

Day 15: Security & Performance

perlsec

Benchmark module
(if you like)

Homework Review

Security


Think Like a Hacker

- How can I affect this script?
 - Outside data (@ARGV, files, environment, ...)
 - Perfectly crafted
 - Too much, too little, malformed, out of range, ...
 - No limits to creativity
- What can I cause the script to do?
 - Crash
 - Change the system
 - Change the results
 - `eval()`, `s///e`, `s///ee`

What Is Wrong With This Script?

```
if (scalar(@ARGV) != 1) {  
    die "$0: need filename argument\n";  
}  
  
my $filename = $ARGV[0];  
  
open(my $fh, $filename) or die "...: $!\n";  
my @lines = <$fh> or die "...: $!\n";  
close($fh) or die "...: $!\n";  
  
for (my $i = 0; $i < scalar(@lines); $i++) {  
    print "$i: $lines[$i]";  
}
```

'rm -f *.txt |'



Problems With open()

- Filename argument is more than filename

<code>open(FH, "< foo")</code>	<i>read</i>
<code>open(FH, "> foo")</code>	<i>create/(over)write</i>
<code>open(FH, ">> foo")</code>	<i>create/append</i>
<code>open(FH, "foo ")</code>	<i>run command, read output (like `foo`)</i>

~~`open(FH, $unchecked_variable)`~~

Safer open()

- *Always* use three-argument version, even for reads

```
if (scalar(@ARGV) != 1) {  
    die "$0: need filename argument\n";  
}  
  
my $fname = $ARGV[0];  
  
open(my $fh, '<', $fname) or die "...: $!\n";  
my @lines = <$fh> or die "...: $!\n";  
close($fh) or die "...: $!\n";
```

Problems With `system()`

- Purpose is to invoke system commands...
- May invoke shell and hence shell interpretation

```
system("curl $url");
```

```
URL; rm -f ...
```

```
--silent -V; rm -f ...
```

```
--upload-file /etc/passwd URL
```

```
system("... $variable_name ...");
```


Safer system()

- Use separate arguments whenever possible
- If you *must* use shell characters, validate everything

```
system('curl', '--silent', $url);  
# what if $url = '-V; rm -f ...'?  
% curl --silent '-V; rm -f ...'
```

A Little Bit of Help: **use taint**

- Perl will *try to help* you identify dangerous values
- Marks all data that comes from “outside”:
 - Command-line arguments
 - Data from a filehandle (including STDIN)
 - Environment variables
 - Results of certain system calls (e.g., readlink)
- Passed to all copies of tainted data
- Cannot use tainted data directly to:
 - Modify file or directory
 - Run a command
- **Does *NOT* automatically make a script secure!!!!!!!!!!**

Taint Example

```
use taint;
```

```
my $date = $ARGV[0];          # $date is tainted  
my $filename1 = "data-$date.txt"; # tainted
```

```
# next line would cause Perl to exit script  
open(my $fh, '>', $filename1) or die "...";
```

```
(my $ok_date = $date) =~ s/\W+/_/g; # ok now  
my $filename2 = "data-$ok_date.txt"; # ok
```

```
open(my $fh, '>', $filename2) or die "...";
```

***Just because you're paranoid doesn't
mean they're not out to get you***

Performance

CPU Cycles Are Cheap...

- Your time versus the computer's time
 - 1 hour to save 50 ms/run — worth it?
 - 1 hour to save 1 hour/run — worth it?
- Moore's Law: next month's CPU will be 10% faster*
- Waste the computer's time, not yours
- If you need a LOT of computing power, use CHTC

* horribly inaccurate representation of Moore's actual statement...

But...

***We should* forget about **small** efficiencies,
say about 97% of the time:
premature optimization is the root of all evil.**

— Donald Knuth, 1974

The Other 3% of the Time...

Easy Metrics

- Use the shell's **time** command

```
% perl homework-15.pl input-15.txt
Done!
% time perl homework-15.pl input-15.txt
Done!

real      0m3.928s
user      0m3.907s
sys       0m0.012s
%
```

More Detailed Metrics

- Use `Time::HiRes` to measure “wall” time (not CPU)
- Start with just a few
- Think binary search

```
use Time::HiRes qw/time/;
my $t_start = time();
initialize();
do_something();
my $t_mid = time();
do_something_else();
wrap_up();
my $t_end = time();
printf "Part 1: %.1f s\n", $t_mid - $t_start;
printf "Part 2: %.1f s\n", $t_end - $t_mid;
```

Very Detailed Metrics

- Use **Benchmark**
- Good for comparing alternatives directly

```
use Benchmark qw/cmpthese/;  
my $x = 3;  
cmpthese( -1,  
          { 'a' => sub{$x * $x},  
            'b' => sub{$x ** 2}, } );
```

	Rate	b	a
b	4745709/s	--	-12%
a	5420446/s	14%	--

Memory Is Cheap...

- ... and fast
- ... but limited
- Running out of memory is bad... but hard to do

```
open(my $fh, '<', $file) or die "...";
```

```
# Option 1: Read ALL lines into memory
```

```
my @lines = <$fh>;
```

```
foreach my $line (@lines);
```

```
# Option 2: Only 1 line in memory at a time
```

```
while (my $line = <$fh>) { ... }
```

Disk Is Cheap...

- ... and huge
- ... but slow
- Do as little I/O as is reasonable
- Also watch out for too many open filehandles

```
open(my $fh, '<', $file) or die "...";  
my @lines = <$fh>;  
close($fh);  
...  
  
my @lines_again = @lines;
```

Things to Avoid

- CPU
 - Inefficient algorithms
 - Needless repetition
 - Expensive operations inside tight loops
- Memory
 - Too much stuff in memory
 - Needless copies
- Disk
 - Needless reads/writes
 - Many small open/close operations
- **ALWAYS USE METRICS!!!!**

Homework

Fix Me!

- Homework provides a simple script
- Likely contains security, performance, and correctness problems
- Make it better!
- *Extra:* Give before/after performance metrics!