Fraus: Launching Cost-efficient and Scalable Mobile Click Fraud Has Never Been So Easy

Elliott Wen
The University of Auckland
jq.elliott.wen@gmail.com

Jiannong Cao, Jiaxing Shen, Xuefeng Liu
The Hong Kong Polytechnic University
csjcao, jxshen@comp.polyu.edu.hk

***Abstract*—Mobile click fraud is a type of attack where an adversary deceptively generates click events on mobile applications in pursuit of revenue. Conventionally, the attack is carried out by automating a massive number of physical devices. However, purchasing the devices incur substantial costs. A cheaper alternative to the physical devices is emulators. However, existing emulators are inefficient and vastly blocked due to their immense resource demand and defective device signatures. In this paper, we propose Fraus[1], a cost-efficient and scalable approach to conduct large-scale click fraud using device emulators. Fraus maintains a low resource profile by circumventing graphics emulation and applying lazy-loading techniques on system components. Besides, Fraus provides a seemingly authentic device signature and disguises itself as a legitimate device by fully emulating the missing hardware components including WiFi interfaces and cellular modems. To facilitate the management of numerous emulator instances, Fraus also offers a distributed management system, which is scalable and fault-tolerant. We evaluate the performance of Fraus by mocking attacks against the top 300 applications from the Google Play store. The results demonstrate that Fraus has high system stability and application compatibility. It also significantly reduces CPU usage and memory footprint up to 90% and 60% respectively compared with the existing emulators.**

I. Introduction

Nowadays, Mobile Click Fraud Attack (MCFA) has become a frequent topic of cyber security experts [?]. In such an attack, malicious individuals repeatedly generate click events on a mobile application with the intention of increasing revenues or personal influence. The common examples include boosting product ratings or increasing the ‘like’ number in social media pages. According to [?], the attack is causing a substantial damage of $16.7 billion on mobile application economy in 2017.

To launch a MCFA, a simple approach referred to as Click Farms [?] is widely adopted. Figure 1 demonstrates the general set-up of such a farm where thousands of mobile devices are deployed and automated by computers to generate click events. Since the click actions are originated from the physical devices, they all appear to be triggered by real users without sophisticated examinations [?]. It makes this approach fairly effective for most applications.

However, despite the effectiveness, setting up such a farm incurs a substantial cost. The major expenses is for purchasing the massive number of devices. Meanwhile, the subsequent deployment and maintenance tasks also involve huge labor costs. These drawbacks naturally inspire us to investigate the potential of replacing the physical devices with device emulators. Device emulators are in essence virtual machines that are mainly designed for developers to swiftly prototype mobile applications without using hardware devices. Compared with physical devices, emulators are significantly more cost-effective because they can be massively deployed in cloud platforms or local servers in a cheap price. For instance, the average price for a low-end smartphone with one-year lifespan is approximately 215 USD [?], while the money can enable us to host 6 emulator instances without optimization in the cloud for a same timespan [?]. Besides, deploying and maintaining emulators is less labor intensive owning to existing automated management systems [?].

Though the idea of launching click fraud with emulators seems quite straightforward, practical implementation entails substantial challenges. First, existing device emulators are CPU-intensive and memory-intensive. It likely results in poor performance and application failure due to resource constraints. Moreover, the high resource demand also decreases the number of emulator instances that can be run concurrently, which limits the power of the attack. Second, the existing emulators are easily blocked by various mobile applications due to their defective device profiles. A profile serves as a unique device signature, usually consisting of properties such as WiFi MAC addresses or phone numbers that are hardware-specific. Since the corresponding hardware components are not emulated, the property values remain empty. Therefore, by scrutinizing the defective device profile, an application can easily identify and block the emulator. Finally, it is challenging to deploy and manage numerous emulator instances in a large-scale attack, because existing emulator management systems are centralized and they fail to cope with the scale-out and fault tolerance issues.

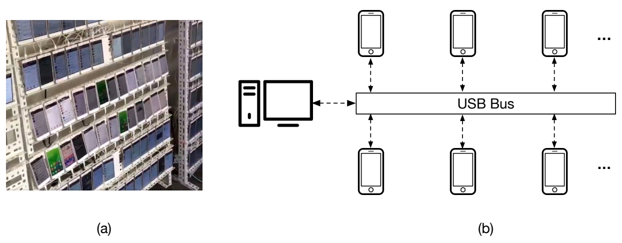

Fig. 1: (a) A photo inside a real-life click farm [?]. (b) The general set up of a click farm, where phones are powered up and connected to a central server via USB connections.

[1]In Roman mythology, Fraus was the goddess or personification of treachery and fraud.

In this paper, we propose Fraus, a novel approach that enables individuals to launch MCFA using device emulators. Specifically, Fraus significantly reduces CPU usage by circumventing the compute-intensive graphics emulation. It also maintains a small memory footprint by applying the lazy-loading and deduplication techniques on system components. To bypass device-profile scrutiny, Fraus implements two virtual hardware components including cellar modems and WiFi interfaces. They enable each emulator instance to generate a seemingly authentic device profile. Meanwhile, Fraus offers a distributed emulator cluster management system, which is scalable, fault-tolerant and easy-to-use. Its API abstraction features a logically centralized view, which allows an user to easily program and launch an attack task without concerning the details of the underlying distributed protocols.

We consolidate the above techniques and implement the prototype of Fraus on top of Android 6.0. We conduct comprehensive experiments to evaluate Fraus against the top 300 applications from the Google Play store. Experimental results show that Fraus has high application compatibility and system stability because 96% of the selected applications run successfully, while the number is only 59% for existing emulators. Fraus also reduces CPU usage and memory footprint up to 90% and 60% respectively compared with the existing emulators. The small resource demand enables fitting 125 instances in a low-end server equipped with 32GB RAM, reflecting the high cost efficiency.

To the best of our knowledge, Fraus is the first system that is designed to launch a cost-efficient and scalable MCFA via devices emulators. By designing Fraus, we aim to raise public concerns about the simplicity of committing click fraud and to suggest countermeasures to mitigate such risks.

The rest of this paper is organized as follows. Section. II investigates the problems of launching MCFA using existing emulators. Section. **??** presents the approach to disable the compute-intensive graphics emulation. Section. **??** describes optimization techniques that reduce the memory footprint. Section. **??** illustrates the methodology to bypass device profile scrutiny. Section. **??** elaborates the architecture for the distributed emulator instance management system. Section. **??** reports the evaluation results. Section. **??** discusses the countermeasures to combat Fraus. Section. **??** surveys the related work and Section. **??** concludes this paper.

II. Background and Motivation

Little research has been conducted to investigate the usability of emulators on MCFA. In this section, we attempt to evaluate the usability and identify the potential problems pending to be addressed via preliminary experiments. We first select 4 target mobile applications from two genres that are frequent victims of MCFA, including social media and online advertising. The selected social media applications consist of *Facebook* and *Wechat*. The selected online advertising applications, which reward users for every view of a commercial video clip, include *Daily Cash* and *Lucky Cash*. We run these applications on three widely-used options including *Android Official Emulator* [**?**], *BlueStacks* [**?**] and *GenyMotion* [**?**]. These emulators are allocated with an one-core 2.2 GHz virtual CPU, 512 MB RAM, and 12 MB video RAM with GPU acceleration disabled. This resource configuration is chosen to be close to the specification of a low-end virtual machine provided by most cloud platforms [**?**]. In our mocking attack, we identify the following difficulties.

TABLE I: System Wide CPU Usage When Running Selected Applications.

CPU Usage	Offical	BlueStack	GenyMotion
Wechat	43%	47%	45%
Facebook	57%	54%	59%
Daily Cash	79%	76%	78%
Lucky Cash	87%	83%	84%

TABLE II: CPU utilization of each running process in an emulator.

CPU Usage	Surfaceflinger	Mediaserver	Systemservice	App. itself	Others
Wechat	25%	0%	3%	11%	4%
Facebook	37%	2%	4%	13%	1%
Daily Cash	11%	49%	7%	7%	5%
Lucky Cash	23%	43%	5%	10%	6%

Performance Issues. The first difficulty arises from the performance issues of emulators. We observe that the emulators tend to be CPU-intensive. To demonstrate our observation, we measure the average system-wide CPU usage when the emulators are automated to perform predefined tasks such as clicking a like button or viewing a video clip in the 4 applications. The results are shown in Table I. It can be seen that all the applications are incurring high CPU workload regardless of the types of emulators. Specifically, the social media applications consume almost half of the CPU processing power. The advertising applications appear to be more compute-intensive and nearly saturate the CPU.

To pinpoint the root cause of high system-wide CPU utilization, we further demonstrate the usage of each running process in an emulator in Table II. Note that we only list the CPU statistics for the Android official emulator because different emulators yield similar CPU performance. One essential finding from the table is that although the overall CPU usage is high, only a small proportion is attributed to the application itself. Two system processes *Surfaceflinger* and *Mediaserver* are the main culprits of the excessive CPU usage. Surfaceflinger is an Android system service, in charge of compositing graphical user interfaces (GUI) of applications into a single buffer, which is finally displayed by a screen. Mediaserver is another important system service that entitles an application to play various types of video files. The two system services involve a great amount of graphical computation, which can be offloaded to GPUs in physical devices. However, GPUs are not available in an emulator and the graphical computation has to be conducted by CPUs inefficiently, which justifies the high CPU workload of the Surfaceflinger and Mediaserver.

Another performance concern is the excessive memory usage. The minimum memory requirement of BlueStack and GenyMotion is 2 GB. It means that the only way to run them in the virtual machine with 512 MB RAM is to enable swapping. However, swapping may negatively impact the system performance due to the potential thrashing behavior. The Android official emulator somehow possesses a smaller footprint, which is approximately 500 MBs right after the system starts up. Nevertheless, the free RAM is extremely limited and swapping is still required in order to run any other applications.

Defective Device Profile. We meet the second obstacle when running applications that enforce device profile scrutiny. A device profile serves as a unique device identification. It usually consists of several hardware-specific properties. Commonly used properties include:

1) WiFi Media Access Control (MAC) Address, a unique identifier assigned to WiFi interfaces.
2) The cell phone number of the device.
3) International Mobile Subscriber Identity (IMSI), a unique number to identify the user of a cellular network.
4) International Mobile Equipment Identity (IMEI), a unique 15-digit serial number given to every mobile phone's cellular modem.

All the properties are retrieved from hardware components including cellular modems and WiFi interfaces. Since they are not emulated, the above property values remain empty in an emulator. The artifact then allows application providers to easily identify and block the emulators. To make matters worse, the authenticity of the properties, especially the phone number, can be easily verified by the applications. For instance, many applications send the number a SMS containing verification codes to ensure the number is genuine. It makes randomly generating the property values an infeasible approach to bypass the scrutiny.

III. CONCLUSION

In this paper, we raise public concerns of the simplicity of launching a large-scale MCFA by presenting Fraus. Fraus has a tiny CPU and memory resource demand. Besides, Fraus can generate a seemingly authentic device profile and disguise itself as a legitimate device. To facilitate a large-scale attack, Fraus also provides a distributed emulator cluster management system, which is scalable and fault-tolerant. We evaluate the performance of Fraus and demonstrate that Fraus has high application compatibility and cost efficiency. Finally, we also propose countermeasures leveraging Fraus's platform-specific characteristics to combat such an attack.