# Red Star OS

Alex M

May 2024

## 1 Introduction

When Linus Torvalds pieced together the first C files that would one day become Linux, the concept of a truly open source operating system seemed like a dream too optimistic to ever come true. Providing the lowest level of system management, an operating system requires an enormous amount of effort and dedication to perfect. For that reason, Linux has evolved into a symbol of what is achievable through mass collaboration toward a common goal. Even people who know nothing about operating systems may still respect the philosophy of providing an essential utility to everybody for free. It is quite possibly this socialist aspect of the project which caught the interest of an organization that its original creator could never have foreseen...

It is unknown for how long the Democratic People's Republic of Korea had searched for an OS before they decided to repurpose the Fedora 11 distribution of Linux. From the start, they found themselves with a task almost as monumental as Torvalds' 20 years prior. All of the emphasis on transparency and configurability which had been so deliberately emphasized in the Linux kernel's construction had to be dismantled. In its place, developers added several services intended to advance the interests of their regime. Linux was no longer just socialist — now it was fascist too.

For a few years, design specifics of the DPRK's creation were unknown in the West, but a leak in 2014 provided a newly informed view of its mechanism. Researchers Florian Grunow and Niklaus Schiess subsequently released a report on their close examination of the operating system disk image. Their report provides the basis for this summary.

## 2 Benign(ish) Features

Red Star Linux, as its developers named it, diverged from its Fedora roots in a few key ways. Graphically, its developers sought to recreate the appearance of OS X (possibly inspired by Apple's own totalitarian stance on systems development). In fact, the taskbar and minimization mechanics are completely indistinguishable from an early 2010's MacBook setup. Instead of Safari, however, the developers of Red Star reasoned that the DPRK's citizens need a new

browser since they had no access to the internet in the traditional sense. Their solution, Naenara (which is Korean for "my country") was benign enough. It allowed users to connect to any of the IP addresses assigned to the various servers on the country's local network[1].

Their next addition, an encryption tool called Bokem (Korean for "Sword") also seemed harmless. Users of Red Star can encrypt any of their files, and ensure security for any information on their system. It is unknown as to whether a master key exists for Bokem-encrypted files[1].

Furthermore, Red Star comes stock with an Open Office equivalent called Sogwang office, a package manager, and a mechanism to access root privileges.

# 3 Malicious Features

To undermine the inherent focus on configurability provided by the Linux kernel, Red Star comes with a service called "securityd" which ensures that crucial files are not tampered with. If it detects a modification to any of the critical operating system files that are under its protection, it will force a reboot of the entire system. Upon startup, the tampered file will still be present, thereby prompting another reboot. Quickly, the user will find themselves trapped in a never-ending reboot loop. As a kernel program, securityd has the power to render the computer unusable indefinitely (or at least until its user can find another copy of Red Star)[1].

One of the most important features the developers added was the ability to hide the source code of certain services. By obscuring the true intent of the various daemons running on a Red Star machine, the developers were free to add unsavory functionality to otherwise legitimate programs. Ironically, the antivirus scanner, "scnprc" is one such program that utilizes this trojan horse scheme. It is unclear as to whether the antivirus actually has any ability to detect malware, but upon scanning the user's files, it will automatically delete anything that is deemed unsavory to the state. Grunow and Schiess strongly suspect that scnprc serves no other use than to delete media containing thoughtcrime but are unable to confirm their suspicions due to their own limited knowledge of the Korean language. However, scnprc cannot be disabled (or viewed), and end users are stuck with it[1].

The most insidious feature is a service called "opprc" which watermarks every new piece of media that it encounters with a the user's hard disk serial number. The serial number is encrypted with a common key that is hard-coded into every copy of Red Star. Interestingly, this common key is a binary string that translates to two dates. The first date in the key is Madonna's birthday, and there is some speculation that this may be due to the prevalence of Madonna fans within the country (yes, really)[1].

Regardless of the rationale behind the choice of encryption key, the end result is that every piece of media that has been read by a Red Star machine carries that machine's watermark. In theory, this technology could be used to construct a web of every file's provenance. Such invasive tracking, coupled with a registry

of hard drives within the country, could easily be used to track dissidents and distributors of banned media[1].

All of these custom components are reliant on one another to work and are very difficult to remove. Grunow and Schiess found a very specific order of removal that was effective in clearing a Red Star machine of malware. It goes like this:

1. Gain root access
2. Kill securityd to prevent the infinite reboot loop
3. kill scnprc
4. kill opprc
5. replace the internal function that validates the OS with a function that always returns 1
6. Delete the initialization file which is called on startup to start securityd, scnprc, and opprc.

Ultimately, Grunow and Schiess concluded that it would be nearly impossible for any North Korean citizen to discover this exploit without access to a computer with another operating system. They determined that the programmers who created Red Star had done a remarkable job in securing Red Star from its users[1].

Surprisingly, Grunow and Schiess did not locate any obvious back doors. However, they decided that, because Red Star OS is also used for servers and official government functions, a backdoor presented too large of a liability. It is also possible that the leak of Red Star was an intentional operation performed by the North Korean government to hide the more evil aspects of their *actual* OS.

# 4 Process Management

Linux adheres to a well known model for handling processes and threads. Each process receives its own address space and process control board which is used to track the state of its registers when it descheduled. Upon getting rescheduled, a processes' registers are recalled from its PCB and it is able to resume execution from where it left off. New processes are created through forking. Forked processes contain pointers to the parent process's address space. This "copy-on-write" protocol protects the system from copying the entire program state from the parent process before any changes occur. A lot of the time, a new process is told to run some executable shortly after its creation and, in this case, it is helpful for that process to not have wasted time copying its parent's address space [7].

Linux also supports lightweight runnable entities called threads which share the address space of their parent processes. Threads can prove exceptionally

handy when data has to be shared between streams of execution. Because threads do not have their own address space, they do not require as much data to be stored when they are descheduled. As such, each thread is allocated a thread control board which stores thread-specific information[7].

# 5 Memory Management

Despite being based on Fedora 11, Red Star's uses the 2.6.38 Linux kernel which shipped with Fedora 15. This version of the kernel featured significant upgrades in memory management and task scheduling.

Linux memory management is reliant on demand paging with either 48 or 57-bit virtual addresses for x86[2]. Linux makes use of the translation lookaside buffer to cache translations for virtual addressses. Should an address fail to appear in the the TLB, the page tables will be referenced to locate the physical address. To facilitate this translation, the address of the top-level page table is stored in a register. Unlike a single level page table system, Linux uses hierarchical page tabling. This more obfuscated process entails using the highest bits in the virtual address as an index for the top level page table. The page table entry corresponding with this index points to another page table indexed by the next highest bits of the address. Only the lowest bits in the virtual address refer to the offset in the frame that is desired. If the bottom-level table, which contains the desired address translation, indicates that the page is out on disk, a page fault must occur[4].

Because the process for address translation is so complicated, a mechanism called *huge TLB filesystem* exists to facilitate management of frames that are up to a full gigabyte. The principle idea is that if there are just a few very large frames that are needed at any given time, then the TLB will never fill and there will rarely be a need for a page table lookup. In theory this functionality could be used to greatly enhance CPU performance for certain tasks. The disadvantage of this approach is that only a few pages will fit in memory if only a modest amount of RAM is installed on the system, and the penalty for incurring a page fault will be tremendous. Grunow and Schiess did not mention what the standard frame size is for Red Star OS or whether the kernel had been modified to prohibit the usage of huge pages, but most operating system (including Fedora 11) ship with 4KB frame sizes as default[4].

Linux also acknowledges that not all memory is created equal. Distinct cores on modern CPUs have easier access to some memory addresses than others. With a system called NUMA (Non-Uniform Memory Access), the operating system is able to assign memory into banks that are closer to the processor that needs it most. The advantage of this methodology is a reduction in latency which can speed up memory-intensive processes.

The most recent change to Linux's memory management system for version 2.6.38 sought to relieve contention on a crucial semaphore called mmap_sem[3]. When a process requests more memory, the operating system must find a spot in the virtual address space large enough to accommodate the request. Linux

manages these *Virtual Memory Areas* in an augmented red-black tree which is considered a critical section. The semaphore which guards this structure, mmap_sem, causes a bottleneck during large calls to mlock(). The function of mlock is to pin frames in memory and protect them from eviction, thereby helping to flag critical areas of memory in advance and avoid future page faults. Unfortunately, this operation can take a long time because if any of the frames referenced by the call to mlock are not in memory, then they must be faulted in individually from disk. This process can be painfully slow if there are many frames referenced. Through the entire faulting process, the mmap_sem is held by mlock while other processes starve for access to the the VMA tree. With version 2.6.38 of the Linux kernel, mlock is capable of pausing its execution and releasing mmap_sem when it detects contention[5]. This change is a major step forward in Linux's memory management and it makes sense that this version of the kernel was used in Red Star OS.

# 6 Task Scheduling

The Linux scheduler is not a traditional Multilevel Feedback Queue, but rather an entirely new invention called the *Completely Fair Scheduler*. The CFS is a red-black tree that orders tasks based on how much virtual CPU time they have received. Linux will always schedule the task that has received the least virtual CPU time which is always stored as the leftmost child node in the red black tree. When a new task is executed, its time value is initialized with the time value of the task which has received the least CPU time. This policy causes new tasks to take precedence over tasks which have had more opportunity to run on the CPU[6].

The goal of this implementation is to allow all tasks to receive exactly the same amount of time in on the processor with only a small compromise in terms of overhead. Because a red-black tree is maintained, each scheduling and descheduling operation runs in logarithmic time.

# 7 Conclusion

While it is fascinating to study the various mechanisms by which a government can control its population through an operating system, examinations of Red Star OS should be conducted with the upmost care. The behavior of this operating system is largely unknown, but it is clear that users of Red Star cede much of the control that Linux users tend to expect. Users who are considering switching to Linux are encouraged to consider other distributions that are less filled with spyware.

# References


[1] Grunow, Florian, and Niklaus Schiess. “Lifting the Fog on Red Star OS.” Home, media.ccc.de/v/32c3-7174-lifting_the_fog_on_red_star_os. Accessed 4 May 2024.

[2] “The Linux Kernel.” 29.3. Memory Management - The Linux Kernel Documentation, www.kernel.org/doc/html/latest/arch/x86/x86_64/mm.html. Accessed 4 May 2024.

[3] Corbet, Johnathan. “How to Get Rid of Mmap_sem.” [LWN.Net], lwn.net/Articles/787629/. Accessed 4 May 2024.

[4] “The Linux Kernel.” Concepts Overview - The Linux Kernel Documentation, docs.kernel.org/admin-guide/mm/concepts.html. Accessed 4 May 2024.

[5] Lespinasse, Michel. Mlock: Only Hold MMAP_SEM in Shared Mode When Faulting in Pages - Kernel/Git/Torvalds/Linux.Git - Linux Kernel Source Tree, git.kernel.org/pub/scm/linux/kernel/git/torvalds/linux.git/commit/?id=fed067da46ad3b9acedaf794a5f05d0bc153280b. Accessed 4 May 2024.

[6] “The Linux Kernel.” CFS Scheduler - The Linux Kernel Documentation, docs.kernel.org/scheduler/sched-design-CFS.html. Accessed 4 May 2024.

[7] baeldung, Written by: “Linux Process vs. Thread.” Baeldung on Linux, 18 Mar. 2024, www.baeldung.com/linux/process-vs-thread.