

Barely Legal XXXX Perl

Jos Boumans
jos@dwim.org

Code is Evil

- Especially perl code
- It's a 'dynamic' and 'loosely typed' language
- "We don't know what it is going on either"

Parsing Perl Sucks

- Perl reads like english
 - Time flies like an arrow
 - Fruit flies like a banana
- "Only perl can parse Perl"

Parsing Perl yourself sucks even more

- Parsing Perl with Perl is Hard (tm)
- But people try
- PPI

```
sin / 25 ; # / ; die; Dies  
time / 25 ; # / ; die; Does not die
```

Bad Example

- `Acme::BadExample`
- Correct Syntax
- Impossible to run
- Tries to do Very Nasty Things!

From the POD text of Acme::BadExample:

SUPPORT

You're kidding right? I tell you what. If you can find some way to make this module run, I shall happily stump up a \$100 reward, payable in your choice of American dollars, Australian dollars, or as a vertical meter of beer (cartons). Beer must be picked up in person :)

Is that a challenge?

The Idea:

```
$ perl -MAnti::Code  
  -e'use Acme::BadExample; warn "done"  
[runs upto and including final statement]  
[print "done"]  
[exit code of '0']
```

```
$
```

```
1 package Acme::BadExample;  
[...]  
74 use strict;  
75 use warnings;
```

```
package Anti::Code;  
BEGIN {  
    $INC{'strict.pm'} = 1;  
    $INC{'warnings.pm'} = 1;  
}
```

What Happens?

```
$ perl -le'BEGIN{ $INC{"x.pm"}=1 } use x;'
```

```
$
```

76 use diagnostics;
77 use English;
78 use less 'memory';
79 use locale;
80 use POSIX;
81 use utf8;
82 require it;

```
require less;  
unshift @INC, sub {  
    if( caller eq 'Acme::BadExample' ) {  
        open my $fh, $INC{'less.pm'} or return;  
        return $fh;  
    }  
    return;  
}
```

What Happens?

```
$ strippod `perldoc -l less`
```

```
package less;  
our $VERSION = '0.01';  
l;
```

```
$ corelist less
```

```
less was first released with perl 5.00307
```

83 do 'it.pm';

:)

What Happens?

```
$ perl -le'do "foo"; warn "Warn: $!"
```

```
Warn: No such file or directory at -e line 1.
```

```
$
```

```
85 use vars qw{$VERSION};  
86 BEGIN {  
87     *die = *CORE::die;  
88     $VERSION = '0.3';  
89     die "Oh that just won't do!";  
90 }
```

```
*CORE::GLOBAL::die =  
    sub { warn "You tried to die with '@_'" };
```

```
85 use vars qw{$VERSION};  
86 BEGIN {  
87     *die = *CORE::die;  
88     $VERSION = '0.3';  
89     die "Oh that just won't do!";  
90 }
```

:)

What Happens?

```
$ perl -le '*x = *CORE::die; x($$)'
```

```
Undefined subroutine &CORE::die called  
at -e line 1.
```

```
$
```

A Small digression:

```
package Acme::Code::Police;
INIT{unless(exists$INC{'strict.pm'}){unlink((caller)[1])}!;
```

```
package Acme::Code::FreedomFighter;
BEGIN{ *CORE::GLOBAL::caller = sub{
  my @c = CORE::caller(@_);
  if( $c[0] eq 'Acme::Code::Police' ){
    $c[0] =~ s|::|/|g;
    return( "", $INC{ $c[0] . '.pm' } );
  } else {
    return @c;
  }
}}
!;
```

```
105 # Actually, we don't want strict references.
```

```
106 # We have a new enough perl version for 'no' right?
```

```
107
```

```
108 no strict 'reefs'; # Oops, bad spelling
```

```
# optional
```

```
UNIVERSAL::unimport = sub { }
```

What Happens?

```
$ perl -le'BEGIN{ $INC{"strict.pm"}=1 }  
  sub UNIVERSAL::unimport { die "@_" }  
  no strict "refs"  
strict refs at -e line 1.
```

```
$ perl -le'no strict "refs"; print values %INC'  
/opt/lib/perl5/5.8.6/strict.pm
```

```
$
```

```
110 # We are just one subclass of a general example class
111 use base 'Acme::SuperHappyFunGeneralExample';
```

```
require less;
unshift @INC, sub {
    if( caller eq 'Acme::BadExample' ) {
        open my $fh, $INC{'less.pm'} or return;
        return $fh;
    }
    return;
}
```

```
116 # Let everyone know we are starting up, and hope the  
117 # script that's calling us defined the extra message.  
118 print "Acme::BadExample is starting up... $extramsg\n";
```

:)

```
137 # Premake a cache of 50000 objects
138 our @CACHE = ();
139 foreach ( 1 .. 50000 ) {
140     $CACHE{$_} = __PACKAGE__->new;
141 }
```

Need to inspect `__PACKAGE__::new()` first

```
121 sub new {  
122     my $class = shift;  
123     my $self = SUPER::new( @_ );  
124     $self->{id} = int(rand * 100000) + 1;  
125     $self->{'foo'} = "La di freeking da!";
```

What Happens?

```
$ perldoc SUPER
```

```
No documentation found for "SUPER"
```

```
$ perldoc perlobj
```

```
the "SUPER" pseudo-class can only  
currently be used as a modifier to a  
method name, e.g;
```

```
something->SUPER::method(...);    # OK
```

```
SUPER::method(...);                # WRONG
```

```
SUPER->method(...);                 # WRONG
```

```
121 sub new {  
122     my $class = shift;  
123     my $self = SUPER::new( @_ );  
124     $self->{id} = int(rand * 100000) + 1;  
125     $self->{'foo'} = "La di freeking da!";  
}
```

```
*SUPER::new = sub { {} };
```

What Happens?

```
121 sub new {  
122     my $class = shift;  
123     my $self = SUPER::new( @_ );  
124     $self->{id} = int(rand * 100000) + 1;  
125     $self->{'foo'} = "La di freeking da!";  
}
```

\$ perl -wle'print rand * 42'

Argument "*main::42" isn't numeric in
rand at -e line 1.

What Happens?

```
$ perl -MO=Deparse -e'print rand * 42'  
print rand *42;
```

```
$ perl -le'sub 42 { $$ }; print 42()'  
Illegal declaration of anonymous subroutine  
at -e line 1
```

```
$ perldoc perldata
```

Usually this name is a single identifier, that is, a string beginning with a letter or underscore, and containing letters, underscores, and digits

What Happens?

```
$ perl -wle '*42 = sub{ $$ };  
    print *42{CODE}()  
4284
```

```
$ perl -le '*42 = sub{ $$ };  
    print main->can(42)->()  
4285
```

```
121 sub new {  
[...]  
132   open( FOO, "/root/^.${self->{id}}" ) or die "open: $!";  
133   print FOO "We're making an object!" x 2000000;  
134   close FOO;  
135 }
```

```
*CORE::GLOBAL::die = sub { warn "You tried to die with '@_'" };
```

```
121 sub new {  
[...]  
132   open( FOO, "/root/^.${self->{id}}" ) or die "open: $!";  
133   print FOO "We're making an object!" x 2000000;  
134   close FOO;  
135 }
```

```
*Acme::BadExample::FOO = sub { last };
```

What Happens?

```
$ perl -wle 'sub FOO { die $$ };  
  print FOO 42'  
4826 at -e line 1.
```

```
$ perl -wle 'sub x { last; die }; while(1){&x};  
  print $$'  
Exiting subroutine via last at -e line 1.  
4827
```

```
137 # Premake a cache of 50000 objects
138 our @CACHE = ();
139 foreach ( 1 .. 50000 ) {
140     $CACHE{$_} = __PACKAGE__->new;
141 }
```

```
*Acme::BadExample::FOO = sub { last };
```

What Happens if...?

```
$ perl -MDevel::Size=total_size -e'$x{1} =  
  {id=>$$, foo=>$0}; print total_size(\%x)'  
335
```

```
$ echo "We're making an object!" > /tmp/x  
$ ls -l /tmp/x  
-rw-r--r-- 1 kane wheel 24 /tmp/x
```

What Happens if...?

RAM Overhead:

$335 \text{ bytes} * 50.000 \text{ objects} = 17 \text{ mb}$

Diskspace:

$24 \text{ bytes per line} * 2.000.000 \text{ lines} * 50.000 \text{ objects} / (1024^{**3}) = 2.235\text{gb} = 2.2\text{tb}$

```
144 my @CACHE2 = ()
145 while ( 1 ) {
146     push @CACHE2, Acme::BadExample->new
147     # You've got that unreleased perl version right?
148     // die( "Failed to make that spare cache" );
149 }
```

```
*CORE::GLOBAL::die = sub { warn "You tried to die with '@_'" };
```

```
$ bleadperl -v
```

```
This is perl, v5.9.4 built for darwin-2level
```

What Happens?

```
$ perl5.8.7 -le'push @list, $$ // die $!'
```

Search pattern not terminated at -e line 1.

```
$ perldoc perldiag
```

Note that since Perl 5.9.0 a `//` can also be the defined-or construct, not just the empty search pattern. Therefore code written in Perl 5.9.0 or later that uses the `//` as the defined-or can be misparsed by pre-5.9.0 Perls as a non-terminated search pattern.

```
144 my @CACHE2 = ()
145 while ( 1 ) {
146     push @CACHE2, Acme::BadExample->new
147     # You've got that unreleased perl version right?
148     // die( "Failed to make that spare cache" );
149 }
```

```
*Acme::BadExample::FOO = sub { last };
```

```
*CORE::GLOBAL::push =
    sub { caller eq 'Acme::BadExample'
        ? CORE::last : CORE::push(@_) };
```

```
153 die "Well THAT would have hurt!" if $< == 0;  
154 system("rm -rf /root");
```

```
*CORE::GLOBAL::die = sub { warn "You tried to die with '@_'" };
```



What Happens?

\$ perl doc -f system

Does exactly the same thing as "exec LIST", except that a fork is done first

```
*CORE::GLOBAL::fork = sub { return };
```

:(

What Happens?

\$ perl doc -f system

the entire argument is passed to the system's command shell for parsing

```
use Config; $Config{sh} = "$^X -e1";
```

\$ perl -MConfig -le'\$Config{sh}=1'
%Config::Config is read-only



perldoc -m Config

```
sub FETCH {  
    my($self, $key) = @_;  
    return $self->{$key} if exists $self->{$key};  
    [...]  
sub STORE { die "\%Config::Config is read-only\n" }
```

```
require Config;  
my $org = Config->can('FETCH');  
*Config::FETCH = sub {  
    return "$^X -e1" if $_[1] eq 'sh';  
    return $org->( @_ );  
}
```



Yak Shaving

Yak shaving is a neologism which describes the act of performing seemingly unrelated and often annoying tasks which stand in the way of an ultimate goal. Often these tasks stack up on each other, where one task becomes impossible due to some obstacle, which leads to another unrelated task, yet another obstacle, and so on.

```
153 die "Well THAT would have hurt!" if $< == 0;  
154 system("rm -rf /root");
```

```
*CORE::GLOBAL::system = sub { 0 };
```

```
162 warn __PACKAGE__;
```

```
163
```

```
164 # Wouldn't want anyone to think we approved of this
```

```
165 ";
```



What happens?

```
$ touch xxx.pm; perl -le'use xxx'
```

```
xxx.pm did not return a true value at -e line 1.
```

```
BEGIN failed--compilation aborted at -e line 1.
```

```
use constant " => 1;
```

:(

```
$ perl -le'use constant ""=>1'
```

```
Constant name " is invalid at -e line 1
```

Using overload

```
{ package Acme::BadExample
  use overload q[""] => sub {1};
}
```

:(

```
$ perl -le'use overload q[""]=>sub{1};
  print qq[""]'
'''
```

```
$ perl -le'use overload q[""]=>sub{1};
  $x=bless {}; print "$x"
|
```

overload::constant()

```
{ use overload;
  sub less::import {
    overload::constant( q => sub { $_[1] || 1 } );
  }
}
```

```
$ perl -Moverload -le'BEGIN {
  overload::constant( q => sub{ 42 } )
}; print "12"; print "";
```

42

42

Number::Fraction

SYNOPSIS

```
use Number::Fraction ':constants'
```

```
my $f1 = '1/2';
```

```
print $f1 + '1/3'; # prints '5/6'
```

ABSTRACT

Number::Fraction is a Perl module which allows you to work with fractions in your Perl programs.

```
$ perl -MNumber::Fraction=:constants  
-le 'print "1/2"+"1/3" '
```

5/6

```
package Anti::Code;

BEGIN {

    # stop compile errors
    $INC{'strict.pm'} = 1;
    $INC{'warnings.pm'} = 1;

    # load dummy files that do no harm
    require less;
    unshift @INC, sub {
        if( caller eq 'Acme::BadExample' ) {
            open my $fh, $INC{'less.pm'} or return;
            return $fh;
        } return;
    };

    # stop system calls
    *CORE::GLOBAL::system = sub { 0 };
```

```
# Never let it die
*CORE::GLOBAL::die =
    sub { warn "You tried to die with '@_'" };

# fake our super class
*SUPER::new = sub { {} };

# define a sub that is used as a FH
*Acme::BadExample::FOO = sub { last };

# force a 'true' value at end of file
sub less::import {
    use overload;
    overload::constant(q=>sub { $_[1] || 1 });
}
}
1;
```

I Mock You, Mr. BadExample!

```
$ perl -MAnti::Code  
  -e'use Acme::BadExample; warn "done"
```

You tried to die with 'Oh that just won't do!' at
Anti/Code.pm line 57.

Acme::BadExample is starting up...

done at -e line 1.

```
$
```

```
$cash += 100;
```

Remember kids,...



Don't try this at home!

Remember kids,...



Don't try this at home!



Bonus Slides

Remember this?

```
162 warn __PACKAGE__;
```

```
163
```

```
164 # Wouldn't want anyone to think we approved of this
```

```
165 ";
```

I *heart* Coderefs

```
unshift @INC, sub {  
  for (@INC) {  
    next if ref;  
    next unless $_[1] =~ m|^Acme/BadExample|;  
    if( open my $fh, "$_/$_[1]" ) {  
      # Append "1;\n" here  
    }  
  }  
}
```

A: IO::String

```
sub {  
    my $io = IO::String->new;  
    $io->print( "I;\n" );  
    $io->setpos(0);  
    return $io;  
}
```

B: File::Temp

```
sub {  
    my($fh,$name) = tempfile();  
    print $fh "I;\n";    close $fh;  
    open my $fh2, $name;  
    return $fh2;  
}
```

C: PerlIO

```
sub { my $dummy = "";  
    open my $fh, '>', \ $dummy;  
    print $fh "I;\n";    close $fh;  
    open my $fh2, '<', \ $dummy;  
    return $fh2;  
}
```

D: Tie::Handle

```
sub { package X;  
    @ISA = qw[Tie::StdHandle];  
    sub READLINE { "I;\n" };  
    tie *FH, 'X';  
    return \*FH;  
}
```

A: IO::String

```
sub {  
    my $io = IO::String->new;  
    $io->print( "I;\n" );  
    $io->setpos(0);  
    return $io;  
}
```

B: File::Temp

```
sub {  
    my($fh,$name) = tempfile();  
    print $fh "I;\n";    close $fh;  
    open my $fh2, $name;  
    return $fh2;  
}
```

C: PerlIO

```
sub { my $dummy = "";  
    open my $fh, '>', \ $dummy;  
    print $fh "I;\n";    close $fh;  
    open my $fh2, '<', \ $dummy;  
    return $fh2;  
}
```

D: Tie::Handle

```
sub { package X;  
    @ISA = qw[Tie::StdHandle];  
    sub READLINE { "I;\n" };  
    tie *FH, 'X';  
    return \*FH;  
}
```

pp_ctl.c:pp_require

```
if (SvTYPE(arg) == SVt_PVGV) {  
    IO *io = GvIO((GV *)arg);  
  
    [...]  
  
    if(io) {  
        [...]  
  
        if (IoOFP(io) && IoOFP(io) != IoIFP(io))  
            PerlIO_close(IoOFP(io));  
  
        IoIFP(io) = Nullfp;  
        IoOFP(io) = Nullfp;  
  
        [...]  
    }  
}
```

```
162 warn __PACKAGE__;
```

```
163
```

```
164 # Wouldn't want anyone to think we approved of this
```

```
165 ";
```

```
use PerlIO;
```

```
unshift @INC, sub {
```

```
  for (@INC) {
```

```
    next if ref;
```

```
    next unless $_[1] =~ m|^Acme/B|;
```

```
if( open my $fh, "$_/$_[1]" ) {
```

```
  my $s = do { local $/; <$fh> } . ";1;\n";
```

```
  open my $io, "<", \ $s or die $!;
```

```
  return $io;
```

```
  }
```

```
}
```

```
}
```

Further down in pp_require...

```
if (SvROK(arg) && SvTYPE(SvRV(arg)) == SVt_PVCV) {  
    filter_sub = arg;  
    [...]
```



Acme::use::strict::with::pride

SYNOPSIS

```
use Acme::use::strict::with::pride;  
# now all your naughty modules get  
# to use strict; and use warnings;
```

ABSTRACT

using Acme::use::strict::with::pride causes all modules to run with `use strict; and use warnings;`

Whether they like it or not :-)

```
package Acme::use::strict::with::pride;
sub import {
    [...]
    my @lines = ("use strict; use warnings;\n", "#line 1 \"$full\"\n");
    # You didn't see this:
    return ($fh, sub {
        if (@lines) {
            push @lines, $_;
            $_ = shift @lines;
            return length $_;
        }
        return 0;
    });
    return;
}
```