2023 53rd Annual IEEE/IFIP International Conference on Dependable Systems and Networks - Supplemental Volume (DSN-S) | 979-8-3503-2545-4/23/$31.00 ©2023 IEEE | DOI: 10.1109/DSN-S58398.2023.00041

Enhancing Honeypot Fidelity with Real-Time User Behavior Emulation

Songsong Liu, Shu Wang, Kun Sun
Center for Secure Information Systems, George Mason University, Fairfax, VA, USA
Email: {sliu23, swang47, ksun3}@gmu.edu

***Abstract*—Honeypot is an effective tool widely adopted in security systems, providing rich information to lure attackers. However, honeypots are not foolproof and can be detected by sophisticated attackers via specific features, e.g., the lack of real user activities. In this paper, we propose HoneyMustard, a real-time application-level user behavior emulation framework to enhance the fidelity of honeypots. HoneyMustard can emulate GUI-based user activities in the honeypots by a remote desktop connection. Because attackers can only observe the remote connection during the emulation, HoneyMustard can conceal the emulator as a normal service so that it achieves real-time user emulation without being detected. The user activities are emulated by reproducing real user operations or converting application manuals, ensuring that attackers can observe logical activities at an application level. We implement a prototype of HoneyMustard and evaluate the decoy effectiveness and overhead. The experimental results show that our solution can effectively improve the fidelity of honeypots with a low overhead.**

***Index Terms*—Honeypot Detection, User Behavior Emulation, Cyber Deception, Computer Vision**

I. Introduction

The honeypot technique has been widely adopted in security systems since it was first introduced by Lance Spitzer [1]. Over the past 20 years, the honeypot has evolved from a simple toolkit into a complex system equipped with "real" data [2]. Honeypots can capture, analyze, and derive the motivation of attacks and serve as an early warning to intrusions.

Meanwhile, honeypot detection techniques (also known as anti-honeypot techniques) have grown among the black hat community for attackers to identify and bypass honeypots using unique system/network features. Since the honeypots lack real user traces, attackers can check the "wear and tear" artifacts caused by user activities [3] or directly monitor user activities [4]. To defeat those anti-honeypot techniques, D2U [5] generates an application usage sequence, but it does not support logical user operations. UBER [6] can generate static artifacts by emulating user activities; however, the emulator running in the system may be detected by the attacker.

In this paper, we propose a real-time application-level user behavior emulation framework, namely HoneyMustard, to enhance the fidelity of honeypot systems. Compared to existing solutions [5], [6], HoneyMustard has two advantages, namely, *generating logical user operations* and *retaining the emulator stealthiness*. It consists of two main stages: user operation collection and computer vision-based emulation. In the first stage, an action dataset is constructed by collecting the user operations from both real user activities and application user manuals. These collected user operations share the same logic as real users, making it hard for attackers to distinguish them based on user traces. In the second stage, HoneyMustard uses computer vision techniques to manipulate the GUI interfaces in the honeypots from a remote server via remote desktop connections. Nowadays many companies and organizations have supported employees to work remotely, so it is normal for attackers to discover a remote desktop connection on a victim's device. Therefore, when we use the remote desktop connection to manipulate the GUI interface in the honeypot, since the emulation engine is running on a remote server, the attackers cannot find the emulator in the honeypot. Thus, HoneyMustard can emulate user activities in real time without being detected.

We implement a prototype of HoneyMustard to demonstrate the deception effectiveness. We record both emulated and real user activities into videos for five common tasks, respectively. Then we conduct a user study that asked 100 online participants to identify the emulated videos. The experimental results show the average success rate of deception is 71.8%, which shows HoneyMustard can effectively deceive attackers with real-time emulated user activities.

In summary, we make the following contributions:

- We propose a real-time application-level user behavior emulation framework to enhance the honeypot fidelity and deceive sophisticated attackers.
- Our design can achieve both authenticity and concealment by emulating GUI-based user behaviors with computer vision techniques through remote desktop connections.
- We implement a prototype of HoneyMustard and evaluate its deception capability via both user study and performance measurement.

II. Threat Model and Assumption

We focus on defeating the honeypot detection methods by checking GUI-based user behaviors. In practice, real users usually interact with computers via GUI interfaces, so the lack of user behaviors has been used as an obvious feature in honeypot detection. In other words, the attackers expect to observe GUI-based user behaviors in the compromised system; otherwise, it could be a honeypot.

We assume the sophisticated attackers (e.g., APT) have gained the root privilege of the compromised computer to better dwell in the compromised system and stealthily collect sensitive information. We assume that attackers can monitor the GUI desktops and all user operations on the GUI interfaces via periodically taking screenshots or direct remote desktop connections (e.g., RDP, VNC). Moreover, with the root privilege, the attackers may scan the entire system to detect the

979-8-3503-2545-4/23/$31.00 ©2023 IEEE
DOI 10.1109/DSN-S58398.2023.00041

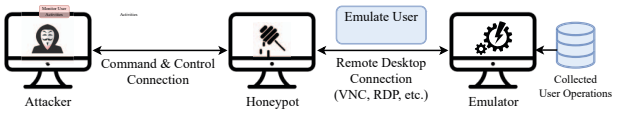

Figure 1. The Overview of HoneyMustard.

honeypot emulators in the compromised systems. By observing user activities and analyzing the activity logic, attackers aim to decide if the compromised system is a honeypot. To maintain stealthiness, attackers would avoid direct interaction with the real users or interference with the user operations.

III. System Design

A. Overview of HoneyMustard

We propose the HoneyMustard framework to enhance honeypot fidelity with the following three design principles.

- *Stealthiness.* The emulator trace is invisible in the honeypot.
- *Authenticity.* The emulated user activities have the same logic as those of real users.
- *Real Time.* It can emulate user activities in real time.

Figure 1 shows that HoneyMustard consists of two core components: collected user operations and the emulator. Both of them are deployed in an independent device other than the honeypot. The emulator connects to the honeypot via a remote desktop connection and leverages computer vision techniques to redo the collected user operations on the GUI interfaces. Attackers can only notice the existence of a remote desktop connection in the compromised system, not the emulator. With the usage of cloud services, more companies and organizations deploy their working platforms in the cloud and allow employees to access virtual desktops via remote connections; hence, attackers cannot simply take the remote desktop connection as an indicator of an emulator. Therefore, the remote desktop connection enhances the system's stealthiness. We collect the GUI-related user operations from real users to ensure the system's authenticity. To achieve real-time performance, HoneyMustard emulates different user activities periodically.

B. User Behavior Emulation

Figure 2 shows the workflow of HoneyMustard, which contains two stages: *user operation collection* and *computer vision-based emulation*.

1) User Operations Collection: Intuitively, it is convenient to collect a large number of user operations from the daily activities of real users. However, to protect users' privacy, the private information contained in daily activities should be eliminated. In this stage, we construct the user operation dataset with two types of data sources, namely, real user activities and user manuals. To collect from real user activities, we invite real users to complete assigned tasks without using their personal information (decoy information is allowed). To collect from user manuals, we download the user manuals for completing specific application tasks and extract the logical user operations. Since the manual contributors (e.g., software vendors) have sanitized the content, no private information remains in the user manuals.

User Activity Processing. We invite users to complete specific basic tasks on common applications, each task containing multiple essential operations. Meanwhile, we leverage a GUI recorder to record each step conducted by users over the GUI interface. For each task, we convert user operations to a manipulation log, which contains the actions (e.g., mouse click, input) and the GUI elements (e.g., button, check box).

User Manual Processing. First, we leverage a user manual normalizer to convert various manual formats to the normalized Markdown format, which can facilitate the parser to analyze the manual content [7]. Similar to the manipulation logs, the Markdown-formatted user manuals preserve the actions and GUI elements. Second, for each task, an action extractor generates an action list from descriptive text to preserve the operation logic. The action list contains action keywords (e.g., click, select, type) and the target objects (e.g., GUI elements, text). Finally, we build the dataset for user activity emulation by aggregating all the action lists.

2) Computer Vision-based Emulation: During user activity emulation, HoneyMustard picks up a desired task (i.e., action list) from the dataset and emulates actions over the honeypot. To emulate a logical application usage sequence, we also consider the work hours and break times. For each task, HoneyMustard converts the action list to an enriched action code, which makes the actions executable for the emulator. The action code converter also introduces more action details that are not provided by user manuals. To enrich the action code, the converter completes the following tasks:

Action Keyword Conversion. The action keywords in the action list should be converted to emulator-readable keywords. Because some descriptive action keywords are not available in the emulator and should be represented as specialized action keywords, this step ensures the emulator can read and execute the action keywords in the code.

Waiting Time Addition. HoneyMustard adds extra waiting time slots among consecutive actions because real users always pause for a while between two consecutive operations. To increase system authenticity, the waiting time can reflect the user's thinking time or encountered disturbance. Also, there is a processing time between two consecutive actions so that the software can switch status or interface to receive the following actions. Therefore, we add the waiting time slots with random numbers, which are decided by the emulation environment performance and software response speed.

Image Path Addition. The image paths of pre-collected GUI elements should be added to the code because the emulator needs to identify the corresponding GUI elements on screen via computer vision techniques. For the target objects in the action list, the converter can easily access the GUI element images and add their paths to the code. However, some icon-based GUI elements do not present descriptive text on the screen. In this case, we collect these GUI elements as pre-knowledge data and link them to the object names.

Action Parameter Addition. Some actions need additional parameters, e.g., asking users to fill in usernames or emails. For the action lists collected from real user activities, we have

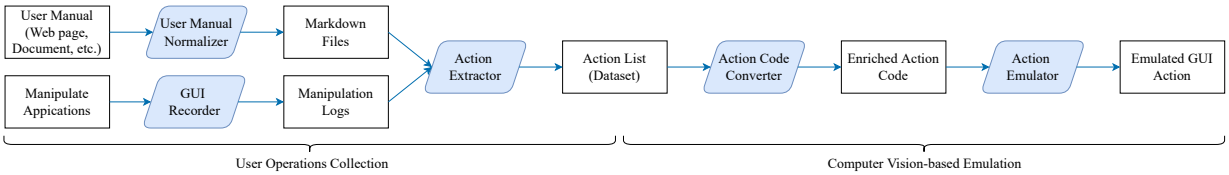

Figure 2. The Workflow of HoneyMustard.

already provided decoy information. For the action lists collected from user manuals, we use default or decoy information.

With the enriched action code, HoneyMustard can emulate the operations of input peripherals over the software/system interfaces. The action emulator first leverages computer vision techniques to locate the target objects (i.e., images, text) on the screen. For images, the emulator locates the corresponding GUI elements by template matching. For text, the emulator locates the matching text by optical character recognition (OCR). After obtaining the actions and target objects, the emulator can sequentially execute the action code on the honeypot via a remote desktop connection.

IV. Implementation

A prototype of HoneyMustard is built on a Linux host. We use a honeypot in Windows 10 with *UltraVNC server* 1.3.2a, connecting to the emulator via a VNC connection.

A. User Operations Collection

1) GUI recorder: We use *WinAppDriver* [8] as the GUI recorder, which tracks both keyboard and mouse interactions over application interfaces, i.e., GUI actions. The record logs include the GUI actions and the involved GUI elements.

2) Manual Normalizer: We use *pandoc* 2.5 [9] to convert user manuals in various formats (e.g., HTML, DOCX, EPUB) to the Markdown format. For the manuals in other formats, we can use the corresponding Markdown format converters.

3) Action Extractor: The action extractor, which is developed in Python 3, obtains the action lists from the descriptive instructions in manipulation logs and Markdown-formatted manuals. The basic idea of the action extractor is to identify the pre-defined keywords in the instructions. However, not each matched keyword represents an action. A matched keyword can indicate an action only if it is a verb, e.g., the keyword "*Input*" can appear as a noun or a verb in a manual while only the verb-form "*Input*" can indicate an action. Thus, we identify the word classes by lexical category analysis. With the *NLTK* [10], the action extractor splits the instructions into word tokens and then attaches the class tag to each word.

With the action keywords, the action extractor will obtain the involved target objects, typically a noun or a short phrase. The action extractor identifies the target objects by conducting a forward search of nouns from the identified action keyword to the next one or to the end of the current sentence. Note that some identified action keywords may not even have target objects. Besides, the target objects can be located via symbols, e.g., quotation marks and asterisks. Our action extractor can also identify these target objects, which usually appear in the descriptive sentences of system configuration manuals.

B. Computer Vision-based Emulation

1) Action Code Converter: The action converter transforms the action list into the Robot Framework code, which has easy-to-use tabular test data syntax and utilizes the keyword-driven testing approach. The Robot Framework code can ensure the action converter is simple and effective. The action converter translates the action records into the test cases in sequence, takes action keywords as the code keywords, and transforms the target object information as the code parameters.

2) Action Emulator: The action emulator consists of *Robot Framework* 4.1 [11] and *Sikuli* 2.0.5 [12]. Robot Framework is a Python-based, extensible, and keyword-driven automation framework, with various generic and custom libraries. We use the Sikuli library of Robot Framework to manipulate the GUI elements on the screen. Sikuli uses image recognition algorithms powered by OpenCV to identify GUI elements and uses OCR to identify text.

V. Evaluation

HoneyMustard is running on an Ubuntu 20.04 (kernel version 5.4.0) device with Intel Xeon E5-2620 CPU @ 2.40GHz and 16 GB RAM. The honeypot is a Windows 10 device with an Intel Core i7-6500U CPU @ 25.0GHz and 8 GB memory.

A. Deception Effectiveness

1) Study design: To evaluate the deception effectiveness of HoneyMustard, we conduct a user study to check if humans can distinguish the emulated user activities from the real ones. To simulate the real-time environments, we record the user activities in videos. After watching the videos, the participants need to decide if the activities are conducted by real humans. Considering that real attackers are more sensitive than normal participants, we also provide videos of real user activities as a baseline, which can assist participants to notice the differences in activities. We collect the subjective and direct feelings from participants with an online survey via the crowdsourcing platform - Amazon Mechanical Turk [13]. We totally received 100 unique responses from anonymous users. We check the response patterns manually to confirm there is no malicious filling on the questionnaire.

Materials. We select five common tasks for the user study, including OpenVPN installation (**T1**), LibreOffice Marco editing (**T2**), Adobe Reader trust configuration (**T3**), windows system environment setting (**T4**), and windows network discovery configuration (**T5**). The involved manuals are downloaded from the knowledge platform - Dummies [14] and the technique blog - Twilio Blog [15]. On average, it takes 10 steps to complete the tasks. For each task, a video is generated for

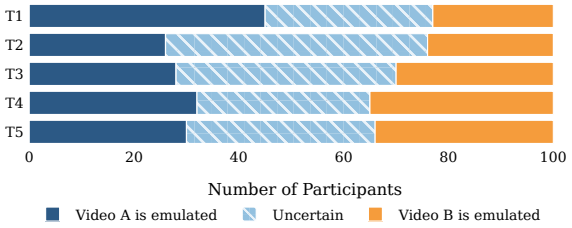

Figure 3. Results of the Activities Identification Survey.

emulated and real user activities, respectively. To generate the videos of emulated user activities, HoneyMustard converts the user manuals to emulated user operations on the honeypot. To generate the videos of real user activities, we manually conduct the same tasks by ourselves and record the videos. Since all the data is collected from the authors and public user manuals, IRB approval is not required in our experiments.

Procedure. Each participant needs to watch ten videos of five tasks and then answer five questions to indicate which videos may contain emulated user activities. Each question provides three choices: video *A* (real user activities), video *B* (emulated user activities), and uncertain. We set the maximum completion time of the questionnaire as 30 minutes.

2) Results: Figure 3 shows participants cannot accurately identify the emulated user activities. Since only video *B* in each task contains emulated user activities, the average recognition accuracy is 29.2%. On average, 32.2% of participants select the wrong videos, and 39.6% of participants cannot make their decision. The deception success rate is 71.8%, showing HoneyMustard's effectiveness in deceiving attackers.

We find the deception success rate varies among different tasks. The success rate is higher in software manipulation activities (77% for **T1**, 76% for **T2**, 70% for **T3**) than system configuration activities (65% for **T4**, 66% for **T5**). That is because software manipulation activities are relatively more complicated and contain more mouse/keyboard operations. From the participants' comments, they mainly observe the cursor movement and cannot find out any logical issues in these videos. Also, participants tend to select videos with more mechanical and faster cursor movement. These criteria are critical for improving our future design.

In practice, the VNC connection can introduce network delay. Because the action emulator manipulates the GUI interface projected to the local, the network delay only influences the response time. Real users confront the same issue when using a VNC connection, so the attackers cannot leverage the network delay to identify the emulated user activities.

B. System Overhead

We measure the average memory consumption and CPU usage for processing the user manuals of these five tasks. The action extractor takes about 190 MB of memory and 12% CPU usage. The action code converter takes about 35 MB of memory and less than 1% CPU usage. Because we do not modify the third-party tools, their system overhead can be found in the related official documents. The experiment results show that HoneyMustard can be readily deployed in the real world without excessive performance overhead.

VI. Discussion And Future Work

Our current solution inserts a random waiting time slot between operations to emulate a real user. However, a real user usually has a specific operation pattern in the waiting time slots. This pattern is decided by the activity type, personal character, and job type. From a long-term observation, sophisticated attackers can usually extract an access pattern of waiting slots from real users, while the randomly generated waiting time slot cannot provide a fixed pattern. In this case, attackers may detect the emulated user activities. In future work, we plan to create various user profiles to increase the success rate of deception. Based on the various profiles, HoneyMustard can insert specific patterned waiting time slots to impersonate different types of real users. Besides, the cursor movement speed should also adapt to different user profiles.

VII. Related Work

Honeypot Detection. Honeypot detection methods focus on the artifacts in two major categories: *network-related fingerprinting* and *system-related fingerprinting* [16], [17]. Network-related fingerprinting is to detect the discrepancy in network activities [18] and network latency [19]. System-related fingerprinting is to detect the setup differences in operating systems or applications, e.g., files, operating system flags, running processes, volatile user information, and installed programs [20]–[22]. Besides, attackers can detect user activities in systems, e.g., the "wear and tear" artifacts [3] or mouse clicks [4].

User Emulation. User emulation is an effective method to counter honeypot detection. To defeat network-related detection, defenders can emulate the network traffic [23], [24] or reproduce real traffic [25]–[27], To defeat system-related detection, defenders can generate usage artifacts [6] or emulate application usage sequence [5]. However, none of them propose a real-time application-level emulation solution.

VIII. Conclusion

This paper presents HoneyMustard, an application-level real-time user behavior emulation framework to enhance the fidelity of honeypot systems. We construct an action dataset by collecting logical user operation sequences of various applications from real user activities and user manuals. Then, we leverage computer vision techniques to emulate user activities in the honeypot. With the VNC connection, HoneyMustard remotely manipulates the GUI interface of the honeypot without deploying any extra suspicious agent in the honeypot. Meanwhile, the emulated user activities can be observed directly by attackers. We implement a prototype of HoneyMustard and conduct an online survey with 100 participants. The experimental results show that HoneyMustard can effectively deceive attackers with real-time emulated user activities.

IX. Acknowledgments

This work was partially supported by ONR grant N00014-18-2893.

REFERENCES

[1] L. Spitzner, "Honeypots: Catching the insider threat," in *19th Annual Computer Security Applications Conference, 2003. Proceedings.* IEEE, 2003, pp. 170–179.

[2] C. K. Ng, L. Pan, and Y. Xiang, *Honeypot frameworks and their applications: a new framework*. Springer, 2018.

[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, pp. 1009–1024.

[4] S. O. Vashisht and A. Singh, "Turing test in reverse: new sandbox-evasion techniques seek human interaction," 2014.

[5] S. Oesch, R. A. Bridges, M. Verma, B. Weber, and O. Diallo, "D2U: Data driven user emulation for the enhancement of cyber testing, training, and data set generation," in *Cyber Security Experimentation and Test Workshop*, 2021, pp. 17–26.

[6] S. Liu, P. Feng, S. Wang, K. Sun, and J. Cao, "Enhancing malware analysis sandboxes with emulated user behavior," *Computers & Security*, p. 102613, 2022.

[7] J. Gruber, "Daring fireball: Markdown," *Récupéré le*, vol. 3, no. 04, 2004. [Online]. Available: https://daringfireball.net/

[8] Microsoft, "Windows Application Driver," https://github.com/microsoft/WinAppDriver, accessed in Feb 2023.

[9] J. MacFarlane, "Pandoc - a universal document converter," https://pandoc.org/releases.html, accessed in May 2022.

[10] NLTK Team, "NLTK," https://www.nltk.org/, accessed in Feb 2023.

[11] Robot Framework Foundation, "Robot Framework," https://robotframework.org/, accessed in Feb 2023.

[12] R. Hocke, "SikuliX," https://sikulix.github.io/, accessed in Feb 2023.

[13] Amazon, "Amazon Mechanical Turk," https://www.mturk.com/, accessed in Feb 2023.

[14] Dummies, "dummies - Learning Made Easy," https://www.dummies.com/, accessed in Feb 2023.

[15] Twilio Blog, https://www.twilio.com/blog, accessed in Feb 2023.

[16] J. Uitto, S. Rauti, S. Laurén, and V. Leppänen, "A survey on anti-honeypot and anti-introspection methods," in *World Conference on Information Systems and Technologies*. Springer, 2017, pp. 125–134.

[17] S. Morishita, T. Hoizumi, W. Ueno, R. Tanabe, C. Gañán, M. J. van Eeten, K. Yoshioka, and T. Matsumoto, "Detect me if you… oh wait. an internet-wide view of self-revealing honeypots," in *2019 IFIP/IEEE Symposium on Integrated Network and Service Management (IM)*. IEEE, 2019, pp. 134–143.

[18] J. Rrushi, "Honeypot evader: Activity-guided propagation versus counter-evasion via decoy os activity," in *Proceedings of the 14th IEEE International Conference on Malicious and Unwanted Software*, 2019.

[19] X. Fu, W. Yu, D. Cheng, X. Tan, K. Streff, and S. Graham, "On recognizing virtual honeypots and countermeasures," in *2006 2nd IEEE International Symposium on Dependable, Autonomic and Secure Computing*. IEEE, 2006, pp. 211–218.

[20] T. Holz and F. Raynal, "Detecting honeypots and other suspicious environments," in *Proceedings from the Sixth Annual IEEE SMC Information Assurance Workshop*. IEEE, 2005, pp. 29–36.

[21] A. Bulazel and B. Yener, "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*, 2017, pp. 1–21.

[22] A. Afianian, S. Niksefat, B. Sadeghiyan, and D. Baptiste, "Malware dynamic analysis evasion techniques: A survey," *ACM Computing Surveys (CSUR)*, vol. 52, no. 6, pp. 1–28, 2019.

[23] L. M. Rossey, R. K. Cunningham, D. J. Fried, J. C. Rabek, R. P. Lippmann, J. W. Haines, and M. A. Zissman, "Lariat: Lincoln adaptable real-time information assurance testbed," in *Proceedings, ieee aerospace conference*, vol. 6. IEEE, 2002, pp. 6–6.

[24] T. M. Braje, "Advanced tools for cyber ranges," MIT Lincoln Laboratory Lexington United States, Tech. Rep., 2016.

[25] P. Megyesi, G. Szabó, and S. Molnár, "User behavior based traffic emulator: A framework for generating test data for dpi tools," *Computer Networks*, vol. 92, pp. 41–54, 2015.

[26] S. Molnár, P. Megyesi, and G. Szabó, "Multi-functional traffic generation framework based on accurate user behavior emulation," in *2013 IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*. IEEE, 2013, pp. 13–14.

[27] P. Dutta, G. Ryan, A. Zieba, and S. Stolfo, "Simulated user bots: Real time testing of insider threat detection systems," in *2018 IEEE Security and Privacy Workshops (SPW)*. IEEE, 2018, pp. 228–236.