FindEvasion: An Effective Environment-Sensitive Malware Detection System for the Cloud

Xiaoqi Jia[1,2,3,4], Guangzhe Zhou[1,2,3,4], Qingjia Huang[1,2,3,4](✉), Weijuan Zhang[1,2,3,4], and Donghai Tian[5]

[1] Institute of Information Engineering, CAS, Beijing, China
{jiaxiaoqi,zhouguangzhe,huangqingjia,zhangweijuan}@iie.ac.cn
[2] School of Cyber Security, University of Chinese Academy of Sciences, Beijing, China
[3] Key Laboratory of Network Assessment Technology, CAS, Beijing, China
[4] Beijing Key Laboratory of Network Security and Protection Technology, Beijing, China
[5] Beijing Key Laboratory of Software Security, Engineering Technique, Beijing Institute of Technology, Beijing, China

Abstract. In recent years, environment-sensitive malwares are growing rapidly and they pose significant threat to cloud platforms. They may maliciously occupy the computing resources and steal the tenants' private data. The environment-sensitive malware can identify the operating environment and perform corresponding malicious behaviors in different environments. This greatly increased the difficulty of detection. At present, the research on automatic detection of environment-sensitive malwares is still rare, but it has attracted more and more attention.

In this paper, we present FindEvasion, a cloud-oriented system for detecting environment-sensitive malware. Our FindEvasion system makes full use of the virtualization technology to transparently extract the suspicious programs from the tenants' Virtual Machine (VM), and analyzes them on our multiple operating environments. We introduce a novel algorithm, named Mulitiple Behavioral Sequences Similarity (MBSS), to compare a suspicious program's behavioral profiles observed in multiple analysis environments, and determine whether the suspicious program is an environment-sensitive malware or not. The experiment results show that our approach produces better detection results when compared with previous methods.

Keywords: Cloud security · Environment-sensitive malware · MBSS
Transparent extraction · Multiple operating environments

1 Introduction

In recent years, increasing malwares have gradually become an important threat to the construction of cloud computing. These malwares can not only occupy

© ICST Institute for Computer Sciences, Social Informatics and Telecommunications Engineering 2018
P. Matoušek and M. Schmiedecker (Eds.): ICDF2C 2017, LNICST 216, pp. 3–17, 2018.
https://doi.org/10.1007/978-3-319-73697-6_1

the computing resources maliciously, but also attack other tenants and even the underlying platform to steal the other tenants' private data. As more and more data with sensitve and high commercial value information is migrated to the Cloud, researchers paid more attention to the malware detection for the Cloud.

Among the various kinds of malwares, environment-sensitive malwares are growing rapidly. This kind of malware can identify the current operating environment and perform corresponding malicious behaviors in different environments. According to Symantec's security threat report [1], 20% of new malwares are environment-sensitive currently and the number of environment-sensitive malware is increasing at a rate of 10–15 per week.

In order to detect environment-sensitive malwares, some methods have been proposed gradually, such as BareCloud [2] and Disarm [3]. BareCloud is based on the bare-metal and only considers the operations that cause a persistent change to the system. This will lead to many meaningful non-persistent malicious operations being ignored, for example, the remote injection. Besides, BareCloud uses a Hierarchical similarity algorithm to compare the behavioral profiles, however, the detection ability of this algorithm will be greatly affected if the environment-sensitive malware performs a lot of independent interference behaviors. Disarm deploys two kinds of sandbox with different monitoring technologies. However, two kinds of environments are not enough to detect a variety of evasive behaviors within the environment-sensitive malware. Therefore, how to make the environment-sensitive malware exhibit the evasive behavior and cope with the interference behaviors is the key issue for the detection.

In this paper, we present FindEvasion, a cloud-oriented system for automatically detecting environment-sensitive malwares. The FindEvasion performs malware analysis on multiple operating environments, which include Sandbox environment, Debugging environment, Hypervisor environment and so on. In order to analyze the suspicious program running in the guest VM, we make use of the virtualization technology to transparently extract it from the guest VM and the suspicious program will not be awared of this whole process. We propose an algorithm to compare the suspicious program's behavioral profiles and determine whether it is an environment-sensitive malware or not.

Our work makes the following contributions:

– We present a system called FindEvasion for detecting environment-sensitive malwares. Our system makes full use of the virtualization technology to transparently extract the suspicious program from the guest VM, and then performs suspicious program analysis on multiple operating environments to make the environment-sensitive malware exhibit the evasive behavior.
– We introduce a novel evasion detection algorithm, named MBSS, for behavioral profiles comparsion. Our algorithm can cope with the interference behaviors to make the detection more effective.
– We present experimental evidence that demonstrates that the operations of eliminating interference behaviors are effective for detecting enviornment-sensitive malwares, and the recall rate is increased to 60% with 100% precision.

The rest of this paper is organized as follows. The next section presents the system architecture of FindEvasion. Section 3 shows the implementation in detail. In Sect. 4, we design four experiments for evaluating our system and MBSS algorithm. Finally, we discuss related work in Sect. 5 and conclude the paper in Sect. 6.

2 System Architecture

As Fig. 1 shows, the FindEvasion architecture consists of two parts. One is the Cloud service node, which provides service to the tenants. It contains an Extraction module in the VMM. The Extraction module can extract a suspicious program running in the guest VM and transfer it to the multiple environments analysis platform for analyzing. More details are provided in Sect. 3.1. The other is the multiple environments analysis platform, which includes Sandbox environment, VM environment, Hypervisor environment and debugging environment, etc. It contains an Environment-sensitive detection module, which can compare the behavioral profiles extracted from multiple analysis environments and make a judgment that whether the suspicious program is environment-sensitive malware or not. This is achieved by our MBSS algorithm.

The purpose of deploying multiple environments analysis platform is to identify the deviations in the behaviors of a suspicious program. That is, if a suspicious program is environment-sensitive, then it would have different behaviors obviously in a specific environment compared to the other environment. Besides, it is necessary to point out that the Hypervisor used in multiple environments analysis platform is modified particularly. It can not only transparently monitor a suspicious program based on the virtualization technology, but also avoid being detected by the malware. This can be achieved by some skills, for example cheating the guest. We insert a kernel module in the VM environment and debugging environment for monitoring. As for Sandbox environment, it contains own in-guest monitoring components. Various monitoring technologies can also help us to find the environment-sensitive malware that targets a specific monitoring technique.

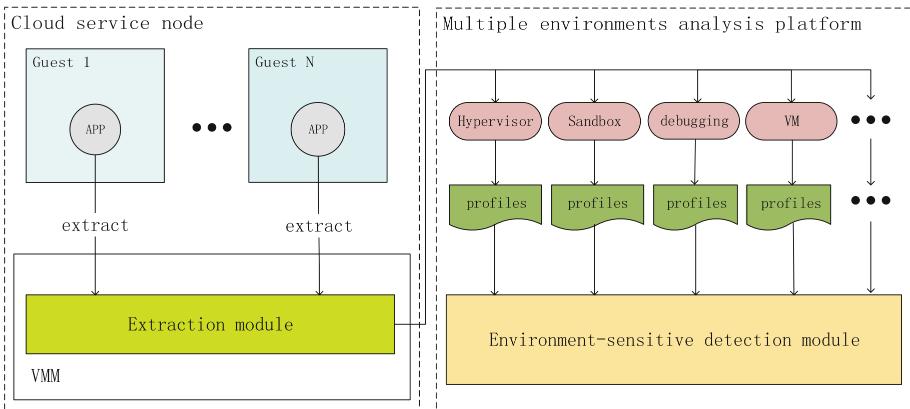

Fig. 1. FindEvasion system architecture.

3 Implementation

3.1 Transparent Extraction

In order to analyze a suspicious program which is in the guest Operating System (OS), we should extract it to the multiple environments analysis platform transparently. Note that the suspicious program is running in the guest VM. So the simple socket operation, like FTP, is easy to be awared by environment-sensitive malware because of the abnormal network behaviors. For this reason, we need to make use of the virtualization technology to extract the suspicious program and the whole process will not be awared by the malwares in the guest VM.

The detail is illustrated in Fig. 2. It is necessary to point out that the kernel module in the Guest OS has no HOOK operations and it can be completely hidden and protected by the VMM. Hence, the suspicious program is hard to detect inside the VM. For instance, if the Guest OS is win7, we can hide the module through monitoring the *NtQuerySystemInformation* function in the VMM. While a malware calls the function to query the modules in system, the VMM will intercept it and return the fake information to the malware. In this way, the kernel module can be hidden.

To better understand the procedure, we introduce the step details in Fig. 2. (1) While a suspicious program is going to run in the Guest OS, the Extraction module can capture this behavior. Then the Extraction module injects an event to notify the kernel module in the Guest OS. (2) The kernel module in the Guest OS receives notification from the Extraction module, then locates the suspicious program's executable file and copies it to a buffer. (3) The kernel module in the Guest OS calls instruction *VMCALL* to cause a VM-Exit. Now,

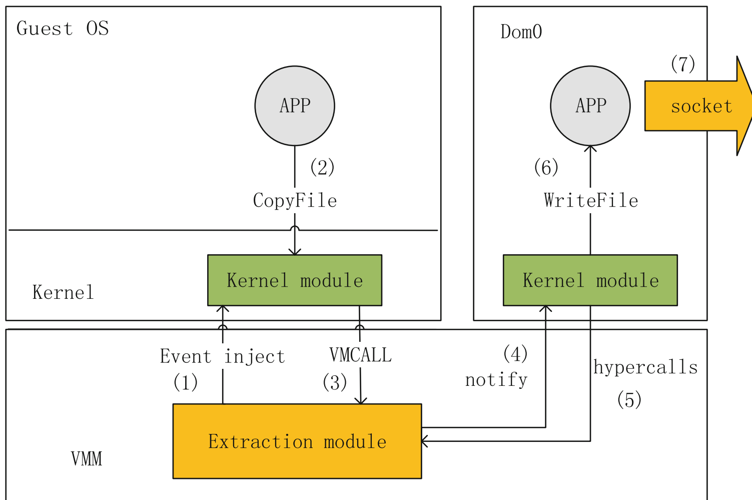

Fig. 2. Extract suspicious programs from Guest OS using virtualization technology.

the Extraction module obtains the binary executable file. (4) The Extraction module notifies the kernel module in Dom0. (5) The kernel module in dom0 reads the Extraction module through hypercalls. (6) The executable file is saved in Dom0. (7) We use socket operation to send the file from Dom0 to the multiple environments analysis platform. Here, we can use the socket operation, because the extracted suspicious program in Dom0 is only a static executable file and it can not be aware of the network behaviors. By this way, we can extract the suspicious program from Guest OS transparently.

3.2 Behavioral Profile

While the analysis of a suspicious program finishes in multiple operating environments, we need to extract it's behavioral profiles. Bayer et al. [6] have proposed an approach about how to extract behavioral profile from system-call trace. We will use a similar method in our system.

Similar to the model proposed by Bayer et al., we define our behavioral profile BP as a 4-tuple.

BP := (obj_type, obj_name, op_name, op_attr)

Where, *obj_type* is the type of objects, *obj_name* is the name of objects, *op_name* is the name of operation and *op_attr* is a corresponding attribute to provide additional information of a specific operation.

The *obj_type* is formally defined as follows.

obj_type := File(0) | Registry(1) | Syspath(2) | Process/Thread(3) | Network(4)

The *File* type represents this Behavioral Profile (BP) is a file operation, such as creating a file. The *Registry* type represents this BP is a registry key/value operation. The *Syspath* type represents this BP is a system key path operation, for example the *%systemroot%*. The *Process/Thread* type represents this BP is an operation about a process or a thread, such as terminating a process. And the *Network* type represents network behaviors, which include the remote IP and port. Each type is represented by integers 0, 1, 2, 3, 4 to reduce the complexity of behavior comparison later.

An operation must have a name, which is the API in reality. Besides, a corresponding attribute is needed to provide additional information about the operation. For example, the kernel function *NtDeviceIoControlFile* is used uniformly to represent all the socket functions related. Hence, we need additional information to tell us what exactly it is. That is, if we set the *op_attr* to the string "send", then we can clearly know this operation is the *send* function.

3.3 Behavior Normalization

In order to eliminate the influence of irrelevant factor and get a more reliable result, it is necessary for us to perform a series of normalization steps. As we all know, the same object may be represented differently in different systems, however, this will bring great differences in the behavioral profiles and then lead to a wrong judgment. Hence, we perform the following actions:

(1) We transformed uniformly all of the behavioral profiles into lowercase. The same behavioral profiles in different environment usually have different format. Some use uppercase and some use lowercase. In order to eliminate the differences, we use lowercase uniformly.
(2) We set a fix value to the *SID*. The registry key $HKEY_USERS\backslash <SID>$ is a secure identifier and the value is generally different in each system.
(3) We performed repetition detection. Some malwares perform many times with the same behaviors, which will cover up the real malicious acts. Therefore, if the number of repetitions is more than five times, the processing of duplicate removal is executed.

3.4 Behavior Comparison

The environment-sensitive malware often performs a lot of independent interference operations for anti-detection. The interference behaviors will appear in each environment, and if we do not deal with them, they will make up a large proportion of the behaviors and impact on the calculation of similarity. The previous methods, such as Hierarchy similarity [2], did not consider this issue, and it would lead to an absolutely opposite analysis result. Therefore, we propose a novel algorithm, named MBSS, which can eliminate interference behaviors and make the comparison more robust.

The algorithm model. Let $\mathrm{X} = \{\mathrm{x}_1, \mathrm{x}_2, \mathrm{x}_3, \ldots \mathrm{x}_n\}$, $\mathrm{Y} = \{\mathrm{y}_1, \mathrm{y}_2, \mathrm{y}_3, \ldots \mathrm{y}_m\}$, where x_1–x_n, y_1–y_m, each element represents a BP defined as Sect. 3.2, such that the set X represent all the Behavioral Profiles captured from a specific environment. Let L(X) be the number of elements of the set X and L(Y) be the number of elements of the set Y. Let set S be the intersection of set X and set Y, that is $\mathrm{S} = \mathrm{X} \cap \mathrm{Y}$. We recursively define Sim as:

$$Sim(X, Y) = \begin{cases} 1 & \text{if } 0 < \mathrm{L(X)} \leq \beta \text{ and } 0 < \mathrm{L(Y)} \leq \beta \\ 0 & \text{if } \mathrm{L(x)} == 0 \text{ and } \mathrm{L(Y)} == 0 \\ cpt(X, Y) & \text{if } \mathrm{S} == \varnothing \text{ and } \mathrm{L(X)} > \beta \text{ and } \mathrm{L(Y)} > \beta \\ Sim(X - x_i, Y - y_j) & \text{if } \mathrm{S} \neq \varnothing \text{ and } \mathrm{x}_i == \mathrm{y}_j \end{cases} \tag{1}$$

where,

$$cpt(X, Y) = \frac{AB}{|A||B|} = \frac{\sum_{i=1}^{n} A_i B_i}{\sqrt{\sum_{i=1}^{n} A_i^2}\sqrt{\sum_{i=1}^{n} B_i^2}} \tag{2}$$

Here, β is a configurable parameter and we designed an experiment in the Sect. 4.1 to try to search an optimal value for it. x_i is an element in set X and y_j is an element in set Y. A is a vector transformed from set X and $A_i \in A$. Also, B is a vector transformed from set Y and $B_i \in B$. We realized a method to transform the set into vector in Algorithm 2. The expression (2) is derived from the cosine similarity algorithm and it represents the similarity between set X and set Y after the interference operators are eliminated from set X and set Y. Therefore, *Sim(X, Y)* represents the similarity score. More details about how to eliminate interference behaviors are provided hereinafter.

We can clearly see that *Sim(X, Y)* always lies between 0 and 1. Hence, the deviation score between set X and set Y can simply be defined as:

$$Dis(X, Y) = 1 - Sim(X, Y) \tag{3}$$

Also, *Dis(X, Y)* is in interval [0,1], that is if the value tends to 0, the deviation between set X and set Y is small. On the other hand, if the value tends to 1, the deviation is large. We define a deviation threshold t. If the *Dis(X, Y)* is greater than t, we consider the suspicious program as an environment-sensitive malware.

Eliminate interference behaviors. Here, we use a simple but effective method to eliminate interference behaviors. First we scan the behavioral profiles captured from different environments, if there is a common behavioral profile, that is all the elements in the 4-tuple defined as Sect. 3.2 are the same, we record the position until all the common behavioral profiles are found. Then we remove common behavioral profiles according to the positions we record. In this way, we can eliminate most of the interference behaviors and leave the real malicious behaviors behind. This simple method works well in our experiment.

We implement the above algorithm with pseudo code.

Algorithm 1. MBSS algorithm

Input: a suspicious samples behavioral profiles extracted in different environments

Output: the sample is environment-sensitive or not

```
1  def Judge(bp1,bp2):
2          Dis = 1 - Sim(bp1,bp2)
3          if Dis > t:
4                  return TRUE
5          else:
6                  return FALSE
7  def Sim(bp1,bp2):
8          if 0 < len(bp1) ≤ β and 0 < len(bp2) ≤ β:
9                  return 1
10         elif len(bp1) == 0 and len(bp2) == 0:
11                 return 0
12         lines=[line for line in bp1 if line in bp2]
13         if len(lines) == 0:
14                 return cpt(bp1,bp2)
15         for line in lines:
16                 bp1.remove(line)
17                 bp2.remove(line)
18         return Sim(bp1,bp2)
```

In Algorithm 1, the parameter t in the line 3 is a threshold. Lines 3–6 give the result that the sample is environment-sensitive or not. Lines 7–18 is the mainly part of our algorithm to compute the similarity score. Line 12 is to get

the common behavioral profiles between *bp1* and *bp2*. Lines 13–14 represent that if there is no common behavioral profile, then we compute the similarity score. More details are going to be described in Algorithm 2. Lines 15–17 represent that if there are a few of common behavioral profiles, then we do the processing of eliminating interference, which just removing the common behavior profiles from the set.

We implement the Algorithm 2 with pseudo code. Lines 2–3 is to split all the 4-tuple behavioral profiles into words. Line 4 is to union all the words into a set. Lines 6–14 transform the set into vector, that is if an element not only in the set *allwords* but also in the set *word1*, then the *vector1* appends a value 1, otherwise, appends a value 0. Line 15 makes use of the cosine similarity algorithm to compute the similarity score.

Algorithm 2. Function cpt()

Input: a suspicious samples behavioral profiles after the interference behaviors are eliminated

Output: the similarity score

```
1  def cpt(bp1,bp2):
2          word1 <- split the bp1 into words
3          word2 <- split the bp2 into words
4          allwords <- union all the words in word1 and word2
5          vector1 = [], vector2 = []
6          for w in allwords:
7                  if w in word1:
8                          vector1.append(1)
9                  else:
10                         vector1.append(0)
11                 if w in word2:
12                         vector2.append(1)
13                 else:
14                         vector2.append(0)
15         return cosine(vector1,vector2)
```

4 Evaluation

We use Xen-4.4.0 [4] to build the Cloud service node. The Hypervisor environment used in multiple environments analysis platform is also based on Xen-4.4.0. We use cuckoo [5] to build Sandbox environment. Moreover, we deploy debugging environment with windbg and Ollydbg, and deploy VM environment using VMware workstation 12. And we choose Windows 7 SP1 (32bit) as the operating system for all analysis environments in the experiment.

We use the precision and recall [7] to measure the detection effectiveness.

$$Precision = \frac{TP}{TP + FP}, \ Recall = \frac{TP}{TP + FN} \tag{4}$$

where, TP represents true positive, FP represents false positive and FN represents false negative.

We designed four experiments for the following purposes. The first experiment was to look for the optimal parameter β used in MBSS algorithm. The second was to evaluate MBSS algorithm by performing the precision-recall analysis. The third was to demonstrate the effectiveness of eliminating the interference behaviors on detecting the environment-sensitive malwares. The last experiment was a large scale test for evaluating the feasibility and usability of FindEvasion.

In order to evaluate our approach, we selected the BareCloud [2] as a comparison in the following experiments. The BareCloud was developed to detect environment-sensitive malware in 2015, and used the Hierarchy similarity algorithm to compare the behavioral profiles. It has the 40.20% recall rate with 100% precision.

4.1 Optimal Parameter β Selection

In this experiment, we try to look for the optimal parameter β used in our algorithm.

Dataset. We randomly selected 140 environment-sensitive malwares and 140 common malwares as the dataset of this experiment. For simplicity, we just considered Win32 based malware in PE file format.

We extracted the behavioral profiles of these samples from all the analysis environments and computed the deviation score by varying the parameter β between 2 and 20. The result is illustrated in Fig. 3. We can clearly see that when the parameter β exceeds 8, the precision keeps on 100%. According to our algorithm defined in Sect. 3.4, when we choose a higher value for the parameter β, the similarity score will get higher so that the deviation score will become lower. That is, if a malware is judged as environment-sensitive, it will always be true with the 100% precision. However, from the Sect. 3.4, the expression (1) tells us that if we select the β too high, the similarity score will have great chance to be 1. This will cause the deviation score to be 0 and the recall rate will be lower relatively. Therefore, we can choose β between 9 and 12. Here, we selected $\beta = 10$.

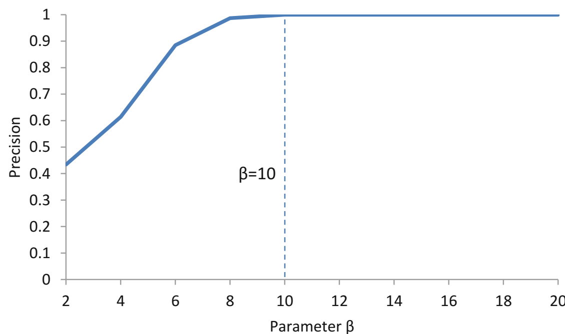

Fig. 3. The selection of parameter β

4.2 Algorithm Evaluation

In this experiment, we evaluated our MBSS algorithm by comparing with the Hierarchy similarity algorithm.

Dataset. We selected 542 environment-sensitive malwares and 319 common malwares. Also, we just considered Win32 based malware in PE file format for simplicity.

We extracted the behavioral profiles of above malwares from all the analysis environments and computed the deviation score using MBSS algorithm and Hierarchy similarity algorithm.

We performed a precision-recall analysis by varying the threshold t for these deviation score. If the deviation score exceeds the threshold t, the sample is considered as environment-sensitive. The result is presented in Fig. 4. We can clearly see that the MBSS algorithm gives better results. The reason is that the interference behaviors can impact on the detection of environment-sensitive malwares and our algorithm is able to cope with this issue. In the Sect. 4.3, we demonstrated the effectiveness of eliminating the interference behaviors.

Figure 5 illustrates the precision-recall characteristics of the MBSS algorithm by varying the threshold t between 0 and 1. We can clearly see that when the threshold $t = 0.75$, we get 100% precision with the recall rate of 60%. Compared to the recall rate of Hierarchy similarity algorithm, our algorithm's recall rate increases by 20% approximately.

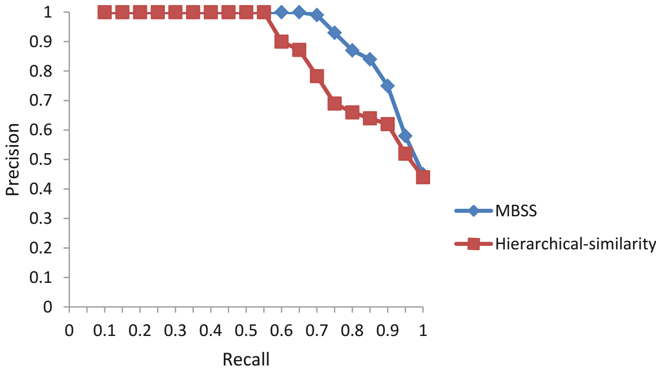

Fig. 4. Precision-Recall analysis of the MBSS and Hierarchy similarity behavior comparison

4.3 The Effectiveness of Eliminating Interference Behaviors

Since the Hierarchy similarity does not consider the influence of interference behaviors, we can therefore demonstrate the effectiveness by comparing the detection number of environment-sensitive malwares.

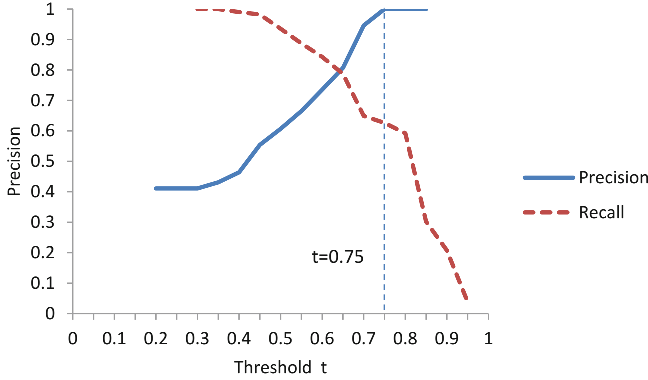

Fig. 5. Precision-Recall analysis of the behavior devision threshold value t

Dataset. We selected 380 environment-sensitive malwares as the dataset of this experiment. Each of the above malwares can perform a lot of interference operations. We only considered Win32 based malware in PE file format.

We extracted the behavioral profiles of these samples from all the analysis environments and computed the MBSS-based deviation score. We used the threshold $t = 0.75$ and parameter $\beta = 10$ that were selected in the previous experiments. We also used the Hierarchy similarity to calculate deviation score. The comparion result is shown in Fig. 6. We can clearly see that the MBSS algorithm gives better results. The MBSS algorithm was able to detect a total of 351 environment-sensitive malwares, which accounted for 92.4%. By contrast, the Hierarchy similarity only detected a total of 93 environment-sensitive malwares, which accounted for 24.5%. In other words, if an environment-sensitive malware performs a lot of interference operations, our MBSS algorithm works better than Hierarchy similarity algorithm. It also proves that the operation of eliminating interference behaviors is useful to detect the environment-sensitive malware.

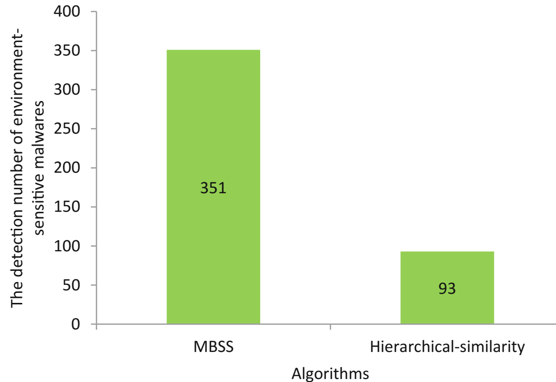

Fig. 6. The detection effect of MBSS algorithm compared to Hierarchy similarity algorithm

4.4 Large Scale Test

In this experiment, we evaluated the feasibility and usability of our FindEvasion system on a larger dataset, using BareCloud [2] system as a comparison.

Dataset. We have used VXHeaven Virus Collection [8] database which is available for free download in the public domain. We selected a total of 7257 malware samples and only considered Win32 based malware in PE file format. Note that, since we do not have a ground truth for this dataset, we cannot provide the precision rate and recall rate.

We ran FindEvasion and BareCloud using the same dataset, and made a judgment. The result is presented in Fig. 7. We can clearly see that our FindEvasion system detected 176 more samples than BareCloud did. Through manual reverse analysis, we confirmed that these samples are environment-sensitive malwares.

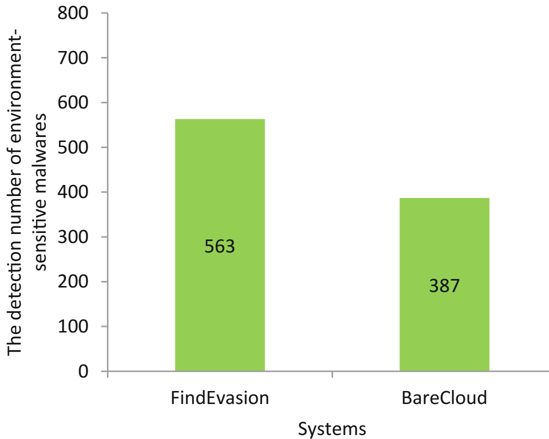

Fig. 7. The detection effect of FindEvasion and BareCloud

5 Limitations

Through the experiments result, we can clearly see that FindEvasion is able to detect environment-sensitive malwares. However, some samples using specific technologies can escape the detection. In this section, we describe the limitations of our system.

Firstly, if a sample uses stalling code to wait for some times before performing malicious behaviors, our system will lead to a wrong analysis result. The reason is that, our system's analysis time is limited. Within the limited time, the malware sample may be sleeping and escape the detection.

Secondly, our system can only identify the environment-sensitive malwares and it can not find out the provenance of the infection which may lead back to the offender. Our log files can only record the behaviors of malwares which do not include the attack's information.

6 Related Work

6.1 Dynamic Analysis

Dynamic analysis is the testing and evaluation of an application during runtime. Recently, many dynamic analysis tools have been developed for automatically analyzing malware. Most of them make use of the sandbox techniques. A sandbox is implemented by executing the software in a restricted operating system environment. Some tools like CWSandbox [9] and Norman Sandbox [10], making use of in-guest techniques for intercepting Windows API calls. This method is easy to be awared by environment-sensitive malware and be bypassed. The emulation or virtualization technologies are also universally used, for example VMScope [11], TTAnalyze [12], and Panorama [13], which are based on the Qemu [14] to record the API. Besides, Ether [15], VMwatcher [16] and HyperDBG [17] are the representative of hardware-supported virtualization technology.

6.2 Transparent Monitoring

In order to prevent the environment-sensitive malware from escaping the detection, it is necessary to develop transparent analysis platforms. Cobra [18] uses dynamic code translation, fighting with the environment-sensitive malware with anti-debugging techniques. It performs the behavioral analysis by modifying the memory properties. There are also a number of tools based on the out-of-VM monitoring which can provide transparent monitoring. Examples include Ether [15] which makes use of the hardware-supported virtualization. However, the tools above only provide very few kinds of environments which is not conducive to identify the environment-sensitive malware.

6.3 Evasion Detection

Chen et al. [19] proposed a detailed classification of anti-virtualization and anti-debugging techniques used by environment-sensitive malwares. According to their experiments, if an environment-sensitive malware is under a debugger or virtual machine environment, it showed less malicious behaviors. Lau and Svajcer [20] have proposed a method to detect VM detection by dynamic-static tracing technique. Disarm [3] deployed two kinds of analysis environments to compare the behavioral profiles. It requires each sample to be analyzed multiple times in each analysis environment. This procedure would reduce the influence of random files name. After that, it computes the deviation score through the inter-sanbox distance and intra-sanbox distance based on the Jaccard similarity. BareCloud [2] use the bare-metal environment, which has no monitoring component in the Guest OS. They only consider the persistent change to the system and they proposed a hierarchical similarity algorithm based on the Jaccard similarity to compute the deviation score. The major difference between BareCloud and our work is that we deployed multiple analysis environments and we proposed a novel algorithm, which can deal with the interference behaviors.

7 Conclusions and Future Work

In this paper, we present FindEvasion, a malware detection system for the Cloud. Different from traditional system, our system introduces a novel evasion detection algorithm that can effectively detect environment-sensitive malwares. As mentioned above, the environment-sensitive malwares can identify the operating environment and perform corresponding malicious behaviors in different environment. With the development of cloud computing, they have gradually become an important threat to cloud platforms. In order to make the environment-sensitive malware exhibit the evasive behavior and cope with the interference behaviors, we perform malware analysis on multiple operating environments and propose an algorithm to compare the suspicious programs behavioral profiles. Our approach can tranparently extract the suspicious programs from the guest VM and eliminate the influence of the interference behaviors. We have empirically demonstrated that this approach works well in practice and that is efficient.

In future, we would like to focus on adding the capability of human-computer interaction and handling stalling code. A malware can sleep for a long time to escape the analysis or the malicious behaviors need human to interact. Within a limited analysing time(e.g., five minutes), our system can not observe the malicious behaviors and this will lead to a wrong analysis result. Besides, our log files should record the provenance of the infection for leading back to the offender. We will deal with these issues in the future. Moreover, we plan to evaluate the robustness of our proposed technique on a customized dataset.

Acknowledgments. This paper is supported by National Natural Science Foundation of China (NSFC) under Grant No. 61572481, National key research and development program of China under Grant No. 2016YFB0801600 and Nation key research and development program of China under Grant No. 2016QY04W0900.

References

1. Symantec. https://www.symantec.com/security-center/threat-report
2. Kirat, D., Vigna, G., Kruegel, C.: Barecloud: bare-metal analysis-based evasive malware detection. In: Malware Detection (2014)
3. Lindorfer, M., Kolbitsch, C., Milani Comparetti, P.: Detecting environment-sensitive malware. In: Sommer, R., Balzarotti, D., Maier, G. (eds.) RAID 2011. LNCS, vol. 6961, pp. 338–357. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-23644-0_18
4. Linux Foundation: The Xen project. http://www.xenproject.org/. Accessed 4 Mar 2017
5. Cuckoo Sandbox. http://www.cuckoosandbox.org
6. Bayer, U., Comparetti, P.M., Hlauschek, C., Krgel, C., Kirda, E.: Scalable, behavior-based malware clustering. In: Network and Distributed System Security Symposium, NDSS 2009, San Diego, California, USA, February 2009
7. Powers, D.M.W.: Evaluation: from precision, recall and f-factor to ROC, informedness, markedness and correlation. J. Mach. Learn. Technol. **2**, 2229–3981 (2011)

8. VX Heaven Virus Collection: VX Heaven. http://vx.nextlux.org. Accessed 4 Mar 2017
9. Willems, C., Holz, T., Freiling, F.: Toward automated dynamic malware analysis using CWSandbox. IEEE Secur. Priv. **5**(2), 32–39 (2007)
10. Norman Sandbox. http://www.norman.com/
11. Jiang, X., Wang, X.: "Out-of-the-Box" monitoring of VM-based high-interaction honeypots. In: Kruegel, C., Lippmann, R., Clark, A. (eds.) RAID 2007. LNCS, vol. 4637, pp. 198–218. Springer, Heidelberg (2007). https://doi.org/10.1007/978-3-540-74320-0_11
12. Bayer, U., Kruegel, C., Kirda, E.: TTAnalyze: A Tool for Analyzing Malware (2006)
13. Yin, H., Song, D., Egele, M., Kruegel, C., Kirda, E.: Panorama: capturing system-wide information flow for malware detection and analysis. In: ACM Conference on Computer and Communications Security, CCS 2007, Alexandria, Virginia, USA, pp. 116–127, October 2007
14. Bellard, F.: QEMU, a fast and portable dynamic translator. In: Conference on USENIX Technical Conference, p. 41 (2005)
15. Dinaburg, A., Royal, P., Sharif, M., Lee, W.: Ether: malware analysis via hardware virtualization extensions. In: ACM Conference on Computer and Communications Security, CCS 2008, Alexandria, Virginia, USA, pp. 51–62, October 2008
16. Jiang, X., Wang, X., Xu, D.: Stealthy malware detection through VMM-based "Out-of-the-Box" semantic view reconstruction. In: ACM Conference on Computer and Communications Security, CCS 2007, Alexandria, Virginia, USA, pp. 128–138, October 2007
17. Fattori, A., Paleari, R., Martignoni, L., Monga, M.: Dynamic and transparent analysis of commodity production systems. In: IEEE/ACM International Conference on Automated Software Engineering, pp. 417–426 (2010)
18. Vasudevan, A., Yerraballi, R.: Cobra: fine-grained malware analysis using stealth localized-executions. In: IEEE Symposium on Security & Privacy, p. 15 pp. -279 (2006)
19. Chen, X., Andersen, J., Mao, Z.M., Bailey, M.: Towards an understanding of anti-virtualization and anti-debugging behavior in modern malware. In: IEEE International Conference on Dependable Systems and Networks with FTCS and DCC, pp. 177–186 (2008)
20. Lau, B., Svajcer, V.: Measuring virtual machine detection in malware using DSD tracer. J. Comput. Virol. Hacking Tech. **6**(3), 181–195 (2010)