Rootkit-Resistant Disks

KEVIN BUTLER, STEPHEN MCLAUGHLIN, AND PATRICK MCDANIEL

Persistent Rootkits

Computing resources are under constant siege. Attacks can originate from both known and unknown sources, and little is trustworthy: even installing commercial devices and components can introduce malware to a system.

Being able to control a system means the ability to use it for financial gain as part of a botnet for profit or as a weapon for cyberwarfare. As a result, there is tremendous incentive to maintain control of an exploited system. This is accomplished by using a ***rootkit***.

Once a rootkit is installed on a system, it can become *persistent*, where it modifies the on-disk system image (binaries and configuration files). The only way to remove the rootkit is to erase the disk.

Rootkit ProtectionGoals

A disk protecting against rootkits must have the following properties:

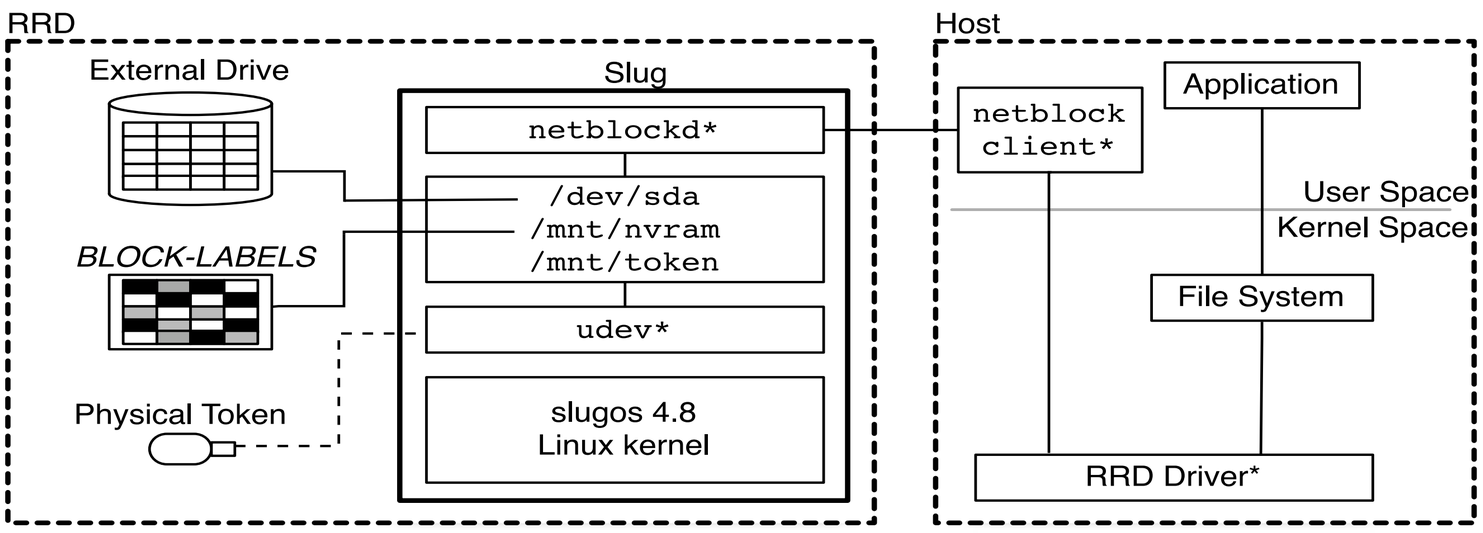
1. It must protect against real (currently-deployed) rootkits.

2. It must be usable without user interaction and with minimal administration. Normal operation should be *transparent* to the user.

3. It must be highly performant.

4. It must have low storage overhead. It should consume as little ancillary metadata as possible.

Design of a Rootkit-Resistant Disk



1. An administrative token containing a **system write capability** is placed in the USB port of the *external hard drive enclosure* during the installation of the operating system. This ensures that the disk processor has access to the capability, but the host CPU does not -- ensuirng a trusted path between storage and the user.

2. Associated with every block is a label indicating whether it is **immutable**. Disk blocks associated with immutable system binaries and data are marked during system installation. The token is removed at the completion of the installation.

3. Any modification of an immutable system block during normal operation of the host OS is ***blocked by the disk processor.***

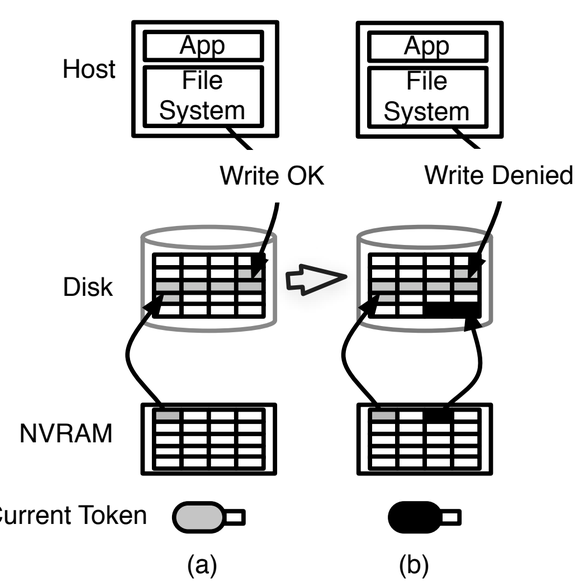
4. System upgrades are performed by safely booting the system with the token placed in the device (and the system write capability read), and the appropriate blocks marked. The token is removed at the completion of the upgrade.

Block Labeling

In (a) on the figure, the host system writes a file to unused disk space initially using the gray token. The write is allowed and the blocks written are **labeled** with the token.

In part (b), any blocks written by the black token are labeled on the disk. However, the gray blocks cannot be overwritten as long as the gray token is not present.

We designed a simple disk policy that encodes thee rules algorithmically on the disk.



Evaluating RRD Performance

Configuration	Completion(s)	% Overhead	95% C.I.
nosec	443.8	—	[437.3, 450.3]
sec	453.6	2.2 %	[446.4, 461.0]

Base System Copy completion times with and without RRD security

Component	Total Time (s)	% Of Measured	95% C.I.
disk	340.5	53.8 %	[340.1, 340.9]
net	288.1	44.7 %	[287.9, 288.5]
sec	16.4	2.5 %	[16.1, 16.7]

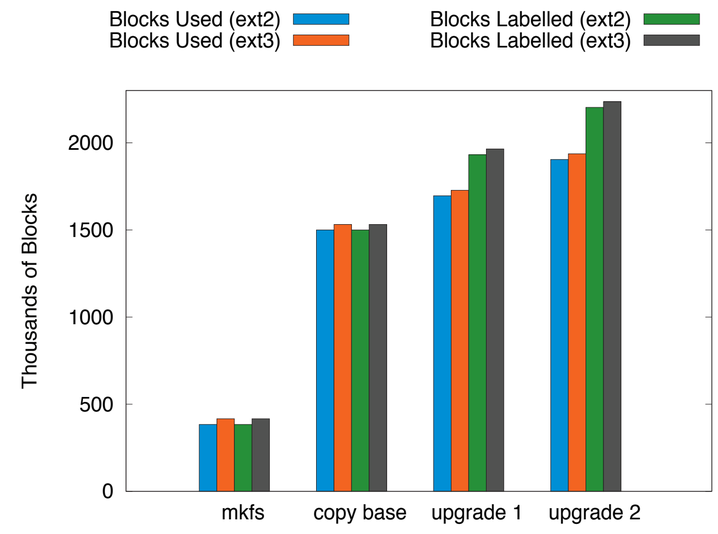
Microbenchmarks of Base System Copy operation

Configuration	TPS	% Decrease	95% C.I.
nosec	235.1	—	[233.2, 236.7]
sec	231.7	1.4%	[230.3, 232.7]

Postmark throughput benchmark with/without security

The tables on the left show that the RRD security adds approximately 2.2% overhead to a system copy operation, which requires large amounts of disk I/O. In a similar vein, a test of the I/O intensive Postmark benchmark shows less than 1.5% change in throughput with the RRD. Microbenchmarks show that security operations account for only 2.5% of total overhead.

The figure on the right shows that there is little *label creep* - most labeled blocks are used by the filesystem and not abandoned. We successfully defended an RRD-enabled system against the **Mood-NT** rootkit.



RESEARCH SUPPORTED BY NSF AND SEAGATE