EnvFaker: A Method to Reinforce Linux Sandbox Based on Tracer, Filter and Emulator against Environmental-Sensitive Malware

1st Chenglin Xie
Peking University
Beijing, China
cony1996@pku.edu.cn

2nd Yujie Guo
Peking University
Beijing, China
justaname@pku.edu.cn

3rd Shaosen Shi
Peking University
Beijing, China
deadpoo3@pku.edu.cn

4th Yu Sheng
Peking University
Beijing, China
syu@pku.edu.cn

5th Xiarun Chen
Peking University
Beijing, China
xiar_c@pku.edu.cn

6th Chengyang Li
Peking University
Beijing, China
lcymoon@pku.edu.cn

7th Weiping Wen
Peking University
Beijing, China
weipingwen@pku.edu.cn

***Abstract*—Sandbox is an excellent tool for dynamic malware analysis. However, the sandbox detection techniques are increasingly adopted to develop malwares, which has been a significant threat to sandbox analysis. These malwares can detect the running environment and show different behaviors in corresponding environments. So far, there have been several studies about countermeasures, but most of them concentrate on Windows OS. Environmental features in Linux sandbox have not been summarized yet. Besides, existing popular sandboxes can hardly combat against sandbox detecting techniques.In this paper, we focus on Linux sandbox. We firstly propose Linux environmental features from six aspects and implement an effective tool to collect features from running environment to tell the discrepancy among physical machine, virtual machine and sandbox. More importantly, we present EnvFaker, an effective method to reinforce Linux sandbox against environmental-sensitive malware. This method uses tracer to track child process and injected process, filters to intercept sandbox detecting behaviors, and emulator to disguise wear-and-tear and network environment. The experimental results further demonstrate that our method is effective against detecting techniques for Linux sandbox.**

***Index Terms*—Sandbox reinforcement, Environment-sensitive malware, Environmental features**

I. Introduction

In recent years, malwares targeting Linux OS are on the rise. To detect malwares, sandbox is widely used in dynamic analysis to facilitate malware analysts to quickly and intuitively gain the runtime behavior of malware. With the increasing use of sandboxes, various environment-sensitive malwares have grown rapidly. These malwares can detect current running environment and show different behaviors in corresponding environments. As shown in Fig. 1, if malwares try to detect current environmental features and find that they are in a sandbox, they will stop executing immediately to avoid exposing too many malicious behavoirs to foil dynamic analysis.

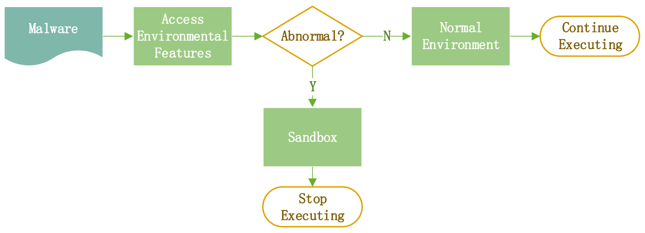

Fig. 1. How to identify a sandbox.

Traditional environmental-sensitive malware studies usually focus on Windows OS. However, the features in Linux sandbox are quite different. There are two main differences. One is that Windows OS has more user interaction than Linux OS because Linux OS is often used for server or IoT devices. So, there is no need to hide user interaction features [1]. Another is that some special features in Windows OS are not suitable for Linux environmental detection. These features will not be taken into consideration in our method, such as registry, browser history and user interaction (Typing, clipboard, mouse movement and display resolution) [2]. By contrast, process features, wear-and-tear artifacts [3], hardware features [4], particular patterns [5], and tracing features can be more effectively applied in Linux OS.

So far, some scholars have proposed a variety of countermeasures against environmental-sensitive malware, such as symbolic execution [6] and multi-environmental analysis [7]–[9]. However, symbolic execution and multi-environment analysis can bring great overhead, difficult to directly deployed in sandbox. Besides, user behavior emulator [10] and Windows API hooking technique [11] have been introduced to interfere with environment detecting behaviors. Even so, these methods are proposed for Windows sandbox and simple user behavior emulator or API hooking technique is not enough for sandbox reinforcement.

In this paper, we concentrate on Linux sandbox and seek to tackle these problems. We propose Linux environmental features from six aspects to identify sandbox and implement an effective tool to collect features from running environment to tell the discrepancy among physical machine, virtual machine and sandbox. By applying this tool, we design experiments to evaluate the effectiveness of our features. In addition, we propose EnvFaker, an effective method to reinforce Linux sandbox against environmental-sensitive malware. This method takes tracer to track child process or injected process of malware, filter to intercept actions of environment detecting and emulator to disguise wear-and-tear and network environment. By these ways, we effectively reinforce Linux sandbox from 6 aspects: process features, wear-and-tear features, hardware features, particular patterns, network emulation and tracing features.

In conclusion, this paper makes the following contributions:

1) We analyze thousands of Linux ELF malware samples and make statistics for common malware behaviors. We filter out improper features from Windows OS and find new features of Linux sandbox, firstly concluding a list of features from 6 aspects to detect Linux sandbox, which can be of great help for future research.
2) We implement an effective tool to gather features from running environment to show the discrepancy among physical machine, virtual machine and sandbox and evaluate the effectiveness of the features mentioned above.
3) We present a comprehensive method called EnvFaker to reinforce Linux sandbox from process features, wear-and-tear features, hardware features, particular patterns and tracing features to induces environmental-sensitive malware to show its malicious behavior.

The rest of the paper is organized as follows. Section II introduces environmental features in Linux sandbox, and Section III is for architecture and implementation. The evaluation will be elaborated in Section IV, and the limitation and discussion will be given in Section V. Section VI is related work. Section VII will conclude the paper.

II. Environmental Features in Linux Sandbox

To determine the plausibility of an environment being a real Linux machine or a sandbox, malwares depend on many details that are indicative of environmental features. In this section, based on previous work, we seek to find corresponding instance of well-known features in Linux sandbox and filter out improper features in Windows OS. Combined with these features, we add some particular features in Linux sandbox and classify them into 6 aspects. Most of the features are reliable and deterministic and will be evaluated in Section IV. To make our features convictive, we analyze thousands of Linux malwares for our study. We download ELF malware samples from VirusTotal [12] and VirusShare [13] platform between 2019 and 2020 seperately to make data more comprehensive. We download 578 samples from VirusTotal and 631 samples from VirusShare. Based on these samples, we make statistics about environmental behaviors, as shown in Table. I.

TABLE I
Top 10 possible environmental detecting behaviors

Behaviors	Counts	Percentage
Initial process getting parent pid	158	13.07%
Getting MAC address information	87	7.20%
Getting CPU information	23	1.90%
Accessing memory file	22	1.82%
Executing ps program	13	1.08%
Executing netstat program	13	1.08%
Getting module information	5	0.41%
Accessing log files	3	0.25%
Getting DMI information	3	0.25%

Process Features. To be a sandbox, firstly, it is necessary for the OS to pre-deploy some monitoring process. Secondly, virtual machine is often adopted to facilitate recovering environment. Furthermore, it is necessary for sandbox to have a tracer to track target process. So, process features in Linux sandbox are fingerprints related to sensitive process in process list and artifacts in process environment. We classify sensitive process into 3 categories: monitoring process, virtual machine daemon process and tracing process. Examples of sensitive monitoring processes include Tshark [14], TCPDUMP [15], etc. Examples of sensitive virtual machine daemon processes include vbox, vmtools, etc. Examples of sensitive tracing processes include strace, systemtap [16], etc. These running processes indicate that the current running environment is abnormal. Malwares use ps program to gather process information. Besides, ld_preload variable in process environment stands for tracing features because some sandboxes may inject codes into malwares to trace their behavior by setting this variable.

Wear-and-Tear Features. Wear-and-Tear features of sandbox were firstly proposed by N. Miramirkhani et al. [3] , which indicate the history of use of an OS. However, they focus on Windows OS. Most of the features they have proposed are not suitable for Linux. We filter out specific features of Windows sandbox such as interaction features and keep common and representative trails such as log count, network connection count, etc. Log files record past activity in OS. To some extent, active network connection count and running process count can represent how long this system has worked. Besides, installation date, update date and last login date are also exploitable sources to check how old a system is because these values will be reset after sandbox recovers its system.

Hardware Features. Hardware information shows the artifacts of running environment. Due to the widespread use of virtual machines, hardware features of virtual machine reveal

TABLE II
Top 9 common used ports of samples

Ports	Protocols	Counts	Percentage
23	TELNET	176	14.56%
53	DNS	102	8.44%
80	HTTP	71	5.87%
2323	/	61	5.05%
666	/	43	3.56%
37215	/	41	3.39%
8080	/	31	2.56%
443	HTTPS	20	1.65%
52869	/	14	1.16%

lots of environmental information in sandbox. A. Yokoyama et al. [1] have propose many hardware features of sandbox. Similarly, their work concentrates on Windows OS too. To find hardware features in Linux sandbox, besides concluding previous features and filter out improper features, we absorb hardware features from VM-detect tools such as check-VM module in Metasploit Framework [17]. In addition, we classify these features into 8 subcategories: DMI information, hardware configuration, CPU information, SCSI information, disk information, kernel ring buffer information, MAC address information and others. Details of files or binaries to get these features will be further demonstrated in Table. III.

Network Environment Features. Some sandboxes such as Lisa sandbox [18] cut off Internet connection simply for security consideration. However, some cunning malwares tend to connect to their server by sorts of protocols in reverse to inform the controller to download additional modules or send commands. It is of great significance to disguise network accessibility to induce malwares to show malicious behaviors. HTTP and FTP are two popular protocols to download files. DNS protocol is often used to resolve domain name of server from attacker. TELNET protocol is often adopted to executing commands in remote devices. As shown in Table. II, these protocols are commonly used by malware samples. The accessibility of these protocols is adopted to network environmental features.

Tracing Features. Tracing techniques is one of the most representative methods applied in sandbox such as ptrace and strace(a ptrace-based tool to track system call of process). By setting TRACE_ME parameter of ptrace function, malwares are able to check if they are being tracked due to the infeasibility of double-ptrace mechanism in Linux. As shown in Code. 1, By checking its parent process name, malware can discover tracer process. Kernel debugging file is also prepared for handling trace tasks.There are two switch files in */sys/kernel/debug/* folder representing the status of kernel debugging, providing an opportunity for malware to detect. Systemtap and kprobe are also widely used in Linux sandbox. By these ways, new kernel modules will be added to kernel module list to track behaviors of malwares. Hence, current kernel module list may be checked to make sure current environment is not under monitoring or tracing.

Particular Patterns. J. Blackthorne et al. [11] have pointed that hardcode sample name and computer name are widely used in Windows sandbox for the convenience of analysis. In Linux sandbox, this phenomenon may happen too. When systemtap is adopted in sandbox, kernel modules of tracing program are often compiled with hardcode sample name for efficiency. Examples of file or folder names include “sample”, “test” and “self”. The hash of the file is also a popular name during analyzing to specify unique names of samples. Examples of hostname include “sandbox”, “analyzer” and other meaningful phrases. These can be a potential threat to sandbox. Once malwares check these patterns before displaying malicious behaviors, they can find that they are running in a fake environment.

```
if ptrace(PTRACE_TRACEME,0,0,0)<0
    exit(-1); // stop running
else
    // expose malicious behaviors
    keep_running();
```

Code 1. Anti-debug method by ptrace

Environmental Detecting Tool For Linux sandbox. We implement an automation tool with Python on the foundation of an open-source tool named checkvm (a detecting module in Metasploit framework) to gather features mentioned above as rich as possible.

Considering that some platforms may rename malware to containing its hash string, our tool packs its Python code file into executable file with Pyinstaller [19] for convenience of hash calculation. While running, the tool calculates hash values of itself and check if its name matches particular patterns or hash strings by accessing three different process files:*/proc/self/comm*, */proc/self/cmdline* and */proc/self/status*. Besides, this tool gathers process features by executing ps program to find sensitive process and detecting ld_preload variable in process environment by read */proc/self/env* file. Hardware features are collected by accessing hardware related file and executing hardware related binaries. Wear and tear features are collected by calculating row counts of log files, counts of TCP connection, counts of running process, installation date, update date and last login date. Several thresholds were set to distinguish the new and the old machine according to the results of our survey of used machine. To test whether network can be reached, this tool tries to download files by HTTP and FTP protocol from a temporary server and resolve a specific domain name. Tracing features are detected by setting the parameter TRAME_ME of ptrace function and check its return value. To get parent process name, it calls getppid function and accesses */proc/self/status* file to transfer pid to name. This tool compares parent process name with sensitive list to check environment. The tool accesses kernel debugging files to check if kernel debugging is enabled and kernel module file to check abnormal modules like systemtap.

The effectiveness of this tool will be evaluated in Section IV-A.

III. Architecture and Implementation

The reason why we deploy EnvFaker into Linux sandbox is that this system, which identifies environmental detecting behaviors of suspicious malwares, provides pre-configured environment with tracer, emulator and filter, and redirects environmental query requests to misguide malware. Fig. 2 shows an overview of EnvFaker architecture, which consists of 3 parts. The first one is target tracer which tracks child process and injected process by ptrace. The second one is features emulator, which offers malwares an emulation environment after operating system booting. When malwares detect wear-and-tear features or network environment to check whether it runs in a sandbox or accessing well-known services, the emulator works to misguide environmental-sensitive malware. The last one is the features filter, which includes process features filter, hardware features filter, tracing features filter and particular pattern filter. The features filter redirects environmental detecting requests from environment-sensitive malwares to a crafted environment to mislead them. By the three modules mentioned above, malwares draw a fake conclusion about the running environment. Details of the system will be elaborated in the following section.

To expound on the implementation of EnvFaker, we demonstrate the principles of the main three modules. The relationship among the three modules are shown in Fig. 3.

A. Target Tracer

During running, malwares may create processes or inject other processes to take various tasks, both of which should be taken into consideration. As shown in Table. IV, about 45% of these samples create subprocesses or threads. Only about 22% of samples just handle their task in single process. As shown in Table. V , about 0.41% of these samples inject other processes by ptrace, 0.58% of which use TRACEME parameter of ptrace against dynamic analysis from ptrace-based tool. To attach to the targets and track the behaviors of target samples comprehensively, we ought to monitor process creation and injection. We adopt two different approaches to maintain a target process list with the help of systemtap.

Child Process Tracer. To track child process or thread of target malware, we monitor the function of process creation. Every time a process calls this function, its parent process id will be compared with that in target process list and recorded. If right, this process id will be appended to the target process list.

Injected Process Tracer. To track the injected process of target malware, we hook the system call named ptrace. Whenever target process calls this system call, its parameters will be recorded, some of which represent the target process id to be injected. This value will be added to the target process list too.

B. Features Filter

To filter out environmental detecting behaviors of malwares and redirect the requests from malwares to crafted environment, the first task we need to handle is to intercept detecting behaviors from malware. Then, our focus shift to how malwares detect environmental features. To this end, there are two feasible ways: executing specific commands like lshw or accessing specific file like */proc/cpuinfo*. In essence, these two methods are the same ways. We trace executing binaries and find that these programs try to get information by accessing specific file in the OS. So, by intercepting file accessing, we are able to intercept detecting behaviors from malwares.

As is known to us, the standard step to get the contents of a file is opening, reading and closing. Now, we focus on the first step and try to intercept file opening requests from sample target. In Linux OS, whenever a process attempt to open a file, open syscall is called and the system traps from user ring into kernel ring. Then, open syscall triggers kernel function sys_open in sequence and sysopen function calls kernel function do_sys_open subsequently. In this kernel function, getname function is called to get filepath of target file and do_filep_open funciton will handle the rest operations. To make this process clear, we introduce the simplified code of calling process, as is shown in Code. 2.

```
/* implementation of open syscall */
SYSCALL_DEFINE3(open, const char __user *,
    filename, int, flags, int, mode)
{
    ...
    // call do_sys_open
    long ret = do_sys_open(AT_FDCWD, filename,
     flags, mode);
    ...
}

/* implementation of do_sys_open function */
long do_sys_open(int dfd, const char__user *
    filename, int flags, int mode)
{
    ...
    // copy filename in user ring to kernel
    ring
    char *tmp = getname(filename);

    // call do_filep_open
    struct file *f = do_filp_open(AT_FDCWD,
    tmp, flags, mode, 0);
    ...
}
```

Code 2. Simplified Implement of open related function

To redirect the detecting requests from malwares, we ought to replace the filepath. However, filepath may be used in two different ways: relative path and absolute path. As shown in Table. VI, only 21% of samples use relative path. Luckily, most of them are used for configuration and environmental feature files are often place in fixed folders. Therefore, intercepting absolute path is enough for us. By hooking parameters

TABLE III
ENVIRONMENTAL FEATURES IN LINUX SANDBOX

Category	Features	Details
Process Features	Sensitive network monitor process	Tshark, Wireshark [14] and TCPDUMP process
	Sensitive tracer process	Sysdig, strace, ftrace, and Systemtap module process
	Sensitive VM daemon process	iprt-VBoxWQueue, iprt-VBoxTscThr, VboxClient, VBoxService, VirtualBoxVM, vm-toolsd, vmhgfs and irq/16-vmwgfx process
	Injected process	Ld_preload variable
Wear-and-Tear Features	Row counts of log files	/var/log/secure, /var/log/messages, /var/log/maillog, /var/log/cron, /var/log/httpd/ac-cess_log, /var/log/httpd/error_log, /var/log/mysqld.log files
	Running process counts	Results of executing ps program
	Active connection counts	Results of executing netstat program
	Particular dates	OS installation data, latest update date and last login date
Hardware Features	DMI information	Results of executing dmidecode program and accessing /var/log/dmesg, /sys/firmware/d-mi/tables/DMI, /sys/class/dmi/id/product_serial, /sys/class/dmi/id/sys_vendor, /sys/-class/dmi/id/modalias, /sys/class/dmi/id/product_name, /sys/class/dmi/id/product_uuid and /sys/class/dmi/id/uevent files
	Hardware configuration	Results of executing lshw and lspci program, accessing /usr/share/hwdata/pci.ids, /usr/share/hwdata/pnp.ids and /usr/share/hwdata/usb.ids files
	CPU information	Results of executing lscpu program and accessing /proc/cpuinfo file
	SCSI information	Results of accessing /proc/scsi/scsi file
	Disk information	Results of accessing files in /dev/disk/by-id/ folder, names of which containing VBOX, VMware, QEMU
	Kernel ring buffer information	Results of executing dmesg program and accessing /dev/kmsg file
	MAC address information	Results of executing ifconfig and ip program, and accessing /proc/net/arp, /sys/-class/net/<network adapter>/address, /etc/sysconfig/network-scripts/ifcfg-<network adapter>files
	Other information	Total memory size, block size of file system, cores of CPU
Network Environment	Network Accessibility	HTTP, FTP, TELNET and DNS accessibility
Tracing Features	Being tracing	Being tracked by ptrace or not, ppid
	Kernel debugging file	/sys/kernel/debug/kprobes/enabled and /sys/kernel/debug/tracing/tracing_on file
	Kernel module information	Results of executing lsmod program and accessing /proc/modules file
Particular Patterns	Particular strings in file or folder names	“sample”, “test”, “self”, “analyzed” strings
	Hash strings in file or folder names	MD5, SHA1, SHA256 strings
	Particular strings in hostname	“sandbox”, “analyze”, “guest”, “cuckoo” strings

TABLE IV
SAMPLES CREATING SUBPROCESSES OR THREADS

Process Type	Counts	Percentage
Single process	264	21.84%
Subprocess count in [1,5]	393	32.51%
Subprocess count in [6,9]	65	5.38%
Subprocess count in [10,19]	37	3.06%
Subprocess count in [20,49]	10	0.83%
Subprocess count in [50,99]	19	1.57%
Subprocess count in [100,319]	12	0.99%
Creating threads	4	0.33%
Creating subprocesses or threads	540	44.67%

TABLE V
SAMPLES USING PTRACE

Ptrace	Counts	Percentage
TRACEME	7	0.58%
Injection	5	0.41%

of getname function, we can intercept the requests from malwares and replace target file with crafted file or non-exist file so that malwares get fake results.We implement 4 features filters from different perspectives.

Process Features Filter. Based on the process features mentioned in Section II, the first thing we should do is to hook process list to hide sensitive processes. Usually, ps program is used to get process list. We try to trace systemcall of this program to know how it works and find that it iterates all legiti-

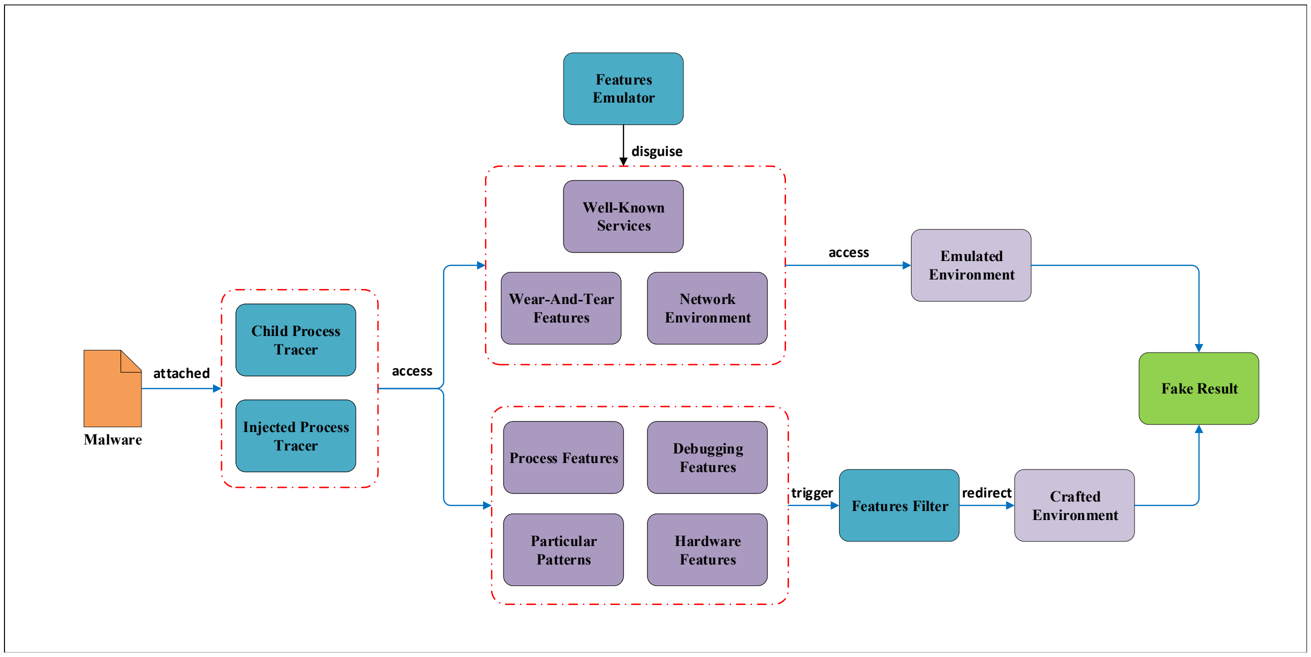

Fig. 2. Architecture of EnvFaker.

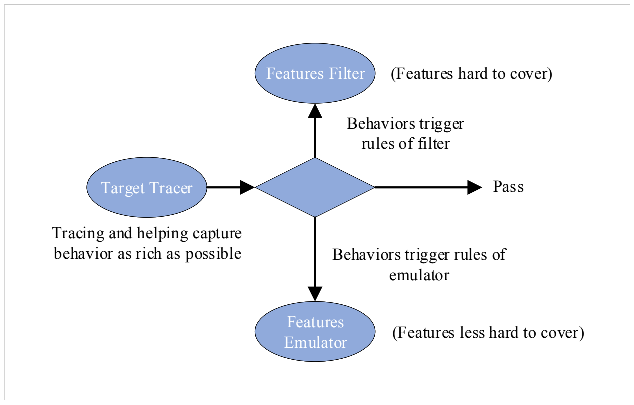
Features Filter
(Features hard to cover)
Behaviors trigger rules of filter
Target Tracer
Pass
Tracing and helping capture behavior as rich as possible
Behaviors trigger rules of emulator
Features Emulator
(Features less hard to cover)

Fig. 3. The relationship between tracer, fileter and emulator.

TABLE VI
WAYS TO USING PATH OF SAMPLES

Type	Counts	Percentage
Using relative path	258	21.34%
Only using absolute path	542	44.83%

mate pids and access */proc/<pid>/stat* and */proc/<pid>/status* file to form a running process list of system. So, if we hook these files, we can hide sensitive process even filter itself from ps program. Besides, once a process is hooked by ld_preload method, its process environment will be set with a special string. Similarly, process environment information stores in a file named */proc/<pid>/environ*. Thus, if we replace malware process the environment file by a normal non-injected process file, malware will gets normal result. These file accessings can be intercepted by hooking getname funciton in do_sys_open, as mentioned above.

Hardware Features Filter. Malwares may execute binaries in */usr/bin* folder like lshw or directly access hardware-related configuration file like */proc/cpuinfo* to get hardware features to detect environment. File details are listed in Table. III. By hooking file opening, we replace these files with our crafted file from physical devices to cheat on malwares.

Tracing Features Filter. Due to the infeasibility of double-ptrace mechanism in Linux, environmental-sensitive malwares can make use of the parameter TRACE_ME of ptrace function and check the return value to determine if their process are traced. Based on this principle, the solution is to modify the return value of ptrace function. Likely, we modify the return value of getppid function called by malwares to misguide them. Besides, we find that kernel debugging files such as */sys/kernel/debug/kprobes/enabled* and */sys/kernel/debug/tracing/tracing_on* will be set to specific values in virtual machine system. In addition, kernel modules list will change if systemtap or other kernel modules are deployed in the OS. The module list is stored in */proc/modules* file. We use kernel function hooking techniques to intercept this progress.

Particular Patterns Filter. Many sandboxes tend to rename samples to a common name or put samples into a particular folder for convenience of analysis, new names of which fit some particular patterns. To get their names, malware processes may access */proc/self/comm*, */proc/self/cmdline*, */proc/self/stat* and */proc/self/status* file. Changing the status file may cause instability of the malware because this file contains not only filename but also other important process information, which is changing during executing. So, we choose the other two files to filter. Hostname is another one that may contain particular patterns, stored in */etc/hostname* file. We prepare several crafted files names hidden name from target to avoid environmental-sensitive malwares from checking their names to find anomalies in OS.

C. Features Emulator

To make the environment real, sandbox environment need to be decorated to cheat on malwares. We implement three features emulators: services emulator, wear and tear features emulator and network emulator. The three emulators are pre-deployed in sandbox after OS booting.

Service Features Emulator. Sandbox is designed to analyze malware fast and accurately. So, a clean system is often adopted as default, which means some popular services such as Apache and MySQL may not be contained in sandbox. However, malwares may attack or collect information from these services such as injection or privilege promotion. To expose behaviors of malwares, well-known services need to be pre-installed in Linux OS.

Wear-and-Tear Features Emulator. Wear-and-tear represents how old a running system is. To disguise sandboxes as a kind of used system, we design an initializer to adjust log count, alive connection count and running process count by setting thresholds config file, according to current environment automatically.

Network Emulator. Internet connections are sometimes dangerous for sandbox in that part of hazardous malwares may attack other devices on the Internet. To handle this problem and provide malwares active feedbacks, INetSim [20] program is adopted in our system to respond network requests to malwares according to varieties of protocols.

IV. Evaluation

In this section, we first evaluate the effectiveness of the tool to distinguish physical machine, virtual machine and sandbox. We then use this tool to test our reinforcement method by comparing popular open-source Linux sandboxes before and after reinforcement, and to prove the effectiveness of our method.

A. Effectiveness of Environmental Detecting Tool

To evaluate the effectiveness of the detecting tool, we design two experiments separately to compare environmental features.

Firstly, we compare a new physical machine with a used physical machine to check the difference of wear and tear features in the two machines. It worth noting that Linux OS is often not installed on personal computer. Hence, we can only collect used Linux machine features from Linux server or IoT devices. Not like personal computer, Linux servers are usually used inside enterprise and IoT devices are often adopted in institute or family. For both of the two situations, features are hard to collect for privacy reason. To tackle this problem, we produce a used machine by ourselves. In the first experiment, we install Ubuntu 18.04 on both of the two physical machines. The new machine keeps its state until one month later. Another one is equipped with frequently-used services such as SSH, HTTP, FTP, DNS and MySQL, accessed by three humans randomly for a month. After one month, we run the detecting tool to check wear and tear features. If the statistic counts of logs, process and TCP connection trigger the threshold, the corresponding value of wear and tear features will be added by a value. The values represent the likeliness of being a machine, which recovers its OS frequently. Results are shown in Table. VII.

TABLE VII
WEAR AND TEAR DIFFERENCE BETWEEN THE TWO MACHINES

Category	Features	New Machine	Used Machine
Wear-and-Tear	Log File	7	3
	Process	1	0
	TCP Connection	1	0
	Date	3	1

As is shown in Table. VII, the feature data collected from the detecting tool is quite distinguishable. By randomly accessing for a month, we find that the secure log, message log, HTTP access log and MySQL log of used machine have grown fast, counts of which are much higher than the new one. Besides, process counts and TCP connections counts surpass the new one a bit. The last login date keeps in a week. The installation date and update date remain a month ago.

Then, we compare other features from the rest 5 aspects in multiple environments including physical machine, virtual machines and sandboxes. In this experiment, we install Ubuntu 18.04 on every machine. We choose 3 famous virtual machine platforms: VMware, VirtualBox and QEMU as representatives of virtual machines. Cuckoo [21], Limon [22] and Lisa [18] are three popular open-source sandboxes widely used to analyze Linux malwares, which can be of great representative for comparison. The Infrastructure in our experiment is featured with Intel(R) Core(TM) i5-8300H CPU @ 2.30GHz x 8 and 16GB memory. Based on the infrastructure, the virtual machines are configured with 4 vCPUs and 8GB memory. For sandboxes, the guest machines to analyze samples are configured with 1 vCPUs and 4GB memory. Specifically, we run this environmental detecting tool and observe the output collected from multiple environments. The detecting tool checks each item of features. If any item triggers the rule, a value will be added to the statistical data of this feature. These values represent the likeliness of being a sandbox. The results are summarized in Table. VIII.

By comparing physical machine with virtual machines, we find that features show great differences in sensitive processes because virtual machines tend to deploy daemon processes for the convenience of platform control. Due to some special configs in virtual machines, hardware strings

TABLE VIII
OTHER DIFFERENCES OF ENVIRONMENTAL FEATURES

Features	Physical Machine	Virtual Machines			Sandboxes		
		VMware	VirtualBox	QEMU	Cuckoo	Lisa	Limon
Process	0	4	6	1	8	3	7
Hardware	0	8	10	8	10	8	8
Network	0	0	0	0	0	2	0
Tracing	0	2	2	2	3	3	3
Pattern	0	0	0	0	1	1	0

and kernel debugging files vary a lot too. While we compare virtual machines with sandboxes, features also show great differences in sensitive processes in that sandboxes tend to deploy monitoring processes for tracing in addition to virtual machine daemon processes. Cuckoo sandbox adopts systemtap to track sample, TCPDUMP to get traffic data, INetSim to emulate network status and "cuckoo" as the default hostname. Lisa sandbox uses systemtap and TCPDUMP but uses "analyzed_bin" as sample name. Limon sandbox utilizes sysdig and strace to trace sample, tshark and TCPDUMP to get and analyze traffic data and INetSim to disguise network protocol. In conclusion, this tool is able to faithfully tell the discrepancy among physical machine, virtual machines and sandboxes.

B. Effectiveness of Reinforcing Method

To evaluate the effectiveness of the reinforcing method, we firstly focus on the components of the three open-source Linux sandbox. As shown in Tabel. IX, Cuckoo and Lisa use Systemtap as their tracer and Limon adopts Strace and Sysdig as its tracer. For Cuckoo, this sandbox maintains a white list to rule out irrelevant process. Except for white list, all behavoirs of other processes in OS will be recorded. This method monitors the changes in OS caused by samples to a large extent, but brings false positive due to the behaviors of irrelevant processes. For Lisa, this sandbox tracks sample processes by hooking fork and clone syscall but ignore the possibility of process injection. For Limon, this sandbox uses strace to track samples, which could be detected by TRACE_ME parameter of ptrace syscall. To mitigate these limitation, we adopt process creating and ptrace call as our target. This method brings no false positive and tracks the behavoirs of target as much as possible.

Then, we shift to focus on the effectiveness of filter and emulator. We compare three popular open-source Linux sandboxes before and after reinforcement from the 6 aspects features. In this experiment, physical device configuration, virtual machine platform configuration and OS configuration are same as those in Section IV-A. We divide machines to be tested into two groups: sandboxes before and after reinforcement. The machines in the first group are installed normally according to the official documentation. Besides normal installation, machines in another group are pre-deployed with target tracer, features filter and features emulator. By running the detecting

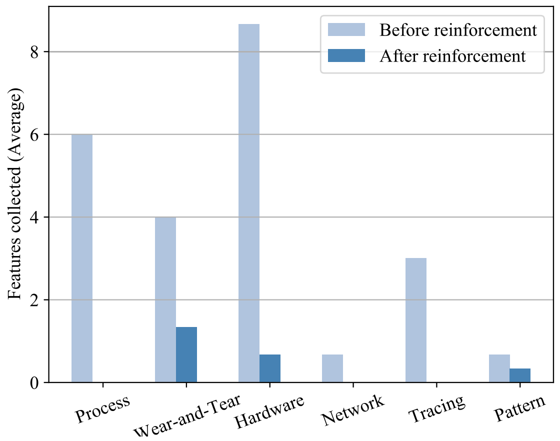

Fig. 4. Average Environmental features before and after reinforcement.

tool in multiple environments, we get the results. To make the results intuitive, we adopt the average value of three sandboxes, shown in Fig. 4.

As we can see from Fig. 4, the features collected by this tool show great discrepancy. After reinforcement, most of the features are hidden from the tool by pre-deployed modules. Process features and tracing features disapear by filter. Network accessibility is successfully disguise by emulator. As for other features, most of them drop down to a very low level. Here is the reason why there exist several features hard to hide. For wear and tear features, installation date and update date are hard to disguise because these two features depend on filesystem. For hardware features, blocks of disk are hard to hide in that these features rely on disk status. For particular patterns, the processname in */proc/self/status* and */proc/self/status* are hard to redirect due to the difficulty of changing state of process. Even though, this method is able to hide most of these features from six aspects and effectively reinforce sandboxes against environmental-sensitive malware, making the environment indistinguishable from normal systems.

C. Qualitative comparison with other countermeasures

There exist some efforts against environmental-sensitive. Some researchers aim at activating environmental-sensitive

TABLE IX
COMPONENTS OF THREE LINUX SANDBOX

Sandbox	Supported Architecture	Default OS	Tracer	Traffic Capture	Network Emulator
Cuckoo	QEMU/VMWare/VirtualBox/ESX/Xen/AVD/KVM	Ubuntu	Systemtap	TCPDUMP	INetSim
Lisa	QEMU	Ubuntu	Systemtap	TCPDUMP	/
Limon	VMWare	Ubuntu	Strace&Sysdig	TCPDUMP&Tshark	INetSim

malware. Symbolic execution can obtain feasible inputs and execute specific regions of codes. Forced execution can traverse possible paths of a program to explore all branches of a program. These two methods are proved effective in the lab, but both of them have some inherent flaws. On the one hand, those two methods have heavy time costs for they need to traverse part or all of the branches of a program instead of executing a specific branch once. On the other hand, the monitoring modules are not easy to cooperate with traversing progress as the order of collected information is perturbed by process state resetting. For instance, the sequence of network traffic is significant, which represents specific network behavior. However, if we traverse a program to entail it to expose more behavior, the process state recovers to the previous node so that it can explore another branch, which makes the order of traffic information after this node descends into chaos because both of the two statuses are recorded. Some researchers try to utilize multiple environments to execute samples to induce their behavior, which is resource-consuming. To make sure malware shows its behavior as rich as possible, various environments need to be preinstalled in emulators. In addition, due to unknown tactics of malware, this method cannot cover all possible methods to evade detection. By contrast, our method analyzes statistical data from actual behavior and finds common behaviors to detect the environment and only execute once.

V. LIMITATION AND DISCUSSION

Through the results of these experiments, we conclude that EnvFaker is able to reinforce sandbox. However, this method is not a complete reinforcing solution against environmental-sensitive malware, but an indispensable supplement for sandbox.

Firstly, Samples using discrepancy of specific instructions such as RedPill and cpuid can still detect the sandbox. Function hooking techniques are hard to apply to this situation. Virtual machine API offer some feasible ways to modify hardware information. But that should be configured out of virtual machine. In addition, using the difference of TSC timer can detect sandbox too, which is related to VM mechanism and of great complexity to avoid. Our method tries to mitigate the probability of being detected as possible as we can. The second limitation is the counts of wear and tear features. There are so many features in files or software in Linux OS representing the level of wear and tear, some of which are even encrypted such as */var/log/lastlog*, some of which are related to installation time of filesystem such as latest updating time of system. Our method chooses representative wear and tear features to hook. Thirdly, network emulation is a double-edged sword. By network emulation, some malwares using C&C are induced to expose malicious behaviors indeed. However, these malwares can detect the emulation by connecting to a definite non-existing network device deliberately. It seems to be an insoluble problem in dynamic analysis. Luckily, this is just a hypothesis because we have not found any Linux malware of that type for now. We seek to deal with these issues in the future.

VI. RELATED WORK

Security workers traditionally analyze unknown malware samples by virtual machine or sandbox, which give them an overview of malicious behaviors. So, malware authors try their best to make analysis process hard by equip the malware with detecting module to check if they are in sandbox. Under this situation, some researchers have found several VM-detection and sandbox-detection methods. In addition, several countermeasures have been proposed against environment-sensitive malwares.

VM-Detection Methods. T Raffetseder et al. [4] analyzed the possibilities to detect system emulators. Some known attacks against widely used virtual machine, such as Red-Pill [23] and LDT check were described by P Ferrie [24] . Xu Chen et al. [2] developed a detailed taxonomy of malware defender fingerprinting methods. An automatic technique to generate red-pills for detecting if a program is executed through a CPU emulator was proposed by RPaleari et al. [25] L Martignoni et al. [26] presented a testing methodology for CPU emulators based on fuzzing. Two method of artifacts and instruction emulation were concluded by O Bazhaniuk et al. [27].

Sandbox-Detection Methods. Paul Jung [28] introduced common sandboxes detection tricks used in the wild by malwares. Black-box technique to efficiently extract emulator fingerprints without reverse-engineering was proposed by J Blackthorne et al. [11] A Yokoyama et al. [1] introduce SandPrint, a program that measures and leaks characteristics of Windows-targeted sandboxes. A class of sandbox evasion techniques that exploit the wear-and-tear that inevitably occurs on real systems as a result of normal use was presented by N Miramirkhani et al. [3]

Anti-detection Methods. A system that allows us to explore multiple execution paths and identify malicious actions that are executed only when certain conditions are met were proposed by A Moser et al. [29] J Wilhelm, T Chiueh et al. [30] described a system named Limbo that features a forced sampled execution approach to traverse the driver's control flow graph. An automated technique to dynamically modify the execution of a whole-system emulator to fool a malware sample's anti-emulation checks was proposed by MG Kang et al. [31] M Lindorfer et al. [6] proposed a technique for detecting malware samples that exhibit semantically different behavior across different analysis sandboxes. A new VM-aware detection scheme, namely Divergence Detector was proposed to address the swindle of the evolved malware by CW Hsu et al. [7] K Liu et al. [10] proposed a fingerprints randomization methodology to exploits hooking techniques to defeat sandbox-aware malware. A binary analysis engine named X-Force was introduced to explores different execution paths inside the binary by F Peng et al. [32] O Ferrand [5] try to prevent malware to detect that they are under analyze with a few modifications and tricks on Cuckoo and the virtual machine. F Besler et al. [33] introduced countering sandbox evasion techniques used by malware. A system UBER for automatic artifact generation based on the emulation of real user behavior was proposed by P Feng et al. [9] W You et al. [34] proposed a memory pre-planning scheme before the real execution to practice forced execution.

VII. Conclusion

In this paper, we present EnvFaker, an effective method to reinforce sandbox against environmental-sensitive malware. Unlike traditional work, our method focuses on Linux sandbox. Based on previous work and characteristics of Linux OS, we analyze thousands of Linux ELF samples. We filter out improper features and extract several new features of Linux OS and summarize these features from 6 aspects to detect Linux sandbox. Given these, we make use of these features to implement a method to prevent malwares from detecting environmental features by target tracer, features filter and features emulator. To prove the effectiveness of these features, we implement a tool to show the discrepancy of the new and used physical machine and the difference among physical machine, virtual machines and sandboxes. To evaluate our method, we use the tool to detect features in sandboxes before and after reinforcement. The experiment shows that our method can effectively reinforce existing Linux sandboxes.

Acknowledgment

We would like to thank our mentor and other teammates. In the thesis work, Professor Weiping Wen provides great guidance from paper writing to research direction. We appreciate that every person in our research team gives us help as much as possible, not only in experiments but also in mind. Besides, we would like to express thanks to VirusTotal and VirusShare for their kindness in sharing malware samples. This work has been partly supported by NSFC 61872011.

References

[1] A. Yokoyama, K. Ishii, R. Tanabe, Y. Papa, K. Yoshioka, T. Matsumoto, T. Kasama, D. Inoue, M. Brengel, and M. Backes, "Sandprint: Fingerprinting malware sandboxes to provide intelligence for sandbox evasion," in *International Symposium on Research in Attacks, Intrusions, and Defenses*. Springer, 2016, Conference Proceedings, pp. 165–187. I, II, VI

[2] X. Chen, J. Andersen, Z. M. Mao, M. Bailey, and J. Nazario, "Towards an understanding of anti-virtualization and anti-debugging behavior in modern malware," in *2008 IEEE international conference on dependable systems and networks with FTCS and DCC (DSN)*. IEEE, 2008, Conference Proceedings, pp. 177–186. I, VI

[3] N. Miramirkhani, M. P. Appini, N. Nikiforakis, and M. Polychronakis, "Spotless sandboxes: Evading malware analysis systems using wear-and-tear artifacts," in *2017 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2017, Conference Proceedings, pp. 1009–1024. I, II, VI

[4] T. Raffetseder, C. Krügel, and E. Kirda, "Detecting system emulators," in *International Conference on Information Security*, 2007. I, VI

[5] O. Ferrand, "How to detect the cuckoo sandbox and to strengthen it?" *Journal of Computer Virology and Hacking Techniques*, vol. 11, no. 1, pp. 51–58, 2015. I, VI

[6] M. Lindorfer, C. Kolbitsch, and P. M. Comparetti, "Detecting environment-sensitive malware," in *International Workshop on Recent Advances in Intrusion Detection*. Springer, 2011, Conference Proceedings, pp. 338–357. I, VI

[7] C.-W. Hsu and S. W. Shieh, "Divergence detector: A fine-grained approach to detecting vm-awareness malware," in *2013 IEEE 7th International Conference on Software Security and Reliability*. IEEE, 2013, Conference Proceedings, pp. 80–89. I, VI

[8] X. Jia, G. Zhou, Q. Huang, W. Zhang, and D. Tian, "Findevasion: an effective environment-sensitive malware detection system for the cloud," in *International Conference on Digital Forensics and Cyber Crime*. Springer, 2017, Conference Proceedings, pp. 3–17. I

[9] P. Feng, J. Sun, S. Liu, and K. Sun, "Uber: Combating sandbox evasion via user behavior emulators," in *International Conference on Information and Communications Security*. Springer, 2019, Conference Proceedings, pp. 34–50. I, VI

[10] L. Ke, L. Shuai, and C. Liu, "Poster:fingerprinting the publicly available sandboxes," *ACM*, 2014. I, VI

[11] J. Blackthorne, A. Bulazel, A. Fasano, P. Biernat, and B. Yener, "Avleak: fingerprinting antivirus emulators through black-box testing," in *10th USENIX Workshop on Offensive Technologies (WOOT 16)*, 2016, Conference Proceedings. I, II, VI

[12] V. Total, "Virustotal-free online virus, malware and url scanner," *Online: https://www. virustotal. com/en*, vol. 2, 2012. II

[13] J.-M. Roberts, "Virus share," *Automatic Analysis of Malware Behaviour using Machine Learning*, 2014. II

[14] G. Combs, "Tshark—dump and analyze network traffic," *Wireshark*, 2012. II, III

[15] D. A. Joseph, V. Paxson, and S. Kim, "tcpdump tutorial," *University of California, EE122 Fall*, 2006. II

[16] F. C. Eigler and R. Hat, "Problem solving with systemtap," in *Proc. of the Ottawa Linux Symposium*. Citeseer, 2006, Conference Proceedings, pp. 261–268. II

[17] D. Kennedy, J. O'gorman, D. Kearns, and M. Aharoni, *Metasploit: the penetration tester's guide*. No Starch Press, 2011. II

[18] D. Uhrıcek, "Lisa–multiplatform linux sandbox for analyzing iot malware," 2020. II, IV-A

[19] D. Cortesi, "Pyinstaller manual," *Online]. Disponible en https://pyinstaller. readthedocs. io/en/stable/[Último acceso el 17 de noviembre de 2020]*. II

[20] T. Hungenberg and M. Eckert, "Inetsim: Internet services simulation suite," *Internet*, 2014. III-C

[21] D. Oktavianto and I. Muhardianto, *Cuckoo malware analysis*. Packt Publishing Ltd, 2013. IV-A

[22] K. Monnappa, "Automating linux malware analysis using limon sandbox," *Black Hat Europe*, vol. 2015, 2015. IV-A

[23] J. Rutkowska, "Redpill: Detect vmm using (almost) one cpu instruction," *http://invisiblethings. org/papers/redpill. html*, 2004. VI

[24] P. Ferrie, "Attacks on virtual machine emulators," 2007. VI

[25] R. Paleari, L. M. Giampaolo, F. Roglia, and D. Bruschi, "A fistful of red-pills: How to automatically generate procedures to detect cpu emulators," *harvard theological review*, 2009. VI

[26] L. Martignoni, R. Paleari, G. F. Roglia, and D. Bruschi, "Testing cpu emulators," in *Proceedings of the Eighteenth International Symposium on Software Testing and Analysis, ISSTA 2009, Chicago, IL, USA, July 19-23, 2009*, 2009, Conference Proceedings. VI

[27] O. Bazhaniuk, Y. Bulygin, A. Furtak, M. Gorobets, J. Loucaides, and M. Shkatov, “Reaching the far corners of matrix: generic vmm fingerprinting,” *SOURCE Seattle*, 2015. VI

[28] P. Jung, “Bypassing sanboxes for fun!” in *Bot Conf*, 2014, Conference Proceedings. VI

[29] A. Moser, C. Kruegel, and E. Kirda, “Exploring multiple execution paths for malware analysis,” in *2007 IEEE Symposium on Security and Privacy (SP’07)*. IEEE, 2007, Conference Proceedings, pp. 231–245. VI

[30] J. Wilhelm and T.-c. Chiueh, “A forced sampled execution approach to kernel rootkit identification,” in *International Workshop on Recent Advances in Intrusion Detection*. Springer, 2007, Conference Proceedings, pp. 219–235. VI

[31] M. G. Kang, H. Yin, S. Hanna, S. McCamant, and D. Song, “Emulating emulation-resistant malware,” in *Proceedings of the 1st ACM workshop on Virtual machine security*, 2009, Conference Proceedings, pp. 11–22. VI

[32] F. Peng, Z. Deng, X. Zhang, D. Xu, Z. Lin, and Z. Su, “X-force: Force-executing binary programs for security applications,” in *23rd USENIX Security Symposium (USENIX Security 14)*, 2014, Conference Proceedings, pp. 829–844. VI

[33] F. Besler, C. Willems, and R. Hund, “Countering innovative sandbox evasion techniques used by malware,” in *29th Annual FIRST Conference*, 2017, Conference Proceedings. VI

[34] W. You, Z. Zhang, Y. Kwon, Y. Aafer, F. Peng, Y. Shi, C. Harmon, and X. Zhang, “Pmp: Cost-effective forced execution with probabilistic memory pre-planning,” in *2020 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2020, Conference Proceedings, pp. 1121–1138. VI