Day 15: Security & Performance

perlsec
Benchmark module
(if you like)
Homework Review
Security
What Is Wrong With This Script?

```perl
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 foo |
Problems With open()

- Filename argument is more than filename

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

```perl
open(FH, $variable_name)
```
Safer open()

• Always use three-argument version, even for reads

```perl
if (scalar(@ARGV) != 1) {
    die "$0: need filename argument
";
}

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

```perl
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

```perl
system('curl', '--silent', $url);

# what if $url = '-V; rm -f ...'?
% curl --silent '-V; rm -f ...
```
A Little Bit of Help:  

- 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!!!!!!!!!!
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 "...";
# Levels of Security Risk

<table>
<thead>
<tr>
<th>Script</th>
<th>Environment</th>
<th>Risk</th>
</tr>
</thead>
<tbody>
<tr>
<td>Not a service</td>
<td>Only you run</td>
<td>Low (but add safety checks)</td>
</tr>
<tr>
<td>Not privileged</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Not a service</td>
<td>Anyone else runs</td>
<td>Medium</td>
</tr>
<tr>
<td>Not privileged</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Service or Privileged</td>
<td>Anywhere</td>
<td>High</td>
</tr>
<tr>
<td>Privileged</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Performance
CPU Cycles Are Cheap

- Your time versus the computer’s time
  - 600,000 ms 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*
- Scripting: Waste the computer’s time, not yours
- If you need a LOT of computing power, use Condor

* horribly inaccurate representation of Moore’s actual statement…
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

```bash
% ./hw-15.pl hw-15.txt
Done!
% time ./hw-15.pl hw-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

```perl
use Time::HiRes qw/time/;
my $time_start = time();
initialize();
do_something();
my $time_checkpoint_1 = time();
do_something_else();
wrap_up();
my $time_end = time();
my $blk1 = $time_checkpoint_1 - $time_start;
```
Very Detailed Metrics

- Use `Benchmark`
- Good for comparing alternatives directly

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

<table>
<thead>
<tr>
<th>Rate</th>
<th>b</th>
<th>a</th>
</tr>
</thead>
<tbody>
<tr>
<td>b 4745709/s</td>
<td>--</td>
<td>-12%</td>
</tr>
<tr>
<td>a 5420446/s</td>
<td>14%</td>
<td>--</td>
</tr>
</tbody>
</table>
Memory Is Cheap…

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

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

```perl
open(my $fh, '<', $file) or die "...";
my @lines = <$fh>;
close($fh);
...
my @lines_again = @lines;
```
Things To Watch Out For

- 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
• May contain security and/or performance issues
• Make it better!
• *Extra*: Give before/after performance metrics!