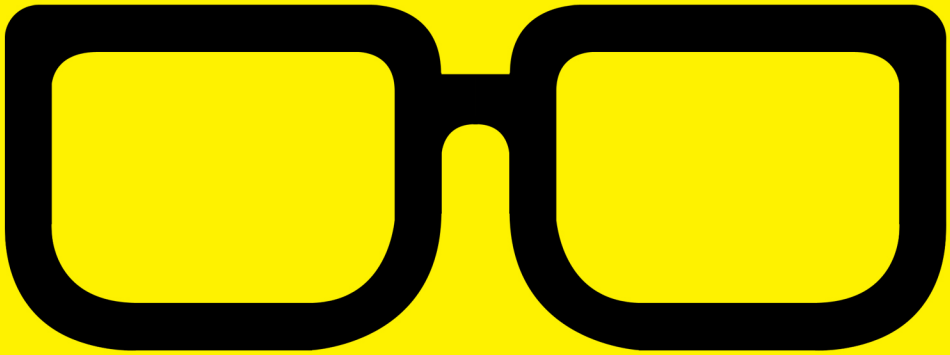
SPONSORED BY

helpsystems

GEEK GUIDE



Self-Audit: Checking Assumptions at the Door

LINUX™ JOURNAL

Table of Contents

GREG BLEDSOE is **VP** of **O**perations at **P**ersonal, **I**nc. (http://www.personal.com). **He** is **CEH** and **CPT** certified and has 20 years of hard-fought experience in security and operations. **Y**ou can reach him at lj@bledsoehome.net, or via **T**witter: @geek_king.

GEEK GUIDES:

Mission-critical information for the most technical people on the planet.

About the Sponsor

HelpSystems

HelpSystems makes IT lives easier by meeting critical needs like IT and business process automation, and system security. Policy Minder is a robust, user-friendly security monitoring solution that allows systems administrators to spend less time developing compliance reports, administering security, and managing scripts. Automating these labor-intensive tasks doesn't just improve efficiency—it gives you control over settings and activities on your system that put business-critical data at risk.

For more information on Policy Minder, see http://www.helpsystems.com/policy-minder.

Self-Audit: Checking Assumptions at the Door

GREG BLEDSOE

Introduction

One could argue that civilization's material success arose because the concept of experimental verification became the foundation of the scientific method. By ruthlessly applying the scientific method and experimental verification to your assumptions, you can bridge the gap between beliefs and reality. You learn to make decisions based on data, experience, observation and tested principles. This has worked quite well because, contrary to postmodernist

Sysadmins, devops engineers and security personnel alike often are kept awake at night by any number of paradoxical questions when they sense that their assumptions are untested.

philosophy, there is indeed a verifiable reality external to perception and belief that, when ignored, will assert itself in the form of bumps, bruises and harm. Much of this harm can be avoided if you do the best job you can at validating your assumptions before their falsehood costs more than you wish to pay.

Applying this concept to modern technological operations, and particularly to security operations, means self-auditing early and often through the process of delivering services to your clients, partners or business units. Self-auditing is how you apply the scientific method to your environments and come to know whether what you *believe* to be true about your systems—that is, that they comply with your policies, are secure and under your exclusive control—actually is true.

Sysadmins, devops engineers and security personnel alike often are kept awake at night by any number of paradoxical questions when they sense that their assumptions are untested. How should I monitor my monitoring to be sure it will alert me when there is an issue that needs my intervention? How can I be sure that the servers and processes in the environment haven't been rooted already?

What if I'm really just a brain in a jar connected to an experience machine? Despite every automated precaution and deductive reasoning based on assumptions about my process and technology, am I really *sure*?

In the quest to get better sleep, it is a good idea to give your systems a thorough self-audit every so often, but this is where a sysadmin can get lost in the weeds. Knowing what to check and how to be as sure as possible that things are okay can be a moving target. Although the basic principles can be enumerated easily (verify configurations and inspect everything), the practical reality of getting this done can be daunting.

In this ebook, I provide a jump-start and walk through some of the checks to go through to self-audit a Linux server. I use the basic and omnipresent commands most likely to be available everywhere as much as possible and give a pointer on how to secure the other tools to use for the rest.

Two Steps Necessary for Security

Assumptions are insidious, and the one assumption you don't realize you are making may be the one check you don't make that invalidates the rest of your results. So as far as possible, it is critically important to start the process of a self-audit with no preconceptions. Although some assumptions obviously are necessary (for instance, I assume I'm not dreaming right now), or you'd never do anything, you can shed your underlying beliefs about things only to a certain degree. Nevertheless, every assumption you can state and test explicitly brings you that much closer to

In fact, the one assumption that will serve you well in this process is to assume that every layer of security already has been compromised and be determined to find the evidence.

surety in the integrity of your systems.

Modern sysadmins/devops engineers/security personnel really need to be sure of two things. First, be sure that your systems are configured in the way you believe they are configured—for example, making sure that the intended firewall rules are applied and that the password policy is enforced as you've committed to in your policies and procedures, and so on. (Of course, whether those are the appropriate controls, policies and procedures is a question for another day.) After all, these are not things you want pointed out in a forensic analysis after the fact. The second thing you need to do is look for evidence that your security measures have been defeated. In fact, the one assumption that will serve you well in this process is to assume that every layer of security already has been compromised and be determined to find the evidence. Set a high bar for coming to the belief your system has *not* been compromised. Believe that the evidence is there somewhere, in hidden files, folders and processes, buried somewhere in some log, and that it is your job to find it. Dig in until you can conclude only that the system hasn't been compromised.

Much of this process can be automated. I would submit that you should automate as much as possible for several reasons. For one thing, an automated process doesn't forget to do checks, doesn't overlook things or miss an item on the checklist, and it doesn't make human errors. The automation itself can have human errors, and that has to be mitigated by adequate logging along with testing your scripts thoroughly and periodically to maintain trust in their effectiveness. Scripts that will do things like attempt to set weak passwords and look for unexpected listening ports are the first tools you need to have in your toolbox. Now that I've identified these two necessary steps (verify configurations and inspect for evidence of compromise), let's take a look at what that means in practice.

Verify Configurations

Verifying configurations isn't something that any tool can do automatically without your input. First, you need to have the comprehensive list of requirements and policies that your systems are subject to, and then step by step, line by line, verify that these have been implemented correctly, not just by looking at the configuration, but by testing those assumptions in practical ways.

For example, verify firewall rules by both examining startup configuration files however you manage them, and verify the running configuration with `iptables -L -vn` to verify the open ports. If you have a password policy set, try a simple password that doesn't meet the policy. If it works, something is wrong and to /etc/pam.d you go! If you have user- and group-level SSH permissions, verify them

Verifying large numbers of items on large numbers of servers can become quite cumbersome, so developing an automated and audit management methodology along with the appropriate usage of tools becomes key.

both in the configuration and by attempting to log in as unauthorized users. This practical testing often will expose issues you didn't anticipate, and this is the basic idea of having a quality assurance program.

There are a lot of items to check, and going through how to verify every possible configuration item would take more space than I have here. As I mentioned previously, this all depends completely on your environment and your organization's policies. Verifying large numbers of items on large numbers of servers can become quite cumbersome, so developing an automated and audit management methodology along with the appropriate usage of tools becomes key. Tools like HelpSystems Policy Minder come to mind as good options to investigate.

Assume Security Has Been Compromised

After an exhaustive review of the configuration files and running configurations, let's move on to phase two: assume all layers of defense have been circumvented. This phase breaks down into the following sections: the network, the

filesystem and the kernel.

A few tools exist that look over all three areas for signs of badness, and you should make use of them. If you've done some basic server hardening (as you should), you likely have some scanning tools installed and performing at least daily checks. If not, you really should make that happen. Even if you do, it's also likely that you have some whitelists set up in your configurations to prevent daily false positives on outbound ports or files that seem suspicious to the scanner. For a self-audit, since you're abandoning all preconceptions and assuming you've been compromised, you'll run all checks without the whitelists and investigate every warning, disregarding your instinct to assume anything is a false positive and instead assume that each one means the system is compromised—until and unless you can prove otherwise.

Tools to Scan for Malware

I suggest you use at least the following tools for a self-audit to scan for malware: rkhunter, chkrootkit, LMD (Linux malware detect) and lynis. They do perform some overlapping checks, but they are different enough that there is value in running them all. If you are scripting these either through your custom management system or through a tool like Policy Minder, I suggest you have them e-mail you the results, which means your systems need to be configured to send e-mail. For the purposes of this ebook, I assume you are going to investigate all results on the server in question and e-mail those results to root@localhost or view them on-screen or in logfiles.

It is important to note that these tools run file integrity checkers, which means that they check many files for unexpected changes. So, if a monitored file changes, you have to inform the tool immediately, or you will get false positives. From the time the server is deployed until it is retired, you must be fastidious about this, or you open yourself to the prospect of missing a malware infection. When you make changes or when you install software, you should run these checks beforehand, then these integrity checkers must be updated immediately after the change.

rkhunter: First, let's look at rkhunter. Immediately on server deploy, run it with these options:

```
rkhunter --propupd --update --versioncheck
```

This makes sure you are on the latest malware signatures and version of the software, and it establishes a baseline configuration to check against. It will produce warnings when files change between runs of `--propupd`. As stated, the `--propupd` option must be executed every time software is installed or system configurations are updated.

If you never have run any of these scanners before, the value is reduced but not eliminated. Much Linux malware still can be detected. If you have whitelisted any scripts, ports and files in rkhunter, create a new configuration file that lacks those configuration options, and run rkhunter with these options:

```
rkhunter -c --vl --configfile /path/to/newconfig
 ➥-l /path/to/logfile
```

chkrootkit is generally more of a standalone tool run without a configuration file that checks for signs of infection.

This will perform all the checks, log verbosely, use your custom config and log to a file of your choosing. If you are running these sequentially with a tool for later inspection, you can have these e-mailed via the config file. The key here is to investigate every anomaly that you might otherwise be tempted to skip—for instance, if you get a message like the following:

```
[22:09:02] Warning: Hidden directory found:
➥'/etc/.java: directory '
```

Just because you think you remember something about this, don't assume that it's normal and a false positive! Investigate thoroughly. Start by performing an Internet search on the message. In my case, when I investigated, I found that this *is* a normal warning on Ubuntu when you run Sun-Oracle's Java Virtual Machine, but it's something that is handled differently on other distributions like RHEL, so rkhunter doesn't expect hidden files under /dev/, and you get a warning.

chkrootkit: Next, run chkrootkit. chkrootkit is generally more of a standalone tool run without a configuration file that checks for signs of infection. I usually capture all

output to a log file like this:

```
chkrootkit &> file.log
```

On a server I run at home, I got two alarming results:

```
Searching for Adore Worm...
/usr/lib/plexmediaserver/start.sh

Searching for Suckit rootkit...         Warning:
/sbin/init INFECTED
```

Again, I am going to assume that these are real infections unless I can definitively prove otherwise. Upon inspection and after a lot of research, I eventually found that the signature of the Adore Worm is start.sh somewhere in the path under /usr/lib. Evidently, plexmediaserver (which I use to send music to various devices around the house) just happens to match the signature. So, I inspected the plex script against the version installed by the package and found it unchanged.

For the Suckit rootkit, I eventually found this bug in the chkrootkit package for Ubuntu (the distribution of my home server): https://bugs.launchpad.net/cyborg/+bug/454566. This verifies a false positive and points me to a URL that tells me how to see if I actually am infected with the Suckit rootkit. I follow the link and verify that I am not.

LMD: Next, I recommend running LMD. LMD is a less common tool and not generally in distribution repositories, so let's look at how to install it. LMD can use the clamav

detection engine with LMD's signatures, which gives about 4x better performance, so even though it isn't necessary, you might first install clamav from your distribution's repositories (either as root or with sudo):

```
apt-get install clamav
```

Then, download and install the LMD tarball:

```
wget http://www.rfxn.com/downloads/maldetect-current.tar.gz

tar -xvf maldetect-current.tar.gz ; cd maldetect

./install.sh
```

This puts everything in /usr/local/maldetect, so issue:

```
cd /usr/local/maldetect
```

LMD needs a basic configuration file:

```
echo '''email_alert=1
email_addr=root@localhost
email_subj="Malware alerts for $HOSTNAME - $(date +%Y-%m-%d)"
quar_hits=0
quar_clean=0
quar_susp=0
clam_av=1''' >> conf.maldet
```

Here are the options:

- `email_alert=1` — get e-mail alerts, which can be sent outside the local system.
- `email_subj` and `email_addr` — self-explanatory.
- `quar_hits` — set default quarantine action: 0 is alert only, 1 is quarantine.
- `quar_clean` — set whether you want to clean string-based malware injections.
- `quar_susp` — for users with hits, 1 is disable the account.
- `clamav_scan=1` — detect the presence of the ClamAV binary and use as engine.

To run it, use:

```
maldet --scan-all /path
```

You probably don't want to scan network filesystems unless they are very fast to avoid this taking an extraordinarily long time, so you may need to script a progression of paths for it. But, you'll get results like this:

```
sudo /usr/local/maldetect/maldet -u -d
Linux Malware Detect v1.5
    (C) 2002-2015, R-<remark role="fix-me"> Networks
    ➥<proj@rfxn.com>
```

```
    (C) 2015, Ryan MacDonald <ryan@rfxn.com>
This program may be freely redistributed under the terms
of the GNU GPL v2

maldet(28656): {sigup} performing signature update check...
maldet(28656): {sigup} local signature set is version
 ➥2015121610247
maldet(28656): {sigup} latest signature set already installed

sudo maldet --scan-all /home
Linux Malware Detect v1.5
    (C) 2002-2015, R-<remark role="fix-me"> Networks
     ➥<proj@rfxn.com>
    (C) 2015, Ryan MacDonald <ryan@rfxn.com>
This program may be freely redistributed under the terms
of the GNU GPL v2

maldet(27592): {scan} signatures loaded: 10822
 ➥(8908 MD5 / 1914 HEX / 0 USER)
maldet(27592): {scan} building file list for /home,
 ➥this might take a while...
maldet(27592): {scan} setting nice scheduler priorities for
 ➥all operations: cpunice 19 , ionice 6
maldet(27592): {scan} file list completed in 77s, found
 ➥143698 files...
maldet(27592): {scan} found clamav binary at
 ➥/usr/bin/clamscan, using clamav scanner engine...
maldet(27592): {scan} scan of /home (143698 files)
 ➥in progress...
maldet(27592): {scan} processing scan results for hits:
```

```
 ➥3 hits 0 cleaned
maldet(27592): {scan} scan completed on /home: files
 ➥143698, malware hits 3, cleaned hits 0, time 539s
maldet(27592): {scan} scan report saved, to view run:
 ➥maldet --report 160111-2203.27592
maldet(27592): {scan} quarantine is disabled! set
 ➥quarantine_hits=1 in conf.maldet or to quarantine
 ➥results run: maldet -q 160111-2203.27592
```

In my case, it found three hits!
Looking further, I find:

```
/usr/local/maldetect/maldet --report

HOST:      greg-Linuxbox
SCAN ID:   160111-2203.27592
STARTED:   Jan 11 2016 22:03:43 -0500
COMPLETED: Jan 11 2016 22:12:42 -0500
ELAPSED:   539s [find: 77s]

PATH:          /home
TOTAL FILES:   143698
TOTAL HITS:    3
TOTAL CLEANED: 0

WARNING: Automatic quarantine is currently disabled,
detected threats are still accessible to users!
To enable, set quarantine_hits=1 and/or to quarantine
hits from this scan run:
/usr/local/sbin/maldet -q 160111-2203.27592
```

```
FILE HIT LIST:
{HEX}gzbase64.inject.unclassed.15  :
/home/user/maldetect-1.5/files/clean/gzbase64.inject.unclassed
{HEX}php.exe.globals.399           :  /home/user/Downloads/
➥DVWA-1.9.zip
{HEX}gzbase64.inject.unclassed.15  :  /home/user/
➥maldetect-current.tar.gz
==============================================
Linux Malware Detect v1.5 < proj@rfxn.com >
```

It found its own installation directory and the compressed tar file, because both contain the signatures that it is searching for, which makes sense. It also finds Damn Vulnerable Web Application, an intentionally malware-riddled test setup for pen testers, which I have downloaded—nothing here I didn't expect.

lynis: The final stop among the scanners is lynis, a very capable open-source auditing tool. Not only will it scan for problems, it also will make suggestions for how you can improve your system security—very valuable feedback indeed! You will get a lot of output like this:

```
lynis -Q

... snip

[+] Accounting
------------------------------------
  - Checking accounting information... [ NOT FOUND ]
```

```
  - Checking sysstat accounting data [ ENABLED ]
  - Checking auditd  [ ENABLED ]
    - Checking audit rules  [ SUGGESTION ]
    - Checking audit configuration file  [ OK ]
    - Checking auditd log file [ FOUND ]

[+] Time and Synchronization
------------------------------------
  - Checking running NTP daemon (ntpd)... [ NOT FOUND ]
  - Checking running NTP daemon (timed)... [ NOT FOUND ]
  - Checking running NTP daemon (dntpd)...  [ NOT FOUND ]
  - Checking NTP client in crontab file
    ➥(/etc/anacrontab)... [ NOT FOUND ]
  - Checking NTP client in crontab file
    ➥(/etc/crontab)... [ NOT FOUND ]
  - Checking NTP client in cron.d files... [ NOT FOUND ]
  - Checking event based ntpdate (if-up)... [ FOUND ]
  - Checking for a running NTP daemon or client... [ OK ]

[+] Cryptography
------------------------------------
  - Checking SSL certificate expiration... [ WARNING ]

... snip
```

At the end, you also get a summary and suggestions, like so:

```
-[ Lynis 1.3.9 Results ]-
```

```
  Tests performed: 167

  Warnings:
  ----------------------------
  - Found mail_name in SMTP banner, and/or mail_name contains
     'Postfix' [test:MAIL-8818]
  - Found SSL certificate expiration
(/etc/ssl/certs/ca-certificates.crt) [test:CRYP-7902]

  Suggestions:
  ----------------------------
  - update to the latest stable release.
  - Install a PAM module for password strength testing
    like pam_cracklib or pam_passwdqc [test:AUTH-9262]
  - Configure password aging limits to enforce password
    changing on a regular basis [test:AUTH-9286]
  - Default umask in /etc/login.defs could be more strict
    like 027 [test:AUTH-9328]
  - Default umask in /etc/init.d/rc could be more strict
    like 027 [test:AUTH-9328]
  - To decrease the impact of a full /tmp file system,
    place /tmp on a separated partition [test:FILE-6310]
  - Purge old/removed packages (30 found) with aptitude
    purge or dpkg --purge command. This will cleanup old
    configuration files, cron jobs and startup scripts.
    [test:PKGS-7346]
  - Check what deleted files are still in use and why.
    [test:LOGG-2190]
  - Add a legal banner to /etc/issue, to warn unauthorized
    users [test:BANN-7126]
```

On a number of occasions when nothing else has returned results, I've found that a server was compromised by directly checking for ports that are opened in stealth mode.

```
- Add legal banner to /etc/issue.net, to warn unauthorized
  users [test:BANN-7130]
- Renew SSL expired certificates. [test:CRYP-7902]
- One or more sysctl values differ from the scan profile
  and could be tweaked [test:KRNL-6000]
- Harden the system by removing unneeded compilers. This can
  decrease the chance of customized trojans, backdoors
  and rootkits to be compiled and installed [test:HRDN-7220]
- Harden compilers and restrict access to world [test:HRDN-7222]
```

Evaluate and change policy implementation accordingly.

Checking for Stealth Ports

More can be done depending on the server type—things like running Nessus against running applications—but let's move along now to a very valuable secret that I'll share with you. On a number of occasions when nothing else has returned results, I've found that a server was compromised by directly checking for ports that are opened in stealth mode. This often will turn up hidden malware phoning home that nothing else finds. To check for these stealth ports, first find out what ports are not hidden from you and what processes have them open.

lsof: You may not have lsof installed by default, but I suggest installing it, as it's an invaluable tool:

```
lsof -ni
```

Once you know what ports should be available to you, you can run nmap on the local system and see if any unexpected ports are listening for control signaling, which means you probably are pwned and part of a botnet.

Install nmap via `apt-get install nmap` or similar, and then point it at your interfaces, one at a time, to scan every TCP and UDP port (run as root or via sudo):

```
nmap -n -PN -sT -sU -p- 192.168.1.x  #internal network

nmap -n -PN -sT -sU -p- 24.24.7.x    #external network
```

If anything responds that doesn't show up in `lsof` or `netstat -plunt`, you need to look very very closely, as you probably have a rootkit in action.

The last thing to do to make sure you don't have a rootkit that is waiting on a packet "port knock" sequence to open up is actually to attempt to bind to ports and look for "in-use" errors. In the past, I've always run a quick script to brute-force the entire TCP space of the following variety:

```
for i in [ 1..65535 ]
  do
  nc -l -p $i
  done
```

Then, I manually check against `lsof` and `netstat` output.

unhide: Thankfully, most distributions now are shipping with a very handy utility called unhide that does all this for you automatically. After the installing via `apt-get install unhide` or similar, you simply can run:

```
unhide-tcp

Unhide-tcp 20121229
Copyright C 2012 Yago Jesus & Patrick Gouin
License GPLv3+ : GNU GPL version 3 or later
http://www.unhide-forensics.info
Used options:
[*]Starting TCP checking
[*]Starting UDP checking
```

If you get no warnings about hidden ports, you are probably good.

unhide also comes with a utility to perform a variety of checks including brute-forcing the entire PID space (which can take a long time but is pretty reliable) that allows you to audit your running kernel to see if you have processes hidden from your normal commands, like `ps`. You can run it like this:

```
unhide brute

Unhide 20121229
Copyright C 2012 Yago Jesus & Patrick Gouin
License GPLv3+ : GNU GPL version 3 or later
http://www.unhide-forensics.info
```

```
NOTE : This version of unhide is for systems using Linux >= 2.6

Used options:
[*]Starting scanning using brute force against PIDS with fork()

Found HIDDEN PID: 28353
Cmdline: “<none>”
Executable: “<no link>”
“<none>  ... maybe a transitory process”

[*]Starting scanning using brute force against PIDS with
pthread functions
```

If, as I did above, you get a result, you should run it again to make sure you didn't just catch a process coming or going (as I did here). If the results come up clean a second or third time, you probably aren't compromised.

Rootkits

This brings me to a very important point and another of those insomnia-inducing paradoxes. If your kernel is rooted with malware, you can't trust anything that comes out of it, and all the tests you've done depend on the accuracy of what comes from the kernel to be reliable. A sufficiently sophisticated polymorphic polyphasic rootkit still could potentially hide from all of these measures. It is unlikely, but possible.

At the end of the day, if you have any doubt whatsoever, you should use a bootable media and scan

At the end of the day, if you have any doubt whatsoever, you should use a bootable media and scan the system again running from a "known good" kernel.

the system again running from a "known good" kernel. A number of forensically oriented live CD/DVD/USB-type systems are available, notably CAINE (Computer Aided Investigative Environment) and a new up-and-comer in release-candidate stage called DEFT (Digital Evidence and Forensics Toolkit).

If, after performing this process again using verified safe bootable media, you still don't have confidence in the system's integrity, destroying and redeploying from scratch without reusing any data, files or configurations that ever touched the suspect system may be your only recourse.

This is where a strong devops orientation in your environment can save you. If you have a strong management/deployment automation process, suspecting your server of being rooted doesn't have to be the worst thing in the world, as it can be trivial to replace. If you are *sure* your server is rooted, depending on your legal jurisdiction and company policy, you may be required to preserve the affected server for investigation, so be sure you know your responsibility in this regard before destroying it.

Conclusion

If you manage an environment of more than a few servers, you will be well served to develop an auditing regime using appropriate tools to allow you to keep auditing your servers regularly without taking away from the time you need to meet the other requirements of your sysadmin/devops/security engineer job, and I recommend looking into multiplatform tools like HelpSystems Policy Minder, which can manage your systems and schedule your audit scripts.

So to recap quickly, self-auditing your servers is an important practice to verify that you are following your own policies and that your security measures have not been overcome. You need to verify experimentally that your policies are implemented correctly in the system's configuration and that your system is still under your exclusive control. This requires an in-depth knowledge of your own technology stack—from the policies themselves to the reasons for those policies to the actual implementation of each and every policy in configuration, and a possibly tedious comparison and testing of the intention of those policies to the actual working configuration of the systems themselves. This is where the rubber of theoretical security meets the road of reality and where the good operations organization is separated from the excellent. Apply these concepts that allow you to deduce principles, and by the diligent application of those principles, produce results—the results culminating in your self-audit and giving you a reasonable confidence that your environment is secure.■