UBER: Combating Sandbox Evasion via User Behavior Emulators

Pengbin Feng[1]✉, Jianhua Sun[2], Songsong Liu[1], and Kun Sun[1]

[1] George Mason University, Fairfax, VA, USA
{pfeng4,sliu23,kun3}@gmu.edu
[2] College of William and Mary, Williamsburg, VA
jianhua@cs.wm.edu

Abstract. Sandbox-enabled dynamic malware analysis has been widely used by cyber security teams to handle the threat of malware. Correspondingly, malware authors have developed various anti-sandbox techniques to evade the analysis. Most of those evasion techniques are well studied and can be defeated with appropriate mitigation strategies. However, one particular technique is usually overlooked and can be extremely effective in defeating sandbox-based malware analysis, i.e., usage artifacts analysis. This technique leverages a variety of system artifacts that are expected to exist in a real system as a result of typical user activities for sandbox environment identification. To tackle this drawback of lacking authentic system artifacts in existing sandbox designs, in this paper we propose a novel system UBER for automatic artifact generation based on the emulation of real user behavior. Instead of cloning real usage artifacts or directly simulating user behaviors, UBER generalizes the user's computer usage pattern with an abstract behavior profile, employs the profile to guide the simulation of user actions and the generation of artifacts, and then clones the system with generated artifacts into the sandbox environment. We implement a prototype of UBER and verify the effectiveness of the generated artifacts. The experimental results further demonstrate that UBER can effectively mitigate the system artifacts based sandbox evasion and significantly increase the difficulty for the attacker to distinguish the sandbox from the real user system.

Keywords: Malware Analysis · Evasive Malware · Anti-Analysis · Virtualization

1 Introduction

Malware sandboxes have become a highly desirable and widely utilized tool through which most cyber security teams regularly perform malware analysis. They can provide effective analysis of malware by monitoring its runtime behaviors at various levels. Sandboxes allow inexperienced analysts to identify malicious features, which could not be obtained through

malware reverse engineering. In addition, many security companies adopt sandboxes for unknown downloads analysis [1–3].

However, the arm race between attacker and defender never stops. As cyber security teams begin to rely more and more on sandbox-based dynamic analysis, malware authors have begun to develop ever-more sophisticated evasion techniques to circumvent sandboxes. Specific environment indicators, such as system settings [4], analysis instrumentation files or drivers [5], user-like mouse clicking and scrolling movements [11] and derived sandbox configuration [10], as well as timing attacks [6], CPU virtualization [8] and process introspection [7] could be adopted by malware to identify sandbox environments.

Many of these evasion techniques have been identified and well documented by security teams. Furthermore, researchers have proposed several mitigation approaches, such as state modification, multi-platform record&replay and bare metal analysis, to make sandbox environment indistinguishable from the real system [9]. However, all these anti-sandbox techniques are ineffective in mitigating usage artifacts analysis based sandbox evasion [19], which leverages user usage artifacts to determine whether malware is running on actual system or sandbox.

In this paper, we seek to tackle the drawback of lacking real user activity related system artifacts in existing sandbox systems. One straightforward strategy is to construct the sandbox by directly cloning the real user system. However, there are several limitations with this approach. First, copying artifacts from real user system could potentially threat user privacy. Second, the artifacts in the cloned system would become outdated eventually without consistent user interactions, and it is unrealistic to clone the real system for each analysis. Another alternative strategy is to directly simulate real user interactions within the sandbox environment. Although this approach does not suffer from privacy issues, it is unclear whether the generated artifacts are sufficiently, as many artifacts are accumulated through the history of persistent user access. Motivated by these two strategies, we propose a novel architecture, the User Behavior Emulator (UBER), which employs a user behavior profile to generate "real" user activities. Specifically, instead of cloning artifacts from real system or directly simulating user behaviors, UBER generalizes the user's computer usage pattern into an abstract behavior profile, emulates the user actions based on the profile to generate system artifacts, and finally copies the generated artifacts to the sandbox environment.

The main intuition behind the proposed strategy is that simulating user actions from usage pattern allows to faithfully replicate real user be-

havior, thus generating realistic usage artifacts. For example, in a Windows system, multiple artifacts, such as the registry entries, the system logs and the cached files, could be generated due to real user actions. Due to the considerations of user privacy, UBER only records a user's application usage times with the tracker software (e.g. ManicTime[1]) to generalize usage pattern. For the application operation, UBER relies on the statistics data from public websites (e.g. Alex[2], Google Trends[3]) and performs generic user actions, such as accessing top sites, searching common terms. Thus, UBER generates realistic artifacts based on specific usage pattern with generic operations. These artifacts collectively present a holistic profile of the system usage pattern.

To avoid runtime conflicts between the malware and UBER, we deploy UBER on one always-on system, which is cloned to the malware analysis sandbox on demand. In this way, the sandbox environment is rendered indistinguishable from the real system. Since there is clear distinction of system artifacts between a sandbox system running specialized analysis software and the real system, the artifacts in the cloned system will become obsolete eventually without persistent real user actions. Therefore, UBER uses a scheduler to perform the cloning regularly so that the artifacts are not outdated. Given that a sandbox-based malware analysis system is usually rolled back to its initial state after each malware analysis [10], UBER mainly replaces this initial state with the always-on system that possesses the most up-to-date and realistic artifacts.

To demonstrate the effectiveness of the proposed architecture, we implement a prototype of UBER with python script. To assess the usefulness of the generated artifacts, we deploy UBER on a virtual machine with fresh installed Windows Operating System (OS), and manually operate the cloned fresh virtual machine as "real system" simultaneously. After running these two systems for one month, we compare the artifacts accumulation process of them. Overall comparable amounts of system artifacts are accumulated in both systems, which indicates that UBER can effectively generate realistic artifacts through the emulation of real user operations.

In conclusion, this paper makes the following contributions:

- We present a comparative study of the malware sandbox evasion techniques that leverage system artifacts indicating normal usage for sandbox detection.

[1] https://www.manictime.com/
[2] https://www.alexa.com/topsites
[3] https://trends.google.com/trends/

- As a countermeasure, we propose the User Behavior Emulators (UBER) for realistic usage artifacts generation based on the predefined user profile without violating user privacy. UBER manages to create high-fidelity sandbox environments that are indistinguishable from real systems.
- We implement a prototype of UBER and our evaluation results demonstrate its effectiveness in mitigating the system artifacts based sandbox evasion.

2 Threat Model

Sandbox evasion malware usually leverages a variety of techniques to determine whether it is running in a sandbox before performing malicious behavior. This kind of malware would camouflage as "benign" through executing normal function when identifying sandbox system. The ability of hindering malware analysis could greatly harm the effectiveness of current computer system defense mechanisms. In this paper, we focus on the sandbox evasion malware that leverages system artifacts indicating normal user usage to distinguish the sandbox environments from the real systems. In general, UBER aims at preventing the malware from distinguishing a sandbox system from the real system through usage artifacts analysis.

3 System Design

To defeat the sandbox evasion with usage artifacts analysis, we develop a user behavior emulation system UBER for automatic artifacts generation. Fig. 1 shows the system architecture of UBER.

In the following, we first give an overview of UBER and then elaborate on the design details. **Data Collector** gathers the raw user data (e.g., the web access log) which characterizes user behavior. **User Profile Generator** performs statistical and correlation analysis on the raw data to generate an abstract profile that serves as a generalized representation of typical user activities and outputs a configuration file that describes the user profile. This configuration file is then fed to the **Artifact Generation OS**, which uses the **Event Generator** to create events following the configuration and executes them via the **Event Executor**. Finally, continuous operation of the **Event Generator** results in various seemingly "real" system artifacts in the **Artifact Generation OS**, which can then be cloned to create the malware sandbox analysis environment with realistic OS image.

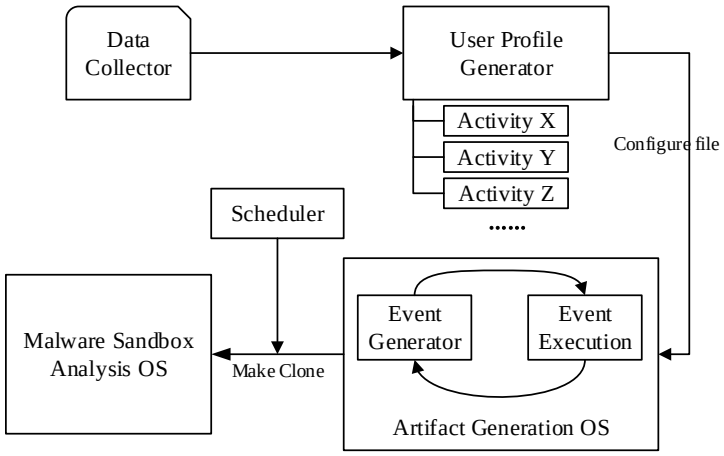

Fig. 1. UBER Architecture

3.1 Data Collector

This component gathers the information needed to derive the user profile. First, it records the user's application usage times with a tracker software and then categorizes each application into the activity types defined by UBER. Second, it collects useful information from several public web usage repositories to define the operations of each activity type. In this paper, we define the activity types according to an individual's real computer usage experience in combination with public stats of popular web[4] and application activities[5]. As an example, we collect data from public statistics website (e.g. Alexa, Google Trends) to construct generic user web activities.

3.2 User Profile Generator

This component translates the information gathered by **Data Collector** into various user activities. It outputs a configuration file that defines how the emulation software will perform user actions. Particularly, the configuration file consists of the application and web usage behavior with corresponding execution probabilities, which are derived from a statistical analysis of the information collected by **Data Collector**. These probability values indicate the likelihood that a user would perform specific activities (application or web).

[4] https://www.infoplease.com/science-health/internet-statistics-and-resources/most-popular-internet-activities
[5] https://www.microsoft.com/en-us/store/most-popular/apps/pc

Fig. 2 shows an example of configure file which represents one common staff who regularly starts to work approximately at 8am and 1pm of everyday. In this configure file, the web browsing usage of 60 indicates that this particular user will likely spend approximately 60% of computer usage time performing web browsing task.

```
# System usage (Start Time,Duration)
    onTimes: 0800+0100-0100,210
    onTimes: 1300+0030-0030,270
# Activity type of user (Type,Probability)
    ActivityTypes: web,60|app,40
# Sub-activities for Web Activity
    WebActivityTypes: searching,60|webmail,20|new,10|miscellaneous,10
# Sub-activities for App Activity
    AppActivityTypes: productivity,70|Leisure,20|miscellaneous,10
# Browsers for Web Activity
    Browsers: iexplore,10|chrome,50|firefox,40
# WebSites for Sub-activities of Web Activity
    SearchSites: www.google.com,70|www.bing.com,30
    WebMailSites: mail.google.com,60|outlook.live.com/mail,20|mail.yahoo.com,20
    NewsSites: www.cnn.com,60|www.bbc.com,20|www.foxnews.com,10
    MiscSites: (Pull from Alex)
# Actions for Sub-activities of Web Activity
    SearchTerms: (Pull from Google Trend)
```

Fig. 2. Configure File Example

3.3 Artifact Generation OS

The Artifact Generation OS directly runs the emulation software to generate realistic usage artifacts. Furthermore, to keep system artifacts up-to-date, the emulation software is executed continuously based on the configuration file. This software consists of two modules: **Event Selector** and **Event Executor**. We do not execute the malware on this OS to eliminate runtime conflicts. To maintain the applicability of the artifacts, the virtual machine (VM) running this OS is never reverted to the previous snapshots. In the following, we first introduce the artifacts we have identified that characterize the usage history of a real system, and then present the workflow of the **Event Selector** and **Event Executor**.

System Artifacts We identify and collect a multitude of useful system artifacts that can be leveraged by the malware to differentiate the sandbox from the real system. Table 1 presents a list of the system usage artifacts used in our experiments.

Category	Artifacts	Description
File System	Downloaded Files	# of download files in computer
Browser	Total URLs visited	# of unique visited URLs
	Unique Domains	# of unique visited domains
	Cookies	# of cookie in browsers
	CookiesTime	# of days since first cookies
	Bookmarks	# of bookmarks in browsers
	Temporary Internet Files	# of temporary files generate by browser
Network	ARP Entries	# of ARP entries
	DNS Records	# of DNS resolver entries
	Bytes Sent	# of count data sent
	Active Connections	# of active TCP/UDP connections
Registry	MUI Cache	# of MUI entries
	Userassist Entries	# of Userassist entries
	MRU Entries	# of MRU entries
	Registry Size	Size of registry (in bytes)
System	System Log Entries	# of system events
	Application Log Entries	# of application events

Table 1. A list of system artifacts

File System: Typically, a variety of files are created and modified during the normal usage of a computer system.

Browser: A user's daily browsing also generates multiple traces that indicate the browser usage history. In particular, normal browser usage usually results in abundant other files such as the temporary internet files, bookmarks and cookies. These files serve as a strong indicator of the regular browser usage, which can be used by the malware to identify normal user activities. Furthermore, the stored bookmarks and cookies in the browser could serve the same purpose as the temporary internet files.

Network: Various network related artifacts are associated with an authentic system. The amount, type and variety of network information from a sandbox system will be vastly different from the real system that has been used to browse web sites, execute various client applications and install OS/application updates.

Registry: The Windows Registry contains a lot of information about the computer system and its usage. Among these, three registry items, Userassist, Most Recently Used (MRU) and MUICache keys, are particularly informative. Userassist keys contain information about the applications accessed via the GUI. Most Recently Used (MRU) keys contain information of the recently accessed or saved files, such as zip files (.zip), text files (.txt) or graphic files (.jpg, .png, etc.). MUICache key is another way to determine the software previously run on the system. Due to the running specialized analysis software and the lacking of abundant operations, fairly limited number of these registry entries are accumulated in the sandbox systems.

System: The event logs of the Windows systems contain plenty of information about the current state and past usage of the system. Event log records various types of events, such as the application, security, setup and update, etc. Considering that the event categories are diverse among different systems, we only enumerate the number of the application and system events to generalize usage pattern.

Event Selector & Event Executor The workflow of the **Event Selector** and **Event Executor** is illustrated in Fig. 3.

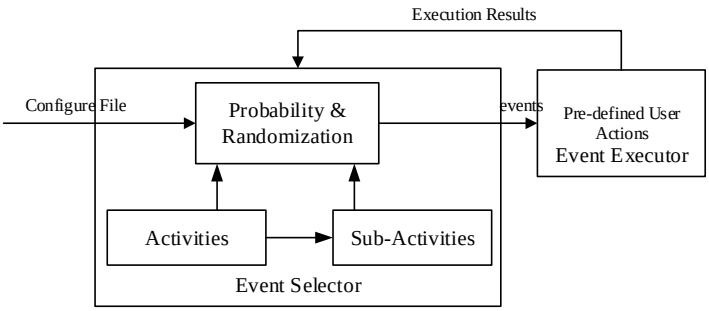

Fig. 3. Workflow of Event Selector and Event Executor

Event Selector: This module makes decisions on which events will be performed according to the configuration file. Each event generates different subsequent events which are based on the output of each primary event. The Probability and Randomization function within this module mainly takes probabilities and additional variables provided by the configure file to select the activities and the corresponding sub-activities.

This function mainly determines the event to be executed based on the probability that indicates which and how user actions will be performed. In this paper, event represents specific behaviors of sub-activity.

The Probability and Randomization function works by following the Algorithm 1. This function takes configuration file, the predefined activity /sub-activity types as inputs, and outputs the list of selected events based on the usage pattern. It firstly loads configuration file to obtain the probability and average duration of the activities and sub-activities. Then, it selects the activity and further sub-activity according to corresponding probability. Finally, it selects specific new sub-activity based on the operation results of sub-activity. In the above process, we record the time of each sub-activity and activity and make sure these times do not exceed the limits defined in the configuration file.

Algorithm 1 Probability and Randomization function

Input: Configuration file of user profile, $config$; Predefined user activity types, *types*; Predefined sub-activity types, *subtypes*;
Output: Selected event list, E;

Loading user profile from $config$ to $User$;
$E = \emptyset$;
while $sys.runtime < User.sys.time$ **do**
  select *type* from *types* based on $User.type.probability$;
  while $type.runtime > User.type.time$ **do**
    select another *type* from *types*;
  while $type.runtime < User.type.time$ **do**
    selects *subtype* from *type.subtypes*;
    while $subtype.runtime > User.subtype.time$ **do**
      select another *subtype* from *type.subtypes*;
    while $subtype.runtime < User.sbutype.time$ **do**
      perform operation of *subtype*;
      $E = E \cup subtype$;
      Obtain *results* from *subtype* operation;
      while *results* contains $subtype_{new}$ **do**
        perform operation of $subtype_{new}$;
        $E = E \cup subtype_{new}$;
        Obtain *results* from $subtype_{new}$ operation;
      Recording runtime for each *subtype*;
    Recording runtime for each *type*;
return E;

Event Executor: This module is responsible for executing the events determined by the **Event Selector**. Each event performs predefined user actions. **Event Selector** provides all the variables needed to perform cor-

responding actions. The outputs of these actions, such as results returned from a particular web search, are then returned to the **Event Selector** to make a new decision. Once the new decision is made, it is returned to the **Event Executor** module again and the process continues.

3.4 Malware Sandbox Analysis OS

The Malware Sandbox Analysis OS is where the malware is executed and the runtime information is gathered to derive the behavior of mawlare. It is a real-time clone of the **Artifact Generation OS** including the already generated realistic artifacts, leading to the inability of malware identifying the analysis environments. The emulation software should not be executed on this VM, as the software would compete resources with the malware, influence the obtained malicious features and even interfere with subsequent analysis. Furthermore, malware can further evade detection through identifying the execution of the emulation software.

3.5 Scheduler

The Scheduler is responsible for creating a copy of the **Artifact Generation OS** that is used as the malware analysis sandbox. Once copied, the configurations of the **Malware Sandbox Analysis OS** sandbox are also updated to use the most recent OS image. The copy process can be done in several minutes, which allows a newly updated VM to be used before each analysis. However, performing a VM copy for each analysis significantly increases the analysis time and may not be feasible in practice. An alternative solution is to clone the VM following a schedule (e.g. daily) or use a service to monitor the analysis system usage and replicate the VM during the system downtime. Our implementation of UBER performs the cloning on schedule when the malware analysis sandbox is idle.

4 Implementation

We implemented a prototype of UBER on Window OS using python script. UBER uses python packages selenium [23], pywin32 [24] and pywinauto [25] to implement the automated control of the browser and other applications. The implementation architecture is shown in Fig. 4. We recruit several volunteers and collect their computer application usage pattern via using the application usage tracker software ManicTime and then obtain generalized user profiles. Although these profiles represent

realistic computer usage pattern, they may not be universally suitable. Taking the configuration file and the data from Google Trends and Alexa sites, UBER randomly selects the browser or application activity and further selects sub-activities to emulate user operations.

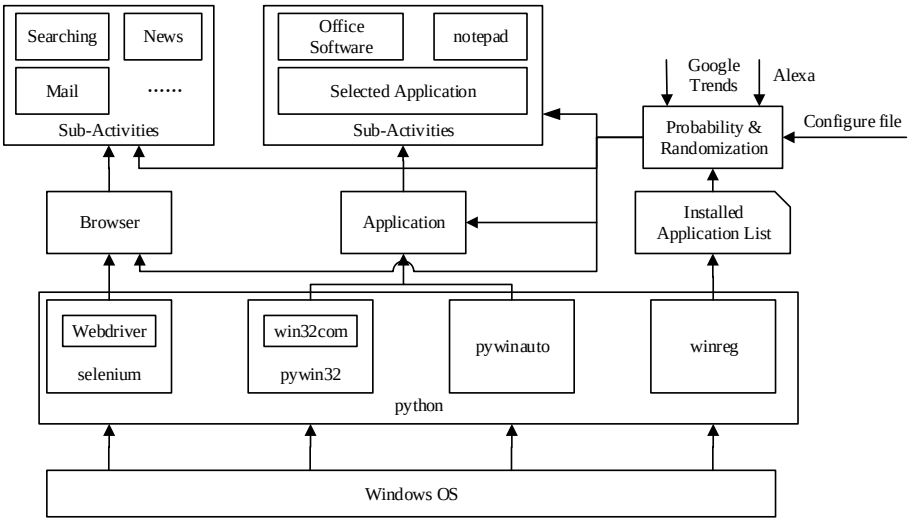

Fig. 4. Implementation Process of UBER

To emulate realistic user activities, UBER performs all user interactions including scrolling, mouse clicking and keyboard input in human-like speed. In addition, UBER emulates user behavior based on user profile which ensures the execution of web and application activities in human-like habits and ways. We manually parse the commonly accessed website, e.g., such as Google, Bing, CNN, and BBC, to help UBER extract meaningful URLs from the searching results and news pages. For example, UBER selects Internet Explorer browser to perform Google search. It employs the method "$find\_elements\_by\_xpath(\text{"}//*[@id=' rso']/div/div//*/div[@class=' rc']/div[1]/a\text{"})$" provided by selenium to extract effective Google searching results. Then, UBER selects one or multiple result pages to browser. We also manually parse the GUI elements from popular applications such as Notepad and PDF reader to help UBER perform realistic operations. Moreover, it also extracts the text paragraphs from the pre-chosen websites and then performs text editing using applications such as Office Word and Notepad. To determine the

application to be executed, UBER looks up the Windows registry items to extract installed application list, and then randomly selects one.

5 Evaluation

In this section, we first demonstrate the differences of system usage artifacts between the sandbox environments and the real systems. We then evaluate the effectiveness of UBER in artifacts generation and comparison with other mitigation strategies.

5.1 Artifacts Difference

We implement an automation script with NirSoft[6] to collect the artifacts introduced in Section 3.3 from the Windows OS. Specifically, we use the script to collect the artifacts from multiple available sandbox systems and real user systems. Table 2 shows the average values of each artifact for the two categories of systems along with their differences. We can see that there exists clear distinction in these artifacts between a sandbox system and a real system. This demonstrates that the identified artifacts can serve as strong indicators of the sandbox environment, which can be exploited by the malware to differentiate a sandbox from the real system.

5.2 Measurement

To assess the effectiveness of UBER in defeating artifacts analysis-based sandbox evasion, we extracted the artifacts from several VMs with fresh installations of Windows OSes as baseline. They include Windows 10, Windows 7, Windows Vista, Windows 8, Windows 8.1 and Windows XP, which covers all available Windows versions commonly used by normal users, according to a recent statistical report [22]. In this experiment, the host system is a PC installing Ubuntu 18.04 LTS, and is featured with Intel Xeon(R) E5-2620 CPU @ 2.40GHz x 12 and 16 GB memory. The VMs are deployed using VirtualBox 6.0, each of which is configured with 3 vCPUs and 4GB memory.

The objective of this experiment is to demonstrate that the artifact values from the sandbox systems are indistinguishable from the real systems once we deploy UBER in the sandboxes. Specifically, We deploy UBER on the VMs with freshly installed OSes, which serve as the sandbox systems as previously mentioned. As for comparison, we manually

[6] https://www.nirsoft.net/

Artifacts	Sandbox	Real Systems	Difference
Downloaded Files	0	27	27
Total URLs visited	3	301	298
Unique Domains	1	55	54
Cookies	0	71	71
CookiesTime	N/A	310	310
Bookmarks	0	921	921
Temporary Internet Files	0	851	851
Arp Entries	0	13	13
DNS Records	0	44	44
Bytes Sent	2,731,035	43,007,337	40,276,302
Active Connections	8	54	46
MUI Cache	2	211	209
Userassist Entries	33	62	29
MRU Entries	57	433	376
Registry Size	52,521,688	73,218,690	20,697,002
System Log Entries	774	1715	841
Application Log Entries	293	1290	997

Table 2. Artifacts Difference between Sandbox and Real System

operate another set of cloned VMs that represent "real systems" with normal user access. We then continuously run these machines for one month and use our automatic script to calculate the artifacts values from these systems. The results are summarized in Table 3. We observe that the sandbox systems with UBER deployed have comparable amount of artifacts accumulated as real systems we manually access. Although there is variation of the artifact values between these two systems, they both possess realistic usage artifacts. In particular, the values of "CookiesTime" and "Arp Entries" in both systems are the same since they are created on the same day and stay in the same LAN network. In Table 3, the "Cookies" and "DNS Records" may be quite different among these two systems, this is mainly caused by that UBER accesses different URLs from real users. However, the malware cannot identify sandbox environment only through the different of URL access pattern because of it exists huge difference among different users. In conclusion, UBER is able to faithfully emulate real user operations and generate realistic artifacts in the sandboxes to make them indistinguishable from real systems.

5.3 Comparison with Other Mitigation Solutions

A variety of mitigation solutions have been proposed to combat malware sandbox evasion. Ether [26] provides a hypervisor-based analysis envi-

Artifacts	Baseline	Baseline + User Operation	Baseline + UBER
Downloaded Files	0	27	34
Total URLs visited	3	1786	1766
Unique Domains	1	373	354
Cookies	5	31	55
CookiesTime	0.8	30	30
Bookmarks	0	151	164
Temporary Internet Files	19	57	55
Arp Entries	8	8	8
DNS Records	8	10	14
Bytes Sent	2,124,684	5,225,592	5,012,932
Active Connections	6	50	46
MUI Cache	14	26	24
Userassist Entries	43	73	74
MRU Entries	17	128	136
Registry Size	87,030,444	92,026,650	91,356,255
System Log Entries	813	845	921
Application Log Entries	694	1124	1208

Table 3. Artifacts Comparison between Baseline, Baseline with UBER and Baseline with User Operation

ronment, which employs hardware-assisted virtualization to remove the emulation and software indicators. BareCloud [27] proposes a bare-metal analysis environment, which runs on real hardware devices and is able to provide transparent analysis. It executes malware in different analysis environments to investigate the evasion behaviors. The bare-metal method manages to create high-fidelity sandbox environment by eliminating the virtualization-related artifacts. However, these two mitigation methods are ineffective when encountering sandbox evasion malwares that utilize system artifacts. Due to the lack of real user activities, the system artifacts indicating past user access are seldomly generated in the sandbox systems. As a countermeasure, we propose UBER to generate realistic usage artifacts through user behavior emulation, which is orthogonal but complementary to existing mitigation solutions.

6 Limitations And Discussions

UBER is not a complete solution to counter sandbox evasion. When combined with existing mitigation solutions, it is an indispensable complement to create authentic sandbox systems that highly resemble real systems.

Data Collection The effectiveness of UBER depends on the collected data. It is usually sufficient to create realistic user profiles with generic operations from the publicly available data. However, if one malware targets a specific individual or organization, generic profiles may be invalid. Defining user profiles of specific targets could be used to hurdle this kind of targeted malwares.

Software Specific Artifacts UBER generates the most commonly identified user artifacts by emulating the relevant activities. This approach is not applicable to generating artifacts associated with proprietary, customized or otherwise less popular software. These softwares usually generate their own unique artifacts. To defeat the sandbox evasion that utilizes these software specific artifacts, we can modify UBER to emulate the software usage to generate those unique artifacts.

UBER Detection The execution of UBER in the **Artifact Generation OS** could also leave footprints, which can in turns be exploited by the malware to identify the sandbox. These footprints could be eliminated by removing the python because UBER is implemented all through python scripts. Moreover, the supporting software for sandbox deployment may generate traces. To alleviate this, we can adopt the techniques employed by the malware to conceal the emulation behaviors. In the case where UBER produces fixed patterns in the generated artifacts, we can adjust the installation and operation of UBER (e.g., randomizing the activity selection) to reduce the likelihood of sandbox identification.

Validation of Artifacts As stated in Spotless Sandboxes [19], the lack of normal usage in existing sandbox analysis system makes its environmental characteristics be very different from real system. UBER tries to fix this defects through user behavior emulation. In this paper, we verify that the generated usage artifacts are similar to real system in terms of statistical features. In fact, the attack surface of usage artifacts is huge. With the evolution of evasion techniques, malware could identify sandbox environment by leveraging a variety of artifacts, or even validating the content of artifacts like the correctness of documents. Therefore, we plan to integrate methods like FORGE [29] into UBER to generate validated artifacts for confusing malware in the future.

7 Related Work

Sandbox Evasion There is a wide variety of literature outlining both sandbox evasion techniques and corresponding mitigation strategies. Chen et al. [14] points out the widely application of sandbox evasion in nowa-

days advanced persistent threat (APT). Dilshan [11] provides an overview of the most common methods resisting sandbox evasion. Hassan et al. [12] details one common evasion technique: delaying malicious behavior execution to evade sandbox-based dynamic analysis. Chailytko et al. [13] lists many sandbox evasion techniques as well as ways to defeat them. These techniques include identifying features created by sandbox environment, configuration files, communication channels and other unique traces the analysis software would leave. Aim at countering usage artifacts analysis based evasion techniques, this paper proposes an anti-evasion technique through user behavior emulation, which could make complement to existing anti-evasion strategies.

Emulation When emulation is discussed, it typically regards to hardware emulation liking testing applications on different platforms, or software emulation to identify function bugs and other deficiencies in software. Kruegel [15] discussed some ways of sandbox evasion towards hardware emulation through virtualization, as well as methods to subvert them. Kaur et al. [16] and Renu et al. [21] both introduce one of the components of UBER, the selenium web driver. In these papers, emulation and replication software is for quality control and software development functionality. However, UBER takes another way and applies these emulation software to trick malware of "realistic" execution environment.

System Artifacts The identification of historical usage artifacts on one system has been recognized as sandbox evasion techniques. Spotless Sandboxes [19] identified many common, and easily accessible, historical usage artifacts generated by OS through normal user activities. Spotless Sandboxes found that the type of activities conducted on a sandbox OS varies greatly from a typical system, so do the generated artifacts. UBER follows this idea and tries to make up this defects by generating "real" historical artifacts through user activities emulation based on predefined usage pattern.

8 Conclusion

With the widely application of sandbox-based malware analysis, evermore sophisticated evasion techniques have been developed by malware authors to evade the sandbox. Among them, one effective approach is to leverage various artifacts generated during the normal usage of a system, which cannot be mitigated by state-of-the-art defense strategies. In this paper, we investigate the useful system artifacts and propose a novel anti-evasion system UBER, which can effectively generate realistic usage

artifacts based on predefined user profile. We implement a prototype of UBER and our evaluation indicates that UBER can generate artifacts through user behavior emulation. We do not intend to defeat all sandbox evasion techniques with UBER. Nevertheless, UBER serves as an essential supplement to the existing anti-evasion arsenal. Furthermore, UBER is flexible to leverage tailored user behavior data to construct user profiles of specific individuals or organizations, which can further improve its capability of handling increasingly sophisticated and targeted sandbox evasion techniques. In the future, we plan to integrate UBER into real-world malware analysis systems (e.g., Cuckoo Sandbox [28]) to build up high-fidelity sandbox environment.

Acknowledgements. This work is partially supported by ONR grants N00014-16-1-3214, N00014-16-1-3216, and N00014-18-2893.

References

1. McAfee Advanced Threat Defense, Advanced detection for stealthy, zero-day malware, `https://www.mcafee.com/enterprise/en-us/products/advanced-threat-defense.html`. Last accessed 14-Aug-2019.
2. Symantec Content - Malware Analysis, Detect and block advanced threats that elude traditional analysis with multiple-layer inspection and customizable sandboxing, `https://www.symantec.com/products/atp-content-malware-analysis`. Last accessed 14-Aug-2019.
3. FireEye Malware Analysis, Safely execute and analyze malware in a secure environment, `https://www.fireeye.com/solutions/malware-analysis.html`. Last accessed 14-Aug-2019.
4. Lindorfer, M., Kolbitsch, C. and Comparetti, P.M.: Detecting environment-sensitive malware. In International Workshop on Recent Advances in Intrusion Detection, pp. 338-357. Springer, Berlin, Heidelberg (2011).
5. Chen, X., Andersen, J., Mao, Z.M., Bailey, M. and Nazario, J.: 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), pp. 177-186. IEEE (2008).
6. Brengel, M., Backes, M. and Rossow, C.:Detecting hardware-assisted virtualization. In International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment, pp. 207-227. Springer, Cham (2016).
7. Blackthorne, J., Bulazel, A., Fasano, A., Biernat, P. and Yener, B.: AVLeak: fingerprinting antivirus emulators through black-box testing. In 10th USENIX Workshop on Offensive Technologies (WOOT 16). (2016).
8. Alwabel, A., Shi, H., Bartlett, G. and Mirkovic, J.: Safe and automated live malware experimentation on public testbeds. In 7th Workshop on Cyber Security Experimentation and Test (CSET 14). (2014).
9. Bulazel, A. and Yener, B.: A survey on automated dynamic malware analysis evasion and counter-evasion: PC, mobile, and web. In Proceedings of the 1st Reversing and Offensive-oriented Trends Symposium (p. 2). ACM (2017).

10. Yokoyama, A., Ishii, K., Tanabe, R., Papa, Y., Yoshioka, K., Matsumoto, T., Kasama, T., Inoue, D., Brengel, M., Backes, M. and Rossow, C.: SandPrint: fingerprinting malware sandboxes to provide intelligence for sandbox evasion. In International Symposium on Research in Attacks, Intrusions, and Defenses, pp. 165-187. Springer, Cham (2016).
11. Keragala, D.: Detecting malware and sandbox evasion techniques. SANS Institute InfoSec Reading Room, 16. (2016).
12. Mourad, H.: Sleeping your way out of the sandbox. SANS Institute InfoSec Reading Room. (2015).
13. Chailytko, A. and Skuratovich, S.: Defeating sandbox evasion: how to increase the successful emulation rate in your virtual environment. In ShmooCon 2017. (2017).
14. Chen, P., Desmet, L. and Huygens, C.: A study on advanced persistent threats. In IFIP International Conference on Communications and Multimedia Security, pp. 63-72. Springer, Berlin, Heidelberg (2014).
15. Kruegel, C.: Full system emulation: Achieving successful automated dynamic analysis of evasive malware. In Proc. BlackHat USA Security Conference, pp. 1-7. (2014).
16. Kaur, H. and Gupta, G.: Comparative study of automated testing tools: selenium, quick test professional and testcomplete. Int. Journal of Engineering Research and Applications, 3(5), pp.1739-1743. (2013).
17. Kirat, D. and Vigna, G.: Malgene: Automatic extraction of malware analysis evasion signature. In Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security, pp. 769-780. ACM (2015).
18. Gilboy, M.R.: Fighting Evasive Malware with DVasion (Doctoral dissertation). (2016).
19. Miramirkhani, N., Appini, M.P., Nikiforakis, N. and Polychronakis, M.: Spotless sandboxes: Evading malware analysis systems using wear-and-tear artifacts. In 2017 IEEE Symposium on Security and Privacy (SP), pp. 1009-1024. IEEE (2017).
20. Vashisht, S.O. and Singh, A.: Turing test in reverse: new sandbox-evasion techniques seek human interaction. (2014).
21. Renu, P. and Temkar, R.: Intelligent testing tool: Selenium web driver. International Research Journal of Engineering and Technology (IRJET). (2017).
22. GlobalStats - Desktop Windows Version Market Share Worldwide, `https://gs.statcounter.com/os-version-market-share/windows/desktop/worldwide`. Last accessed 14-Aug-2019.
23. Python package selenium, `https://pypi.org/project/selenium`. Last accessed 14-Aug-2019.
24. Python package pywin32, `https://pypi.org/project/pywin32/`. Last accessed 14-Aug-2019.
25. Python package pywinauto, `https://pypi.org/project/pywinauto/`. Last accessed 14-Aug-2019.
26. Dinaburg, A., Royal, P., Sharif, M. and Lee, W.: Ether: malware analysis via hardware virtualization extensions. In Proceedings of the 15th ACM conference on Computer and communications security, pp. 51-62. ACM (2008).
27. Kirat, D., Vigna, G. and Kruegel, C.: Barecloud: bare-metal analysis-based evasive malware detection. In 23rd USENIX Security Symposium (USENIX Security 14), pp. 287-301. (2014).
28. Cuckoo, Automated Malware Analysis, `https://cuckoosandbox.org/`. Last accessed 14-Aug-2019.
29. Chakraborty, T., Jajodia, S., Katz, J., Picariello, A., Sperli, G. and Subrahmanian, VS.: FORGE: A Fake Online Repository Generation Engine for Cyber Deception. IEEE Transactions on Dependable and Secure Computing (2019).