International Journal of Science and Research (IJSR)
ISSN: 2319-7064
SJIF (2020): 7.803

Formulating Operating System Anonymization by Monolithic Keyring Subsystem Fabrication

Ujas Dhami[1*], Nisarg Shah[2]

[1, 2]Department of Computer Engineering, Silver Oak University, Ahmedabad-380061, India
[*]Corresponding Author E-mail: ujasdhami[at]gmail.com

Abstract: ***The monolithic kernel is a widely used kernel when building ARM architectures and Linux distributions. With all instructions being in the kernel space, it is insecure, and can easily perform cross-transactions of sensitive information when under a memory leak, thus, it is vital to lock certain sections of the kernel to confine it from exposing personal information to risk. Investigation: This paper demonstrates a Volatile Operating System formulated by the authors with the use of Keyring Subsystems for fragmenting the kernel space into multiple domes using security keys. Method: A kernel is configured and built using security modules, and deployed with anonymization tools to a bootable USB, to check if memory leaks leave traces of cache into the device drivers or the disk. These anonymization tools help to clear memory. Principle Result: Less to no information was transmitted to the device memory, leaving everything as is. The anonymization tools cleared off all the events and cache before shutting down the system, making it challenging for forensics to acknowledge the use of a volatile OS into the device from a USB stick.***

Keywords: Operating System, OS Development, Monolithic Kernel, Kernel Security and Anonymity, Anonymization

1.Introduction

Linus Torvalds discovered the Linux OS in 1991, which grew to develop the UNIX OS further. He proposed enhancements yet was dismissed by UNIX architects. Subsequently, he considered sending off an OS plan to alter its clients. These days, Linux is the quickest developing OS. It is utilized from telephones to supercomputers by practically all significant equipment gadgets.

Linux is an open-source[1] working framework like other working frameworks, like Microsoft Windows, Apple Mac OS, iOS, Google android, etc. An OS is software that sanctions the correspondence between PC equipment and programming. It passes on contribution to handle the processor and carries results to the hardware devices to display it. This is the vital capacity of a working framework. Even though it performs numerous other significant assignments, threats have always been there to acknowledge.

1.1 Significance of Anonymity

Regarding examining the significance of Anonymity, individuals will make statements like, "On the off chance that you're not doing anything wrong, you have nothing to fear.” There's a suspicion that the prominent individuals participating in evil exercises need their personalities to be protected. In reality, there are many valid justifications to think often about secrecy, particularly on the web. [1] Their not coping up with the anonymity of their character when necessary can create significant issues, like:

- Personality Protection: Sometimes, you don't need anybody to know who you are, regardless of whether you're not engaged with anything unlawful or problematic. There's a degree of social security that accompanies obscurity. Also, that can be genuinely significant for thoughtful people in web-based networks.

- Individual Harassment: Online namelessness likewise assumes a significant part in the opportunity of articulation. A most astonishing aspect of the web is that it can give voices to effectively being quieted, people. This permits them to talk unafraid of repercussions. Online Harassment incorporates Doxing, Swatting, and revenge pornography.

- Sensitive Issues: There's another significant classification of individuals who benefit from namelessness: the people who need more data on a given theme yet don't have any desire to be discovered searching out that data. A great many people fall into this gathering without acknowledging it.

1.2 Research Overview

The NIJAS OS is a Linux-based Operating System made explicitly for anonymization purposes. This OS consists of the calibre to provide absolute anonymity to whoever uses it. It is a live OS, meaning that it does not save any data/logs/history if not installed and run live. It has no interference in the threads leading to the Hard Disk Drive (HDD). Thus, no data can ever be exhilarated from the session done with the live OS[2], as everything is operated from the USB stick.

However, the USB Stick, too, is close to being untraceable for the sessions done in the OS because the OS contains a fully-encrypted virtual disk with a 3-layer security architecture [2]. The OS consists of the LSM encryption suite, and dynamic memory[3] relocates to anonymize the session history through randomized memory locations.

[1]Open source software refers to programs which are distributed for anyone to access, modify and redistribute

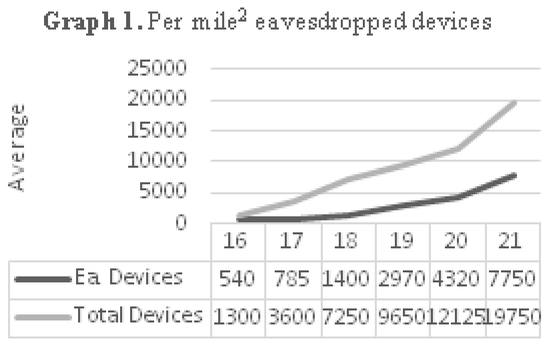
[2]https://en. wikipedia.org/wiki/Live_CD

[3]Dynamic memory allocation refers to managing system memory at runtime.

Furthermore, the Operating System itself destroys all evidence upon shutdown, including RAM clearance,. bash history clearance, deleting logs and terminal history, and demolishing everything which ever used the device's resources for functioning.

1.3 Scaling

NIJAS OS can be used to perform covert communications, research on anonymity and stealth, for maintaining privacy, and for escaping espionage from other organizations, since individual users’ privacy is on stake nowadays, as depicted in Graph 1.



	16	17	18	19	20	21
Ea Devices	540	785	1400	2970	4320	7750
Total Devices	1300	3600	7250	9650	12125	19750

Moreover, the OS can be used exclusively by Intelligence Agencies, Law Enforcement, Reporters, Scientists, and Researchers.

2.Literature Review [3] [4] [5]

Title: Monitors: An Operating System Structuring Concept
This paper creates Brinch Hansen's idea of a screen to organize a working framework. It presents a type of synchronization, depicts a potential strategy for execution as far as semaphores, and gives an appropriate evidence rule. Illustrative models incorporate a solitary asset scheduler, limited support, a morning timer, a cradle pool, a circle head streamlining agent, and an adaptation of the issue of peruses and writers.

An essential point of a working framework is to divide a PC establishment between many projects setting unpredictable expectations upon its resources. An essential undertaking of its originator is in this manner to construct resource allocation (or scheduling) calculations for resources of different sorts (entire store, drum store, magnetic tape overseers, consoles, etc.). To work on his errand, he should attempt to construct separate schedules for each resource day.

Title: The JX Operating System

This paper depicts the engineering and execution of the JX working framework. JX is a working framework written in Java and a runtime framework for Java applications. Our work exhibits that it is feasible to construct a comprehensive working framework in Java, accomplish a decent presentation, and still advantage from the advanced programming innovation of this article arranged, type-safe language.

They clarify how a working framework can be organized that is never again expanded on MMU assurance yet on type security. JX depends on a bit of microkernel, which is answerable for framework introduction, CPU setting exchanging, and low-level assurance space on the board. The Java code is coordinated in parts stacked into areas, checked, and meant local code. Rooms can be disengaged from one another.

Title: Corey: An Operating System for Many Cores

Multiprocessor application execution can be restricted by the working framework when the application utilizes the operational framework often. The working framework administrations use information structures shared and altered by various handling centers. If the application needn't bother with the sharing, then, at that point, the working framework will turn into an excessive bottleneck to the application's performance.

This paper contends that applications should control sharing: the piece should orchestrate every information structure. Hence, just a solitary processor needs to update it, except if coordinated in any case by the application. This plan standard directs this paper. It proposes three working framework deliberations (address ranges, part centers, and offers) that permit applications to control between center sharing and exploit the possible overflow of centers by devoting centers to explicit working framework functions.

Measurements of microbenchmarks on the Corey model operational framework, which epitomizes the new reflections, show how to command over-sharing can further develop execution. Application benchmarks, utilizing Map Reduce and a Web server, show that the upgrades can be massive for by and large commission: Map Reduce on Corey performs 25% quicker than on Linux when utilizing 16 centers. Equipment occasion counters affirm that these upgrades are expected to keep away from costly activities on multicore machines. The outcomes above ought to be considered a case for the rule that applications should control sharing rather than an indisputable "confirmation. " Corey needs many highlights of product working frameworks, for example, Linux, which impacts exploratory outcomes both decidedly and adversely.

3.Methodology

Developing an Operating System required a collective knowledge on Kernels, low-level instructions and the libraries required to build up the roots. Thus, we started researching about it and about how Linux Kernels are configured and compiled.

3.1 Formulation Prerequisites

As the NIJAS Operating System was built based on the libraries of the Linux Mint Operating System, hence, at the initial phase, we downloaded the Linux Kernel. After downloading the Kernel, we studied the different essential components of Linux. That is, we learned about Linux Keyrings and instructions. Keyrings allow you to group all the passwords and keep them in one place. After reviewing different Keyrings, we then jumped towards the integration and the compilation of other Linux components.

Linux components included the Kernel, System Libraries, System Utility, Basic Input Output System (BIOS), Master Boot Record (MBR), etcetera. The Kernel is the core part of the Linux Operating System and is responsible for carrying out all the different activities in the operating system. System libraries are special functions that access Kernel's features. System Utility is a program that carries out individual tasks. BIOS Menu searches and executes the bootloader from the MBR. It acts as a bridge between the hardware and the software.

At the same time, the Master Boot Recorder is located in the first sector of the boot disk. Init Processes are the processes that halt Single/Multi-User mode. We integrated everything into the Kernel and performed couple dependency checks.

3.2 Compartmentalizing Keyrings

After Integrating/Compiling the Linux Components, one needs to secure its inbound keyrings[4]. These include securing the authentication keys, encryption keys, and other data present in the Kernel. Keyrings have attributes like Serial Number, Type, Description, and the Payload, Access Rights, the Expiration Time, and the reference count responsible for various Linux operations. Compartmentalizing them will isolate them from exposing memory leaks to the device.

Moreover, we took the libraries and the themes of the Linux Mint operating system, including its package manager. All these things are used to give a better User Interface to the user. We specifically took two package managers, namely apt and snap. Snap is a software package and deployment system that uses self-contained packages called snaps to deliver software to users, while APT mainly obtains packets from a distribution's official repositories. Snap enables developers to access their apps via the Snap Store directly.

We, then, flashed the KDE Plasma environment[5] as a Window Manager. The desktop environment gives a more compatible user facility to navigate between different parts of the operating system [6]. Then comes the final part of the installation, i.e., installing the GRUB bootloader. The GRUB Bootloader is capable of managing various operating systems on a single machine.

3.3 Configuring Anonymization

Then, we installed anonymity tools as our Operating System depends substantially on anonymization. We installed the drivers and adjusted cron jobs for engaging with the RAM, Port Usage events, and Memory Clearance. As an example, we installed the sdmem tool, which wipes out the RAM after the system boots or during runtime.

Moreover, we also configured a log cleaning script, which clears out all the logs stored in the system. We then compiled all the source files and tested them in a loop for the bugs and errors. After all the debugging and troubleshooting was completed, we converted it into an ISO file. With the help of open-source image burners, we burnt it to an USB Stick. The stick was used to flash the OS into the device using the same.

4.Outcomes and Analysis

After scrutinizing the compilation, the files were then encompassed by a PKZIP codec to form compressed blocks of scattered compartmentalized data for packing it linearly to formulate an ISO file. The packing took a long time for processing streamlined data, and because of the compression, the final output of the file was reduced from 7.42 GiB to 3.11 GiB with less to no compression loss.

4.1 Booting

When the Operating System was booted up, it used GRUB as a bootloader, to prompt for a live session. Upon affirmation, ring 3 of the kernel elevated instructions to itself, and loaded blocks of the BIOS to initialize the Window Manager for flashing the KDE Plasma GUI environment. Proc/Systemd initiated, along with other daemons to complete the loading of the system, and Cron Jobs were allocated to running daemons.

The Linux File System (LFS) got loaded along with the default tools installed by us earlier in the system configuration phase. Since sdmem was in the cron jobs, it ran periodically, clearing off memory traces from the RAM. The LSM locked several memory components of the USB Stick, to prevent any memory flushing of the same to the system disks. Drivers for interfacing hardware devices were loaded, and the network adapter, along with the media drivers could be accessible from the desktop environment itself. We integrated Linux Mint KDE drivers for faster loading and functioning.

4.2 Memory Leaks and Flushing

Memory leaks were checked by opening resourceful applications and monitoring their status of running. Consecutive running and terminating of apps took place, with applications leaking close to zero bytes of memory from the Dynamic Memory Allocation (DMA) unit.

Moreover, the driver cache which is natively dumped into the system for efficient driver interaction is also restricted, and cannot form cache on its own. The system runs on complete volatility and leaves no checksums or headers of the memory chunks directing to the Operation System. Even if the drivers leave cache to the memory, the systemctl command can be used to clear the driver and memory cache from the system.

[4]https://man7.org/linux/man-pages/man7/keyrings.7.html
[5]KDE Plasma 5 is the fifth generation of the graphical workspaces environment created by KDE for Linux systems.

To delete the cache, the command is as follows:

```
sudo systemctl vm. drop=caches=3
```

3 is the signal for clearing PageCache, Dentries and Inodes. Importing this command inside the cron jobs might perform the cache clearance periodically.

4.3 Anonymity Control

The anonymity tools installed by us provide anonymization for several synchronous services, which include DNS rotations, IP masking, MAC spoofing, User-agent rotations, Browser Cache Control, RAM and memory clearance, Anonymous File Transfer and Server Hosting, Traffic Tunneling, Device ID Randomization, etcetera. With everything entirely functional and automated, the system gains enough anonymity to mask any user surfing the web or use the system for interpersonal communications.

Upon execution, the tools initialized successfully, performing what they were obliged to do. The consecutive vectors got rotated/spoofed and could be verified by comparing the headers and checksums of the rotated and the original files. All the tools are security and privacy centric, meaning, that they do not let web engines crawl them or the user operating them, keeping the transmissions private and tunneled [7]. However, some of the tools are operated by remote servers of their respective company, and any breach is likely to give out information about this system or the window properties, such as resolution, beacons, and more. Thus, only those tools are chosen which have less to none breaches occurred in their history.

4.4 System Crashes

NIJAS OS is an extensively versatile and flexible system which can adjust its resources and capabilities by studying the system architecture. The system is very lightweight, and carries everything with a minimal resource usage and higher efficiency.

Otherwise, all the debugging errors in the system have been resolved during the development phase, and the volatile system is tested on devices satisfying every resource range, from 8 GiB of RAM and a quad-core processor, to 2 GiB of RAM with a dual-core processor. Both yield down the system with little to no lags or memory leaks.

5.Conclusion

The NIJAS OS is a privacy-centric, volatile system which holds the responsibility to make the operator near to absolute anonymous by its construction and rotation techniques. This operating system, however, has its own disadvantages of being used by cyber criminals for breaches and other operations. Thus, it is designed solely for research purposes and its usage is only authorized to a few private groups for transmitting confidential information from one node to another.

The OS has one stable release and has all necessities for different purposes, used by people prioritizing security over everyday surveillance and supervision. The simple plug-and-play operating system also leaves no port events populated by its presence. It holds the task to clear those when shutting down, making the tracing almost unfeasible.

References

[1] Smith, H. & Dinev, Tamara & Xu, Heng. (2011). Information Privacy Research: An Interdisciplinary Review. MIS Quarterly.35.989-1015.10.2307/41409970.
[2] Ullah, Shamsher & Li, Yang & Hussain, Muhammad Tanveer & Lan, Zhang. (2019). Kernel homomorphic encryption protocol. Journal of Information Security and Applications.48.10.1016/j. jisa.2019.102366.
[3] Hoare C. A. R. (1974) Monitors: An Operating System Structuring Concept. In: Hansen P. B. (eds) The Origin of Concurrent Programming. Springer, New York, NY. https://doi.org/10.1007/978-1-4757-3472-0_10
[4] Golm, Michael & Felser, Meik & Wawersich, Christian & Kleinöder, Jürgen. (2002). The JX Operating System. .45-58.
[5] Boyd-Wickizer, S., Chen, H., Chen, R., Mao, Y., Kaashoek, M. F., Morris, R. T., Pesterev, A., Stein, L., Wu, M., Dai, Y., Zhang, Y., & Zhang, Z. (2008). Corey: An Operating System for Many Cores. OSDI.
[6] Petersen, Richard. (2014). The K Desktop Environment: KDE.10.1007/978-1-4842-0067-4_10.
[7] Sardá, Thais & Natale, Simone & Sotirakopoulos, Nikos & Monaghan, Mark. (2019). Understanding online anonymity. Media, Culture & Society.41.016344371984207.10.1177/016344371984 2074