Categories

- Blog (78)
- Cheat Sheets (10)
  - Shells (1)
  - SQL Injection (7)
- Contact (2)
- Site News (3)
- Tools (17)
  - Audit (3)
  - Misc (7)
  - User Enumeration (4)
  - Web Shells (3)
- Uncategorized (3)
- Yaptest (15)
  - Front End (1)
  - Installing (2)
  - Overview (2)
  - Using (8)

RSS Feed

unix-privesc-check

Unix-privesc-checker is a script that runs on Unix systems (tested on Solaris 9, HPUX 11, Various Linuxes, FreeBSD 6.2). It tries to find misconfigurations that could allow local unprivilged users to escalate privileges to other users or to access local apps (e.g. databases).

It is written as a single shell script so it can be easily uploaded and run (as opposed to un-tarred, compiled and installed). It can run either as a normal user or as root (obviously it does a better job when running as root because it can read more files).

Download

unix-privesc-check v1.4 can be downloaded here. (Version 1.1 is here if you still need it).

Update: The google code SVN is more up to date.

Usage

The download is gzip'd, so gunzip it. Upload it to the server you're auditing / pentesting then just run it:

```
$ ./unix-privesc-check > output.txt
```

The output's a bit messy (it's hard to be neat with shell scripts), so you're probably best to save the output and search it for the word 'WARNING'. If you don't see the word 'WARNING' then the script didn't find anything. Example:

```
$ ./unix-privesc-check
```

```
Starting unix-privesc-check v1.0 ( http://pentestmonkey.net/tools/unix-privesc-check )

This script checks file permissions and other settings that could allow
local users to escalate privileges.

Use of this script is only permitted on systems which you have been granted
legal permission to perform a security assessment of.  Apart from this
condition the GPL v2 applies.

Search the output below for the word 'WARNING'.  If you don't see it then
this script didn't find any problems.

Assuming the OS is: linux

###########################################
Checking if external authentication is allowed in /etc/passwd
###########################################
No +:... line found in /etc/passwd

###########################################
Checking nsswitch.conf for addition authentication methods
###########################################
Neither LDAP nor NIS are used for authentication
```

```
... lots more output ...
```

What’s the Intended usage of user-privesc-checker?

It's intended to be run by security auditors and pentetration testers against systems they have been engaged to assess, and also by system admnisitrators who want to check for "obvious" misconfigurations. It can even be run as a cron job so you can check regularly for misconfigurations that might be introduced.

I wanted to write something that was at least partially useful to pentetration testers when they gained access to a low-privilege account and wanted to escalate privileges. There are lots of things that pentesters will check in this situation and one of the most tedious to check is weak file permissions – this of often one of the most fruitful, though, so there's no avoiding it.

Disclaimer: Running this script alone isn't a substitute for proper audit (e.g. following one of the NSA's excellent configuration guides). There are lots of possibilities for escalation that are just too hard to audit using a script. This script is intended to be a shortcut, not a replacement for a proper audit. See the "Limitations" section below for lots of examples of areas not covered by this script.

So this is a Unix Audit Script?

Not in the traditional sense. "Unix Audit" means different things to different people. I understand it to mean checking a whole array of configuration settings including:

- Security patches (i.e. that they've been applied)
- Cracking passwords to check for weak ones
- IP Stack configuration (no unnecessary IPv6, no IP Forwarding, etc.)
- Weak file permissions (reading sensitive data, modifiying sensitive files)
- Configuration of local applications (reviewing sshd_config, httpd.conf)
- Other best-practise stuff (remote logging, no insecure protocols, paranoid mount options)

So, no it's not an audit script in this sense. It doesn't set out to do all these things. It checks for a subset of these which relate directly to privilege escalation. It focusses mainly on generic techniques: common misconfigurations and weak file permissions. It doesn't check for missing patches, however this is difficult to check "on-box" using a single shell script. Checkout exploit-suggester if you're interested in doing this "off-box"..

Why Another Auditing Script?

There are lots of scripts out there that will perform a local security audit for you (Tiger and LSAT are good examples). Some hoover data so you can analyse it offline, others will analyse it too and present you a nice report. These have their place and I'll continue to use them. However…

I haven't found one that gives me a quick list of obvious attack vectors which is written as a single shell script (I hate shell scripts too, but it I wanted a script that would run on virtually any Unix system).

I therefore decided to write unix-privesc-audit to focus on finding misconfigurations that can actually be exploited as opposed to finding all the usual best-practise stuff.

Some Vulnerabilities Introduced Through Weak File Permissions

Below is a list of the checks performed by the script. Note that whenever it checks file permissions, it also checks the permissions on the parent directories. When it finds a group-writable file or directory it only flags an issue if that group has more than one non-root member.

Writable Home Directories

If you can write to someone's home directory, you could add a .rhosts file or .ssh/authorized_keys file and log in right away; or alter one of the login scripts (e.g. .bash_profile) and have them create an SUID shell when they log in. There are lots of problems if home directories are writable.

The script flags a warning if any home directories are writable by anyone other than the owner or root.

Readable /etc/shadow

Not so common these days, but could allow you to read password hashes, crack them then log in as other users.

This script checks if the shadow file is readable by non-root users. If it can read the shadow file, it also performs some other checks (see below).

Weak Permissions On Cron Jobs

Cron jobs are normally listed in /etc/crontab and /var/spool/cron/crontabs/. Cron jobs can be run by any user. This script check if cron jobs run programs that can be modified by users other than root and the user the job runs as.

Writable Configuration Files

Programs that are run as root are listed in lots of files (/etc/init.d/*, /etc/inetd.conf, /etc/xinetd.d/*, etc.) If any of these files are writable by non-root users, this script will flag a warning.

Writable Device Files

This script checks that device files corresponding to currently mounted file systems (e.g. /dev/sda1) aren't writable. I doubt this happens very often to be honest, but it doesn't hurt to check.

Readable Files In Home Directories

There can be lots of interesting things in home directories, but this script checks for files that contain passwords (.netrc, .my.cnf) and ACLs (.rhosts, .ssh/*). If readable these can provide a way access local applications or other local user accounts.

Running Processes Correspond To Writable Programs

This script does a “ps” listing, attempts to determine the full path of each program running and check if it can be modified by anyone other than the user it’s running as and root.

Other Stuff Not Related With File Permissions

The script also performs a couple of other checks related to privilege escalation, but not related to file permissions:

Sudo Configuration

Sudo is one of the most obvious ways to escalate privileges if it’s enabled. Sometimes only certain commands can be run, sometimes any command can be run.

If /etc/sudoers is readable, this script checks if it’s being used, lists which users can use sudo and which ones can use it without a password.

Accounts with no Password

If /etc/shadow is readable, the script lists any accounts without passwords.

How useful is unix-privesc-check in practise?

It depends largely on the base OS and the amount of configuration an administrator has done (more configuration = more chance of mistakes). If you run it against a fairly modern OS (e.g. Linux, Solaris 9/10) that hasn’t had much configuration done, then you’re not going to find much. These OSs have faily secure file permissions by default. However, if you run it against Solaris 8 or against a system that has been running for a couple of years and had a lot of configuration done or 3rd party apps installed, then you’ll probably find quite a bit.

I’m ashamed to say that I found a couple of serious misconfigurations in my own Linux box when I ran this script!

Limitations

Currently only the stuff above is checked. None of the other “traditional audit” stuff is checked. There are also some obvious privilege escalation tactics regarding file permissions which are too hard to script up (for me at least). This list acts as a list of limitations / inspiration for manual testing / working TODO list:

- Actually reading shell scripts, finding that they call then checking the file permissions on called programs (e.g. seeing /etc/init.d scripts call anything in an insecure way, checking shell scripts that are run a login time)
- Checking the PATHs for users then looking for insecure programs within those directories
- Polling “ps” to identify processes that don’t run very often and checking that the program being run has secure file permissions.
- Checking for non-standard programs with the SUID bit set
- Checking arguments of currently running processes (e.g. if a process runs as root and had “/dir/somefile.conf” as an argument it would be worth checking the perms on that file).
- It doesn’t parse shell script names, perl script names or any other kind of script name from the process listing. It just checks /bin/sh and /usr/bin/perl normally. This is a pretty big limitation actually at the moment. It a process is called “/bin/sh /tmp/dodgy-n-writable.sh” it’s probably worth investigating).
- It doesn’t check the permissions of shared object files for each running process. This info is availalable in /proc/pid/maps, /proc/pid/smaps on Linux at least, so this feature may follow shorty.
- It doesn’t check the permissions on open files. Again this info is available on Linux at least in /proc/pid/fd/, so this feature may follow shortly.
- Doesn’t report /etc/hosts.equiv trust relationships.
- No checks for NFS mounts / exports. These are a common source of insecurity.
- Doesn’t check stuff run from inittab.

Yeah, lots and lots of limitations, so make sure you do a manual audit too. I hope this script saves you some time, though.

Tags: audit, pentest, tool, unixprivesccheck

Posted in Audit