	-	54.29	100.00	54.29	70.37
***GLASSO* (ALL)**	**152**	**118**	**34**	**0**	**1**	**-**	**97.14**	**100.00**	**97.14**	**98.55**
ChangeFinder (ALL)	**25**	**1**	**24**	**0**	**11**	**-**	**68.57**	**100.00**	**68.57**	**81.36**

Table 3 shows the evaluation results of the detection performance of the *GLASSO engine* and ChangeFinder. The evaluation of the *GLASSO engine* 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, the *GLASSO engine* 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 the *GLASSO engine*.

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. The *GLASSO engine* 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 the *GLASSO engine* overlooked, had overwhelmingly few source hosts compared to other ports. Our darknet analysts used the `NICTER`'s 300,000-scale darknet, but this time the *GLASSO engine* 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 the *GLASSO engine* and added to the ground truth (Table 2).

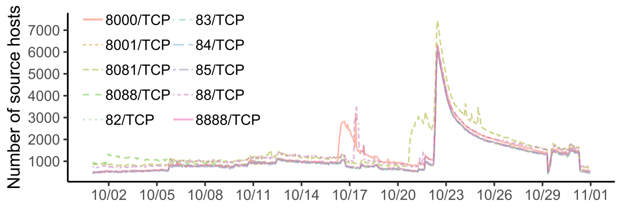

Fig. 2 The number of source hosts per one hour in October 2018 at TCP ports where `Mirai`-related cyberthreats were observed. (sensor A, part I)

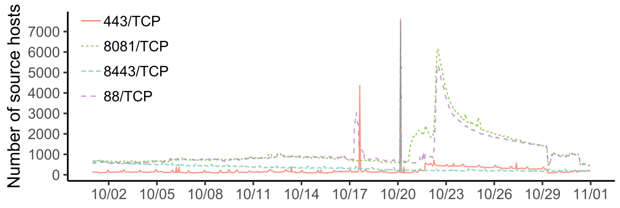

Fig. 3 The number of source hosts per one hour in October 2018 at TCP ports where `Mirai`-related cyberthreats were observed. (sensor B, part II)

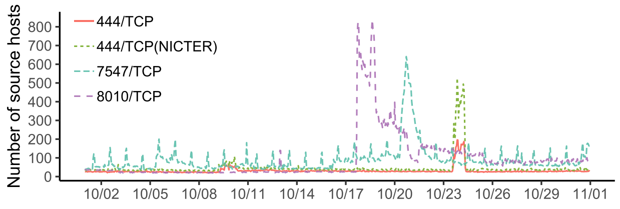

Fig. 4 The number of source hosts per one hour in October 2018 at TCP ports where `Mirai`-related cyberthreats were observed. (sensor A and `NICTER`, part III)

4.4 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 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. 2 to 7) 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 2. First, `Mirai` has a feature where a sequence

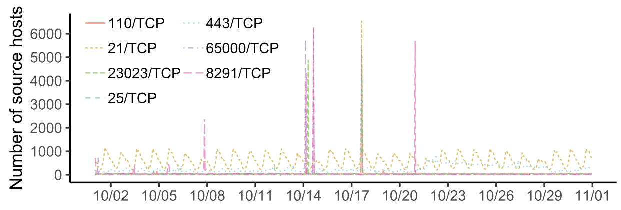

Fig. 5 The number of source hosts per one hour in October 2018 at TCP ports where router vulnerability-related cyberthreats were observed. (sensor A)

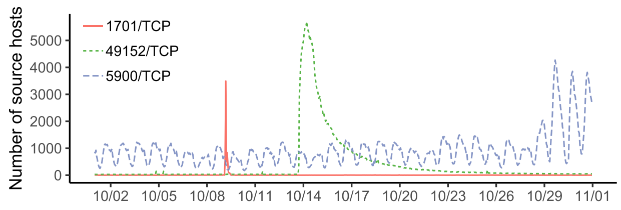

Fig. 6 The number of source hosts per one hour in October 2018 at TCP ports where other vulnerability-related cyberthreats were observed. (sensor A, part I)

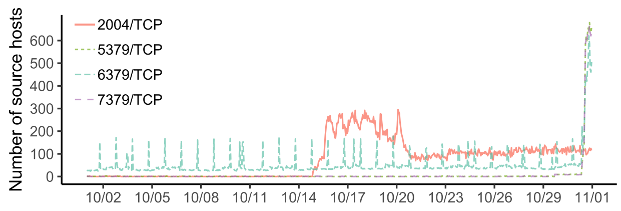

Fig. 7 The number of source hosts per one hour in October 2018 at TCP ports where other vulnerability-related cyberthreats were observed. (sensor A, part II)

number denotes a destination IP address converted to decimal in SYN packets. Figure 2, 3, and 4 show the TCP ports where such Mirai features were monitored. The 10 TCP ports in Fig. 2 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 Fig. 3, 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. 2 that many hosts originated in China and had a fixed window size of 14100. Figure 3 shows the results for sensor B because this threat could not be observed with sensor A.

Figure 4 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 [19]. The event towards 444/TCP, mainly by hosts from Greece on October 23, was the only threat that the *GLASSO engine* overlooked. As described in the previous 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 [20]. 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 [21].

Router Vulnerability: The 12 TCP ports of router vulnerability types in Table 2 were divided into threats against vulnerabilities presented in the router products from five manufacturers. Figure 5 shows multiple scans from devices that seem to be manufacturer A's router products, and they scanned the ports that are open by default [22]. The window size was fixed at 1024 in all cases.

Second, 37215/TCP was scanned from router products of manufacturer B [23], 8181/TCP was scanned from router products of manufacturer C [24], and 8001, 8081/TCP (Fig. 2) 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) [25].

Other Vulnerability: The seven TCP ports of the other vulnerability types in Table 2 were broken down into five threats to vulnerabilities of applications and services. The three TCP ports 1701, 49152, 5900 in Fig. 6 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 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. 7, 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, the *GLASSO engine* has shown the possibility of detecting zero-day threats at an appropriate time.

Table 4 Average performance when the *GLASSO engine* was run five times for each β, using a confusion matrix and five evaluation metrics

Sensor ID	β	#Unique Alert-times	#TPs	#FPs	#FNs	#TNs	Accuracy [%]	Precision [%]	Recall [%]	F-measure [%]	FPR [%]
C	300	49	4.8	5.6	13.2	984.4	98.13	46.15	26.67	33.80	0.57
	500	44	9.2	4.6	8.8	985.4	98.67	66.67	51.11	57.86	0.46
	600	20	14.6	1	3.4	989	99.56	93.59	81.11	86.90	0.10
	700	18	18	0	0	990	100.00	100.00	100.00	100.00	0.00
D	300	38	4.2	6.4	7.8	989.6	98.59	39.62	35.00	37.17	0.64
	500	31	8.4	7.2	3.6	988.8	98.93	53.85	70.00	60.87	0.72
	600	15	10.4	2.2	1.6	993.8	99.62	82.54	86.67	84.55	0.22
	700	14	12	1.2	0	994.8	99.88	90.91	100.00	95.24	0.12

5. Discussion

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

5.1 Limiting the Number of Source Host: β

Among the preprocessing of the *GLASSO engine* 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 the *GLASSO engine* five times, respectively. Here, the *GLASSO engine* 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.

Table 4 shows the average performance when the *GLASSO engine* 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 Table 3, but this time #TNs can be counted, so the accuracy is (#TPs + #TNs) / (#TPs + #FPs + #FNs + #TNs), and the false positive rate (FPR) is #FPs / (#FPs + #TNs). As a result, we can see from Table 4 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 5 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

Table 5 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

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 5, the β was set to 1100 to 1300, but in the experiment in Table 4, β was set to 300 to 700, and strict 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 Table 4 is 300. Moreover, looking at the detection accuracy of each sensor in Table 3, 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.

5.2 Limitation

We describe two limitations that were difficult to evaluate in this paper. 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 the *GLASSO engine* have also been observed in the honeypot will improve the reliability of the *GLASSO engine*. 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.

6. 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 *GLASSO engine* 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 the *GLASSO engine*. 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 paper are available online at https://csdataset.nict.go.jp/darknet.

Acknowledgements

The authors thank Associate Prof. Katsunari Yoshioka from Yokohama National University and Prof. Noboru Murata from Waseda University for their valuable comments. This research was partially supported by JSPS KAKENHI Grant Number 18H03291.

References

[1] 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," 18th IEEE International Conference On Trust, Security And Privacy In Computing And Communications (TrustCom), pp.144–151, IEEE, 2019.
[2] M. Bailey, E. Cooke, F. Jahanian, and J. Nazario, "The internet motion sensor - A distributed blackhole monitoring system," Proc. Network and Distributed System Security Symposium (NDSS), The Internet Society, pp.167–179, 2005.
[3] M. Bailey, E. Cooke, F. Jahanian, A. Myrick, and S. Sinha, "Practical darknet measurement," 40th Annual Conference on Information Sciences and Systems, pp.1496–1501, IEEE, 2006.
[4] V. Yegneswaran, P. Barford, and D. Plonka, "On the design and use of internet sinks for network abuse monitoring," Recent Advances in Intrusion Detection: 7th International Symposium (RAID), pp.146–165, Springer, 2004.
[5] D. Moore, "Network telescopes: Tracking denial-of-service attacks and internet worms around the globe," Proc. 17th Conference on Systems Administration (LISA), USENIX, 2003.
[6] 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," 2007 International Symposium on Applications and the Internet - Workshops (SAINT), p.82, IEEE, 2007.
[7] 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.
[8] J. Friedman, T. Hastie, and R. Tibshirani, "Sparse inverse covariance estimation with the graphical lasso," Biostatistics, vol.9, no.3, pp.432–441, 2007.
[9] G. Gu, J. Zhang, and W. Lee, "Botsniffer: Detecting botnet command and control channels in network traffic," Proc. Network and Distributed System Security Symposium (NDSS), The Internet Society, 2008.
[10] G. Gu, R. Perdisci, J. Zhang, and W. Lee, "Botminer: Clustering analysis of network traffic for protocol- and structure-independent botnet detection," Proc. 17th USENIX Security Symposium, pp.139–154, 2008.
[11] C. Fachkha and M. Debbabi, "Darknet as a source of cyber intelligence: Survey, taxonomy, and characterization," IEEE Commun. Surveys Tuts., vol.18, no.2, pp.1197–1227, 2016.
[12] 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.
[13] Z. Durumeric, M. Bailey, and J.A. Halderman, "An internet-wide view of internet-wide scanning," Proc. 23rd USENIX Security Symposium, pp.65–78, 2014.
[14] C. Fachkha, E. Bou-Harb, A. Keliris, N.D. Memon, and M. Ahamad, "Internet-scale probing of CPS: inference, characterization and orchestration analysis," 24th Annual Network and Distributed System Security Symposium (NDSS), The Internet Society, 2017.
[15] 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," Advances in Neuro-Information Processing, 15th International Conference (ICONIP), pp.579–586, Springer, 2008.

[16] 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," Neural Information Processing - 24th International Conference (ICONIP), pp.442–451, Springer, 2017.

[17] E. Ahmed, A.J. Clark, and G.M. Mohay, "A novel sliding window based change detection algorithm for asymmetric traffic," IFIP International Conference on Network and Parallel Computing (NPC), pp.168–175, IEEE Computer Society, 2008.

[18] 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.

[19] 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/.

[20] S. Herwig, K. Harvey, G. Hughey, R. Roberts, and D. Levin, "Measurement and analysis of hajime, a peer-to-peer iot botnet," 26th Annual Network and Distributed System Security Symposium (NDSS), The Internet Society, 2019.

[21] Netlab360, "Hns botnet recent activities," 2018. https://blog.netlab.360.com/hns-botnet-recent-activities-en/.

[22] Netlab360, "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/.

[23] 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.

[24] 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.

[25] Netlab360, "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/.

[26] T. Idé, A.C. Lozano, N. Abe, and Y. Liu, "Proximity-based anomaly detection using sparse structure learning," Proc. SIAM International Conference on Data Mining, SDM, pp.97–108, Society for Industrial and Applied Mathematics, 2009.

Appendix A: Graphical Gaussian Model

A graphical Gaussian model is a probabilistic model for which a graph expresses the dependence structure between random variables given a multivariate Gaussian distribution. To measure the dependency structure between random variables, there is a method of obtaining a precision matrix Σ^{-1} (the inverse of the covariance matrix Σ) from which the conditional independence of a pair of random variables is able to be estimated [26]. If and only if $(\Sigma^{-1})_{ij} = 0$, then x_i and x_j are independent, conditioned on all the other variables.

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 $V = \{x_1, \cdots, x_N\}$ of N random variables and edge set $E = \{(i, j) | (\hat{\Sigma}^{-1})_{ij} \neq 0\}$. In other words, the graph G shows the conditional independence of all pairs of random variables.

Appendix B: Graphical Lasso

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.

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 $r \sum_{ij} |(\Sigma^{-1})_{ij}|$ [8]. 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. 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.

Chansu Han received B.E. and M.S. degrees in informatics from Kyushu University in 2016 and 2018, respectively. He is now a Ph.D. student at the same university and a researcher at the National Institute of Information and Communications Technology (NICT), Japan. His research interests include analyzing and solving problems in the cybersecurity field (especially malware) using machine learning. He is a member of IEEE and IEICE.

Jumpei Shimamura graduated in electrical engineering from Shibaura Institute of Technology in 2003. He has been with the clwit Inc., since 2011. Since 2005, he has been engaged in the observation and analysis of unauthorized communications on the Internet, including the darknet.

Takeshi Takahashi received a Ph.D. degree in telecommunication from Waseda University in 2005. He worked at Tampere University of Technology as a researcher from 2002 to 2004, and at Roland Berger Ltd. as a business consultant from 2005 to 2009. Since 2009, he has been working at the National Institute of Information and Communications Technology, where he is currently a research manager. His research interests include cybersecurity and machine learning. He was a recipient of the Funai Information Technology Incentive Award in 2005, and ITUAJ's Incentive Award in 2012. He serves as the IETF MILE Working Group chair and ITU-T Q.6/17 Associate Rapporteur.

Daisuke Inoue received his B.E. and M.E. degrees in electrical and computer engineering and his Ph.D. in engineering from Yokohama National University in 1998, 2000 and 2003, respectively. He joined Communications Research Laboratory (CRL), Japan in 2003. CRL was relaunched as the National Institute of Information and Communications Technology (NICT) in 2004, and he is currently Director of its Cybersecurity Laboratory. He has received a number of awards, including a commendation for science and technology from the minister of MEXT, Japan in 2009, the Good Design Award 2013, the Asia-Pacific Information Security Leadership Achievements 2014, and the award for contribution to Industry-Academia-Government Collaboration by the minister of MIC, Japan in 2016.

Jun'ichi Takeuchi was born in Tokyo, Japan in 1964. He graduated from the University of Tokyo in 1989 with a major in physics. He received a Dr. Eng. degree in mathematical engineering from the University of Tokyo in 1996. From 1989 to 2006, he worked for NEC Corporation, Japan. In 2006, he joined Kyushu University, Fukuoka, Japan, where he is a professor of mathematical engineering. From 1996 to 1997 he was a visiting research scholar in the Department of Statistics, Yale University, New Haven, CT, USA. His research interests include mathematical statistics, information geometry, information theory, machine learning, and their applications. He is a member of IEEE, IEICE, IPSJ, and JSIAM.

Koji Nakao received a B.E. degree in mathematics from Waseda University in Japan in 1979. Since joining KDDI in 1979, Koji has been undertaking research on communication protocols and information security technology for telecommunications in KDDI's laboratory. He also began researching the area of cybersecurity for NICT (the National Institute of Information and Communications Technology) in 2004 and for Yokohama National University in 2015. His current positions are "distinguished researcher" to manage research activities for cybersecurity technologies at NICT and "guest professor" of Yokohama National University in IoT security research. Koji has also been a cybersecurity advisor for the cabinet secretariat in the Japanese government since April 2017. He is a member of IPJS and a fellow of IEICE.