PASSWORD-CHECKER

Olivier Bilodeau

PASSWORD-CHECKER

- Grab the binary: http://montrehack.ca
- First hint in 30 minutes

HINT #1

- It's a Perl interpreter

```
$ strings -a password-checker
```

- The flag must be hidden inside
- But there is a **lot** of code
- `B::Deparse` can decompile Perl's internal compiled structures
- Next hint in 30 minutes

HINT #2

- Looks like ELF produced by `perlcc` (google hint)
- You got `B::Decode` running right?
- If we could load a binary compatible environment, maybe `B::Deparse` would work better?
- How is this thing deployed?
- Perl 5.8.9 is end-of-life since 2008 :(
- plenv

HINT #3

- plenv is setup right?
- you have the exact perl version? 32-bit? i686-linux-multi/ is there?
- You need a 32-bit linux VM. http://www.vagrantbox.es/ is easiest for quick setup
- `plenv install 5.8.9 -Dusemultiplicity`
- Some of the usual `B::Deparse` tricks don't work
- Think of `eval`-ing perl code in the right context
- Last hint in 30 minutes

HINT $LAST

You got that `eval` thing working, right?

```
(gdb) b perl_run
Breakpoint 1 at 0x80fc5ea: file perl.c, line 2350.
(gdb) r
Starting program: /home/olivier/Documents/projets/infosec/adctf2014/15/pa

Breakpoint 1, perl_run (my_perl=0x81d0008) at perl.c:2350
2350    perl.c: No such file or directory.
(gdb) p Perl_eval_pv (my_perl, "use B::Deparse; B::Deparse->compile->()",
```

Warnings:

- be careful of output buffering, add a `die("flush")` to flush the buffers
- backslashes need to be escaped (gdb eats them)

HINT $LAST (CONT.)

Think introspection. Like python's

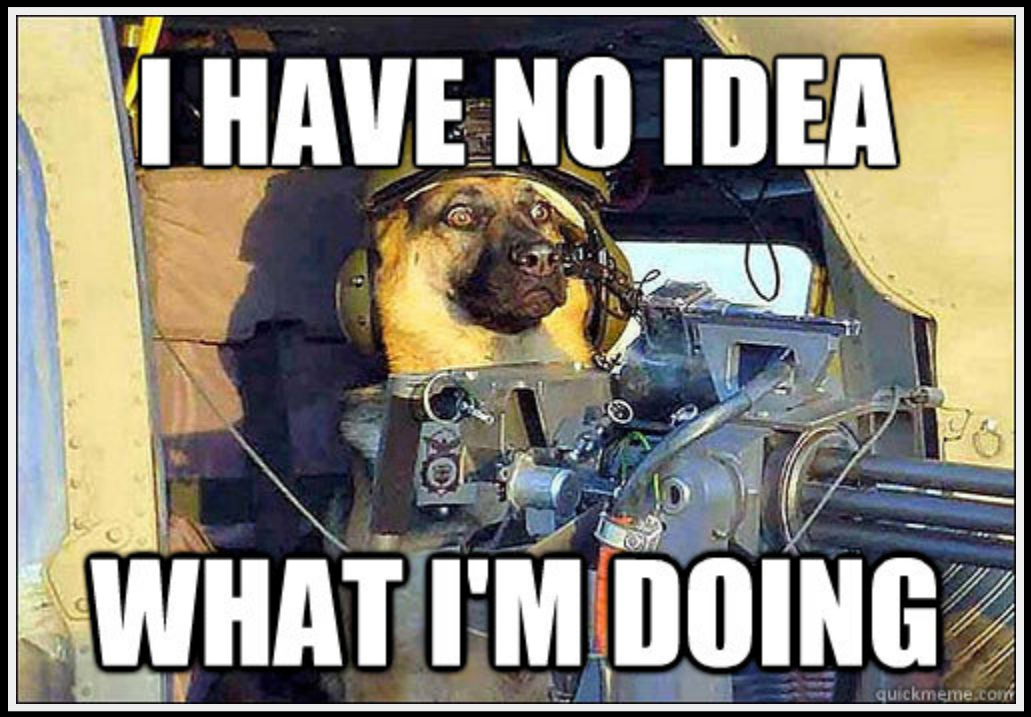
```
import __main__ as i
dir(i)
```

Of course I won't tell you perl's equivalent ;)

SOLUTION

HOW I SAW MYSELF

HOW IT REALLY HAPPENED



FIRST, B::DEPARSE

B::Deparse is **the** thing to reverse perl code.

Trying to add B::Deparse to my PERL5OPT environment variable to force it to dump its code

```
$ export PERL5OPT=-MO=Deparse
$ ./password-checker
-e syntax OK
```

This usually work but it didn’t work here.

HOW CAN I BYPASS THIS CHECK?

- gdb and break later? but break where?
- listing calls under main

```
...
0x080a9e6d    e8aa9afbff   call sym.imp.exit
0x080a9e79    e8a2b50b00   call sym.Perl_Gcurinterp_ptr
0x080a9e83    e800030000   call sym.dl_init
0x080a9e90    e84b270500   call sym.perl_run
0x080a9ea1    e87a540500   call sym.perl_destruct
0x080a9eae    e84d540500   call sym.perl_free
...
```

- perl_run stands out as a good spot
- and it's documented here: http://stackoverflow.com/a/4048916

EVAL'ING CODE

Use `eval` to load `B::Deparse` and dump optree

```
gdb-peda$ b perl_run
Breakpoint 1 at 0x80fc5ea: file perl.c, line 2350.

gdb-peda$ r
Breakpoint 1, perl_run (my_perl=0x81d0008) at perl.c:2350
2350    perl.c: No such file or directory.

gdb-peda$ p Perl_eval_pv (my_perl, "use B::Deparse; B::Deparse->compile->
package [w;
use warnings;
use strict 'refs';
print 'password: ';
While deparsing near line 5,
at /home/vagrant/.plenv/versions/5.8.9/lib/perl5/5.8.9/i686-linux-multi/B
    B::Deparse::gv_name('B::Deparse=HASH(0x83935bc)', 'B::SPECIAL=SCALAR
[...]
    eval 'use B::Deparse; B::Deparse->compile->()
' called at (eval 1) line 0
```

WHAT HAPPENED?

`B::Deparse` relies on C code we need to have a binary compatible module ready to load.

THE RIGHT ENVIRONMENT

- based on strings in the binary:

```
/home/vagrant/.plenv/versions/5.8.9/lib/perl5/5.8.9/i686-linux-multi/
```

- 32-bit plenv Perl built with `-Dusemultiplicity` (builds the native code with multiple interpreter per process support)

SETUP A VM

Since vagrant is all over the place, lets leverage it by doing the same thing. (and I love fancy devops tools)

- vagrant up (an old CentOS 32-bit box)
- yum install git gdb
- install plenv (git clone two repos)
- install Perl 5.8.9 with binary compatibility with `password-checker`

```
~/.plenv/bin/plenv install 5.8.9 -Dusemultiplicity
```

- vagrant ssh

GETTING B::DEPARSE RIGHT

If you used `vagrant` + `plenv` you don't need to mangle with PERL5LIB since all the files will be at the expected place.

```
(gdb) p Perl_eval_pv (my_perl, "use B::Deparse; B::Deparse->compile->()",
use warnings;
use strict 'refs';
print 'password: ';
<ARGV>;
print "wrong\n";
my(@c) = ('"', '`', '!', '$', '#', '{', '/', '&', '_', '^', '.', '[', '(
          '*', ')', '+', '%', ',', '\\', ']', '-', '', ':', '=', ';',
$1 = (SV *) 0x83dfde0
```

Still no flag…

$INTROSPECTION

Lets eval funky stuff

```
local $, = '\n'; print %main::;
```

Lists all top-level symbols.

RESULTS

```
...
flag
*main::flag
...
```

FOUND FLAG

There's a flag symbol but I can't print it…

```
print $flag;
```

or

```
print $main::flag;
```

Yields nothing.

SO WHAT IS IT EXACTLY?

```
{
    no strict;
    local $, = '\n';
    print 'its a scalar (or a ref)' if defined($flag);
    print 'its an array' if defined(@flag);
    print 'its a hash' if defined(%flag);
    print 'its code' if defined(&flag);
    die('flush buffers');
}
```

```
(gdb) p Perl_eval_pv (my_perl, "{ no strict; local $, = '\n'; print 'its
```

Turns out its code…

DUMP IT

`B::Deparse` to the rescue!

```
use B::Deparse;
$deparse = B::Deparse->new();
print $deparse->coderef2text(*main::flag{CODE});
```

`*main::flag{CODE}` is a fancy way of saying `\&flag` (reference to `flag()`) without a backslash

```
(gdb) p Perl_eval_pv (my_perl, "use B::Deparse; $deparse = B::Deparse->ne
```

RESULT

```
{
use warnings;
use strict 'refs';
die "^_^\n";
print eval eval $c[0] . $c[12] . $c[18] . $c[3] . ($c[1] | $c[4]) . $c[11
               . ($c[9] ^ ($c[1] | $c[6])) . $c[19] . $c[9] . $c[18] .
               $c[3] . ($c[1] | $c[4]) . $c[11] . ($c[9] ^ ($c[1] |
               $c[17])) . $c[19] . $c[14] . $c[10] . $c[12] . $c[12] .
[...]
}
```

- Array operations seems to hold our flag
- Behind the `die()` so it's never reached
- We saw this `c` array before…

FINISH HIM

- Take @c definition from main scope
- Add the `print eval eval` code
- Run it
- $FLAG

THANKS!

Any questions?

REFERENCES

- http://adctf2014.katsudon.org/, Day 15
- http://stackoverflow.com/a/4048916
- https://metacpan.org/pod/B::Deparse
- https://www.vagrantup.com/
- https://github.com/tokuhirom/plenv
- https://github.com/tokuhirom/perl-build
- `perldoc perlmod`: The main:: special package
- `perldoc perlref`: Typeglobs and references