

Unix System Administration

Chris Schenk

Lecture 22 – Tuesday Apr 10

CSCI 4113, Spring 2007

The CSEL Hack

- How did we find out that our machines were hacked?
 - Only after half a day of wondering we found out
 - csel.cs was completely locked up around 10:45am
 - Rebooted back into a normal state
 - lesc.cs was still running, initial check looked ok
 - Turns out it was easily broken into
 - Root login was used from romania
- How can we detect potential problems on the fly?
 - And prevent this sort of post-incident discovery?

Logs are your Best Friend

- Without logging, how could you track anything?
 - Windows has it, anyone know where?
- Understanding your logging is your first step
 - `/var/log` contains most everything needed for forensics
 - NEVER ditch your logs when trying to track down a problem!
- Where is the first place you look for bad things that may happen on your system?
 - **auth** and **authpriv** logging facilities
 - Check `syslog.conf` to see where these are logged
 - `/var/log/auth.log` for Ubuntu

Syslog

- Main config file `/etc/syslog.conf`
 - Configures where all **facilities** and **levels** go
- Facilities are listed first, followed by level
 - Can be comma-separated
 - `auth.*`, `daemon.info`
- Destination files or programs come after
 - `auth.* /var/log/auth.log`
- Some flexibility for managing logs
 - `syslog-ng` I hear is better (haven't tried it though)

Logwatch – A Log Summary

- Simply a system log analyzer and reporter
- Available on most every linux machine I've seen
 - Installed by default on Fedora, Ubuntu
- Default install works right out of the box usually
 - Emails root once a day (around 4am) with a log summary
- Shows information from many different locations
 - httpd, PAM utilities, disk usage, SSHD errors

Proactive Log Scanning

- Logwatch can be configured to scan more frequently
 - But constant emails that large can be annoying
 - Tends to make people ignore the constant emails
 - But sometimes many emails are necessary
- How can we proactively scan our log files?
 - How can we look at each line logged as it arrives?
 - What are we looking for?
 - Successful logins
 - Multiple attempts with incorrect logins
 - Which log files are important?

SSHDFilter Internals

- SSHDFilter spawns SSHD
 - Monitors all syslog entries created from SSHD
 - How can we see to what facility SSHD logs?
 - Logs a few extra things itself
 - “Chanced x.x.x.x, tries=3” etc
- We could edit SSHDFilter to do extra work
 - But I'm not a fan of dirtying things like that
- We need another way to read data from syslog
 - Enter named pipes!
 - `sudo mkfifo /var/log/auth.pipe`

Named Pipes

- Look like a normal file
 - Like most everything else
- Notice the name of the command, 'fifo'
 - First-in, first-out reading and writing
- We configure syslog to write to our named pipe
 - `auth.* , authpriv.* | /var/log/auth.pipe`
- Now we can read this pipe like a regular file
 - And it contains log entries real-time

Perl

- **Pathologically Eclectic Rubbish Lister**
- Perl Intro: <http://perldoc.perl.org/perlintro.html>
- Fantastic parsing capabilities
 - Among lots of cryptic syntax and shortcuts
- Regular expressions built-into the language
 - Sit between forward-slashes: `/[Mm]atched [Tt]ext/`
- Simple file input and output
 - Syntax similar to IO redirection on command line

Perl Basics

- Variables
 - `@var` – array, `$var` – scalar, `%var` – hashmap
- Special variables `@_` and `$_`
 - Default variables for many functions
 - Reading a file puts each line into `$_`
 - Calling 'print' without a variable prints `$_`
 - Subroutines put arguments into array `@_`
- Open and read file syntax
 - `open(HANDLE, "</path/to/file_to_read.txt");`
 - `while(<HANDLE>) { }`

Perl Regular Expressions

- Are the bomb
- Can match on variables or on `$_`
 - `if(/some regex/) { ... } -- uses $_`
 - `if($var =~ /regex/) { ... } -- uses $var`
- Can capture parts of the regex into variables
 - `if($var =~ /re(ge)x/ { print $1; } -- prints 'ge'`
- Can also ignore case with options at the end
 - `/another ReGeX/i -- 'i' ignores case`
- Do straight substitution
 - `'s/shit to replace/new shit to replace it with/'`