

# Unix System Administration

Chris Schenk

Lecture 23 – Thursday Apr 10

CSCI 4113, Spring 2008

# 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
  - We need a smart script to parse our log file

# 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
- ...And again, extremely f'ing cryptic at times

# Totally Random Tangent

- Print “just another perl hacker,” to the screen
  - [http://en.wikipedia.org/wiki/Just\\_another\\_Perl\\_hacker](http://en.wikipedia.org/wiki/Just_another_Perl_hacker)
- Obfuscated Perl Contest
  - [http://www.foo.be/docs/tpj/issues/vol3\\_2/tpj0302-0012.html](http://www.foo.be/docs/tpj/issues/vol3_2/tpj0302-0012.html)
- Obfuscated C Contest
  - [http://en.wikipedia.org/wiki/The\\_International\\_Obfuscated\\_C\\_Code\\_Contest](http://en.wikipedia.org/wiki/The_International_Obfuscated_C_Code_Contest)
- Quines
  - <http://www.nyx.net/~gthompso/quine.htm>

# Our Use of Perl

- Perl was developed specifically for sysadmin
  - Lots of packages and libraries out there
- Our use will include:
  - Regular Expressions for extensive text parsing
    - Both in our auth.log and in other files
  - Email for notification of an event
  - MySQL database interface (DBI)
    - For both inserts and queries of database information
    - Support for prepared statements (very important!)

# 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/'`

# Perl + MACs

- Message Authentication Codes
  - Used to authenticate plain-text data
- 'emailreport' has no authentication
  - Actually a security hole, information leak
- MACs allow us to send plain text and a tag
  - Tag is verified at the other end
- MACs use a shared key on both ends
  - 'rndc' for BIND uses this exact concept

# Perl + MACs (cont)

- Use the `Digest::HMAC_SHA1` module (after installing `libdigest-hmac-perl`)
  - `Use Digest::HMAC_SHA1;`
- Create a new HMAC object to create a tag
  - `my $hmac = Digest::HMAC_SHA1->new("shared key");`
- Add data to be digested
  - `$hmac->add("just another perl hacker,");`  
`my $digest = $hmac->b64digest();`
- Now, on the other end, verify tag before performing any tasks (such as sending email)!

# Perl + MySQL

- Use the DBI module (after installing `libdbd-mysql-perl` package in Ubuntu)
  - `use DBI;`  
`use DBD::mysql;`
- Define and set your variables
  - `my $user = "monitor";`  
`my $pass = "wearewatchingyou";`  
`my $db = "login_monitor";`  
`my $host = "localhost";`
- Create a connect string and connect
  - `$cs = "dbi:mysql:$db:$host:3306";`  
`$conn = DBI->connect($cs, $user, $pass);`

# Perl + MySQL (cont)

- **Prepare a statement and execute**

```
- $stmt = $conn->prepare("SELECT * from  
logins");  
$stmt->execute();
```

- **Bind columns (results) and print them out**

```
- $stmt->bind_columns(undef, \$id, \$username,  
\$ip, \$time);  
while($stmt->fetch()) {  
    print "$username, $ip, $time\n";  
}
```

- **Close the connection**

```
- $conn->disconnect();
```