# Patching Binaries to Improve Fuzzing

Alex Eubanks <alex.eubanks@forallsecure.com>

August 1, 2018

## Agenda

## Introduction

In this tutorial, we will learn how to modify binary applications to improve the fuzzing process. If source code is available, there is never a reason to modify a binary. Make the necessary changes in source, and recompile your binary for testing.

When testing of 3rd-party, or legacy applications, is required, source code may not be available. Binary patching can be used to improve the speed and efficiency of testing these programs with MAYHEM.

## Requirements

- A working copy of MAYHEM, with the mayhem client installed on the command line.
- Reverse-engineering expertise. This is an advanced tutorial, and you should be comfortable with assembly-level code.
- A disassembler with the ability to patch instructions. We will be using Binary Ninja[1].
- The example files that are referenced throughout the slide deck.

[1]https://binary.ninja

## Why Patch Binaries

Binary modification is not required for fuzzing, but there are several reasons why vulnerability researchers sometimes modify binaries. Some example reasons:

- Allow a program to cleanly exit after running a single testcase.
- Remove checksum checks that some input-generation tools are not aware of.
- Remove cryptographic primitives from a binary.
- Remove irritating program behaviors that disrupt fuzzing.

In general, we want to remove checks that an attacker can create themselves, but will cause a program to exit before exercising code paths. A common example is a checksum over an incoming data stream.

## A Checksum Story

Checksums can be disruptive to fuzzing. They cause applications to dismiss input which may otherwise trigger a bug. Attackers will checksum their malicious data before feeding it to your applications, so we shouldn't allow checksums over input data to slow down our analysis.

If we fail to account for this checksum, each time MAYHEM generates a testcase with a bad checksum, it will be thrown out. There are two basic ways to account for this checksum:

- We can pass our inputs through a harness, which will ensure proper checksums are in place before passing those inputs to the program under test.
- We can patch the binary so that it accepts all inputs, even those with invalid checksums.

We will explore the second option in this tutorial.

## A Checksum Story: Imagemagick

In this example, we are going to fuzz png parsing in a popular command-line image-manipulation program called "imagemagick"[2].

Imagemagick is an opensource, freely available application with source. Normally we would recommend you make any necessary modifications in source, but we will be dismissing source code and working directly with the binary for this tutorial.

[2] https://www.imagemagick.org/

FOR ALL SECURE

## Getting set up with Imagemagick

Let's start from a fresh Debian Stretch linux environment, and install imagemagick directly from the package repositories.

```
apt-get update && \
apt-get dist-upgrade -y && \
apt-get install -y imagemagick
```

And let's run imagemagick over an image:

```
$ convert /tutorials/binary-patching/imagemagic/nsa-insignia
    -sm.png -enhance /tutorials/binary-patching/imagemagic/
    enhanced.png
$
```

No errors, Perfect!

FOR ALL SECURE

## Getting set up with Imagemagick

Knowing that the PNG file format uses crc checksums, I've created a modified PNG which causes imagemagic to print out failed checksum errors. I did this by running radamsa[3] over our `nsa-insignia-sm.png` file, and testing the output until I had a new PNG that gave an error for invalid checksum.

```
root@a67f5575a117:~# convert /tutorials/binary-patching/
    imagemagic/nsa-insignia-crc-error.png /tmp/out.png
convert-im6.q16: IDAT: CRC error `/tutorials/binary-patching
    /imagemagic/nsa-insignia-crc-error.png' @ error/png.c/
    MagickPNGErrorHandler/1628.
root@a67f5575a117:~#
```

We can see the CRC error, which is exactly what we want. Perfect!

[3] https://github.com/aoh/radamsa

## Finding the CRC Check

We now need to find the code which is performing this CRC check, and remove it by patching the binary.

There are a couple ways we can move forward. We can:

- Search for magic numbers associated with the algorithm we are trying to modify, and work backwards to see where the output of that algorithm is verified against our input.
- Use the strings in the error message to find the relevant code.
- Reverse-engineer the binary until we find the correct place to patch.

Since we have strings in an error message, let's start with those and see if we can find out how to patch around this crc check.

FOR ALL SECURE

## Finding the "CRC Error" string

We'll start by using our good friend grep to find the string "CRC error".

```
root@a67f5575a117:~# grep "CRC error" /usr/bin/convert
root@a67f5575a117:~#
```

---

Not in the convert binary... Let's take a look at the dependencies on the next slide.

## Listing dependencies

```
root@a67f5575a117:~# ldd /usr/bin/convert
linux-vdso.so.1 (0x00007ffec85d3000)
libMagickCore-6.Q16.so.3 => /usr/lib/x86_64-linux-gnu/libMagickCor
libMagickWand-6.Q16.so.3 => /usr/lib/x86_64-linux-gnu/libMagickWan
libpthread.so.0 => /lib/x86_64-linux-gnu/libpthread.so.0 (0x00007f
...
libglib-2.0.so.0 => /lib/x86_64-linux-gnu/libglib-2.0.so.0 (0x0000
libexpat.so.1 => /lib/x86_64-linux-gnu/libexpat.so.1 (0x00007f29ff
libpng16.so.16 => /usr/lib/x86_64-linux-gnu/libpng16.so.16 (0x0000
libxcb.so.1 => /usr/lib/x86_64-linux-gnu/libxcb.so.1 (0x00007f29fe
...
libbsd.so.0 => /lib/x86_64-linux-gnu/libbsd.so.0 (0x00007f29fe4190
librt.so.1 => /lib/x86_64-linux-gnu/librt.so.1 (0x00007f29fe211000
```

`libpng16.so.16`, this looks interesting.

## Finding the "CRC error" string

Let's search for the string in `/usr/lib/x86_64-linux-gnu`, where most of our dependencies reside.

```
root@a67f5575a117:~# grep "CRC error" /usr/lib/x86_64-linux-
    gnu/*
...
Binary file /usr/lib/x86_64-linux-gnu/libpng16.so.16 matches
Binary file /usr/lib/x86_64-linux-gnu/libpng16.so.16.28.0
    matches
...
```

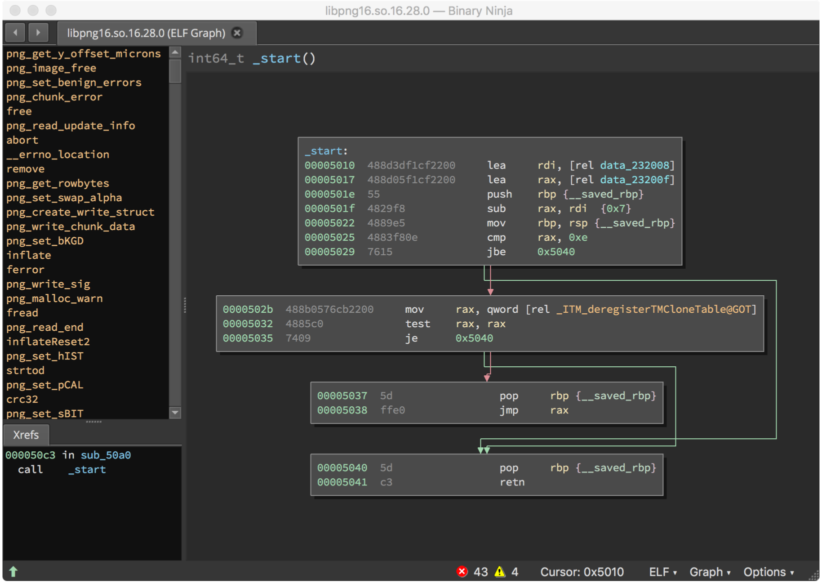
It looks like we need to patch `libpng16.so.16.28.0`. Your linux distribution may have a different version of libpng, but you should be fine to play along with whichever version you have.

Let's open `libpng16.so.16.28.0` in Binary Ninja.

FOR ALL SECURE

## Libpng in Binary Ninja

Here we have the libpng library open in Binary Ninja.


libpng16.so.16.28.0 — Binary Ninja
libpng16.so.16.28.0 (ELF Graph)
Xrefs
43 4 Cursor: 0x5010 ELF Graph Options

```
png_get_y_offset_microns
png_image_free
png_set_benign_errors
png_chunk_error
free
png_read_update_info
abort
__errno_location
remove
png_get_rowbytes
png_set_swap_alpha
png_create_write_struct
png_write_chunk_data
png_set_bKGD
inflate
ferror
png_write_sig
png_malloc_warn
fread
png_read_end
inflateReset2
png_set_hIST
strtod
png_set_pCAL
crc32
png_set_sBIT
```

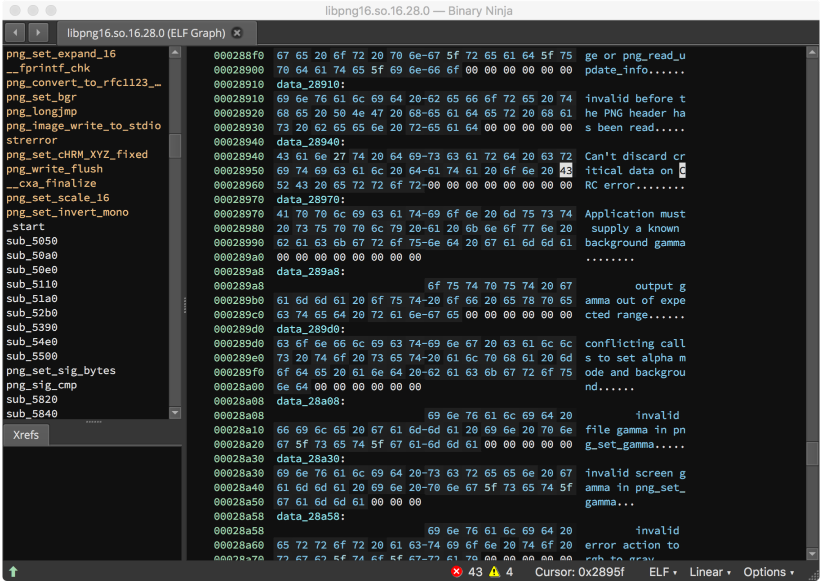
```
000050c3 in sub_50a0
  call    _start
```

```
int64_t _start()

_start:
00005010  488d3df1cf2200    lea     rdi, [rel data_232008]
00005017  488d05f1cf2200    lea     rax, [rel data_23200f]
0000501e  55                push    rbp {__saved_rbp}
0000501f  4829f8            sub     rax, rdi  {0x7}
00005022  4889e5            mov     rbp, rsp {__saved_rbp}
00005025  4883f80e          cmp     rax, 0xe
00005029  7615              jbe     0x5040

0000502b  488b0576cb2200    mov     rax, qword [rel _ITM_deregisterTMCloneTable@GOT]
00005032  4885c0            test    rax, rax
00005035  7409              je      0x5040

00005037  5d                pop     rbp {__saved_rbp}
00005038  ffe0              jmp     rax

00005040  5d                pop     rbp {__saved_rbp}
00005041  c3                retn
```

FOR ALL SECURE

# Libpng in Binary Ninja

We begin searching for "CRC error". It looks like we found a substring.



libpng16.so.16.28.0 — Binary Ninja

libpng16.so.16.28.0 (ELF Graph)

```
png_set_expand_16
__fprintf_chk
png_convert_to_rfc1123_…
png_set_bgr
png_longjmp
png_image_write_to_stdio
strerror
png_set_cHRM_XYZ_fixed
png_write_flush
__cxa_finalize
png_set_scale_16
png_set_invert_mono
_start
sub_5050
sub_50a0
sub_50e0
sub_5110
sub_51a0
sub_52b0
sub_5390
sub_54e0
sub_5500
png_set_sig_bytes
png_sig_cmp
sub_5820
sub_5840
```

Xrefs

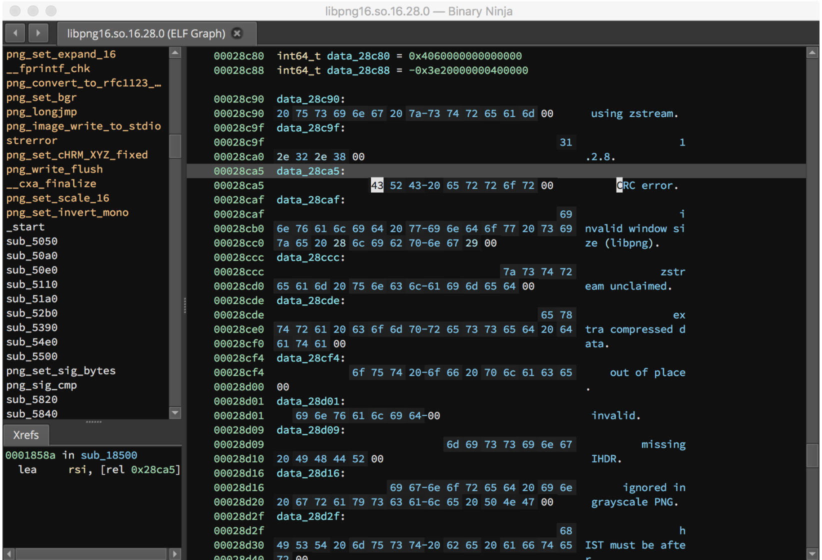
```
000288f0  67 65 20 6f 72 20 70 6e-67 5f 72 65 61 64 5f 75  ge or png_read_u
00028900  70 64 61 74 65 5f 69 6e-66 6f 00 00 00 00 00 00  pdate_info......
00028910  data_28910:
00028910  69 6e 76 61 6c 69 64 20-62 65 66 6f 72 65 20 74  invalid before t
00028920  68 65 20 50 4e 47 20 68-65 61 64 65 72 20 68 61  he PNG header ha
00028930  73 20 62 65 65 6e 20 72-65 61 64 00 00 00 00 00  s been read.....
00028940  data_28940:
00028940  43 61 6e 27 74 20 64 69-73 63 61 72 64 20 63 72  Can't discard cr
00028950  69 74 69 63 61 6c 20 64-61 74 61 20 6f 6e 20 43  itical data on C
00028960  52 43 20 65 72 72 6f 72-00 00 00 00 00 00 00 00  RC error........
00028970  data_28970:
00028970  41 70 70 6c 69 63 61 74-69 6f 6e 20 6d 75 73 74  Application must
00028980  20 73 75 70 70 6c 79 20-61 20 6b 6e 6f 77 6e 20   supply a known
00028990  62 61 63 6b 67 72 6f 75-6e 64 20 67 61 6d 6d 61  background gamma
000289a0  00 00 00 00 00 00 00 00                          ........
000289a8  data_289a8:
000289a8                          6f 75 74 70 75 74 20 67          output g
000289b0  61 6d 6d 61 20 6f 75 74-20 6f 66 20 65 78 70 65  amma out of expe
000289c0  63 74 65 64 20 72 61 6e-67 65 00 00 00 00 00 00  cted range......
000289d0  data_289d0:
000289d0  63 6f 6e 66 6c 69 63 74-69 6e 67 20 63 61 6c 6c  conflicting call
000289e0  73 20 74 6f 20 73 65 74-20 61 6c 70 68 61 20 6d  s to set alpha m
000289f0  6f 64 65 20 61 6e 64 20-62 61 63 6b 67 72 6f 75  ode and backgrou
00028a00  6e 64 00 00 00 00 00 00                          nd......
00028a08  data_28a08:
00028a08                          69 6e 76 61 6c 69 64 20          invalid
00028a10  66 69 6c 65 20 67 61 6d-6d 61 20 69 6e 20 70 6e  file gamma in pn
00028a20  67 5f 73 65 74 5f 67 61-6d 6d 61 00 00 00 00 00  g_set_gamma.....
00028a30  data_28a30:
00028a30  69 6e 76 61 6c 69 64 20-73 63 72 65 65 6e 20 67  invalid screen g
00028a40  61 6d 6d 61 20 69 6e 20-70 6e 67 5f 73 65 74 5f  amma in png_set_
00028a50  67 61 6d 6d 61 00 00 00                          gamma...
00028a58  data_28a58:
00028a58                          69 6e 76 61 6c 69 64 20          invalid
00028a60  65 72 72 6f 72 20 61 63-74 69 6f 6e 20 74 6f 20  error action to
00028a70  72 67 62 5f 74 6f 5f 67-72 61 79 00 00 00 00 00  rgb_to_gray
```

43 4 Cursor: 0x2895f ELF ▾ Linear ▾ Options ▾

FOR ALL SECURE

## Searching for "CRC error"

It looks like we've found the right "CRC error" string. Follow the Xref in the bottom-left corner



libpng16.so.16.28.0 — Binary Ninja

libpng16.so.16.28.0 (ELF Graph)

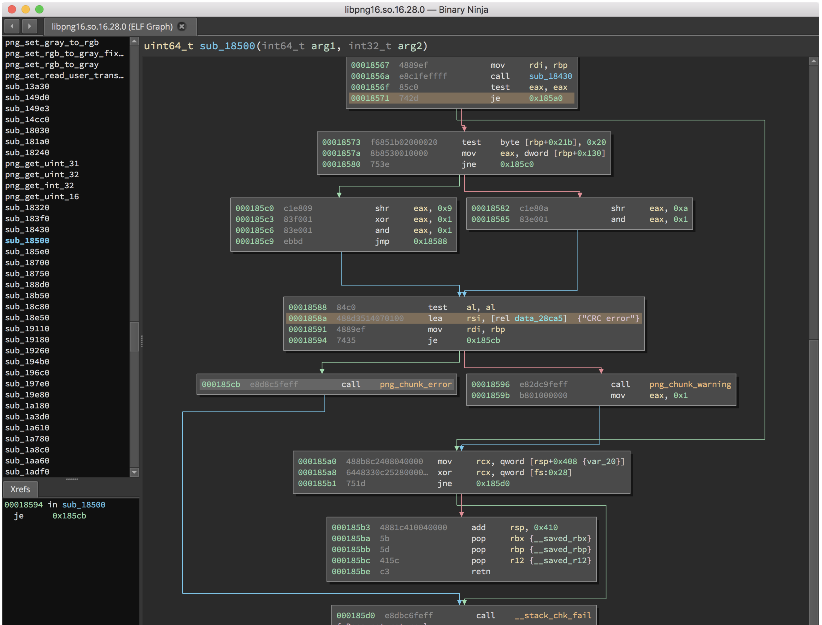
```
png_set_expand_16
__fprintf_chk
png_convert_to_rfc1123_…
png_set_bgr
png_longjmp
png_image_write_to_stdio
strerror
png_set_cHRM_XYZ_fixed
png_write_flush
__cxa_finalize
png_set_scale_16
png_set_invert_mono
_start
sub_5050
sub_50a0
sub_50e0
sub_5110
sub_51a0
sub_52b0
sub_5390
sub_54e0
sub_5500
png_set_sig_bytes
png_sig_cmp
sub_5820
sub_5840
```

Xrefs

```
0001858a in sub_18500
  lea     rsi, [rel 0x28ca5]
```

```
00028c80  int64_t data_28c80 = 0x4060000000000000
00028c88  int64_t data_28c88 = -0x3e20000000400000

00028c90  data_28c90:
00028c90  20 75 73 69 6e 67 20 7a-73 74 72 65 61 6d 00      using zstream.
00028c9f  data_28c9f:
00028c9f                                             31                 1
00028ca0  2e 32 2e 38 00                                  .2.8.
00028ca5  data_28ca5:
00028ca5                 43 52 43-20 65 72 72 6f 72 00          CRC error.
00028caf  data_28caf:
00028caf                                             69                 i
00028cb0  6e 76 61 6c 69 64 20 77-69 6e 64 6f 77 20 73 69  nvalid window si
00028cc0  7a 65 20 28 6c 69 62 70-6e 67 29 00              ze (libpng).
00028ccc  data_28ccc:
00028ccc                                    7a 73 74 72              zstr
00028cd0  65 61 6d 20 75 6e 63 6c-61 69 6d 65 64 00        eam unclaimed.
00028cde  data_28cde:
00028cde                                          65 78                ex
00028ce0  74 72 61 20 63 6f 6d 70-72 65 73 73 65 64 20 64  tra compressed d
00028cf0  61 74 61 00                                      ata.
00028cf4  data_28cf4:
00028cf4              6f 75 74 20-6f 66 20 70 6c 61 63 65      out of place
00028d00  00                                               .
00028d01  data_28d01:
00028d01     69 6e 76 61 6c 69 64-00                         invalid.
00028d09  data_28d09:
00028d09                             6d 69 73 73 69 6e 67           missing
00028d10  20 49 48 44 52 00                                 IHDR.
00028d16  data_28d16:
00028d16                       69 67-6e 6f 72 65 64 20 69 6e        ignored in
00028d20  20 67 72 61 79 73 63 61-6c 65 20 50 4e 47 00      grayscale PNG.
00028d2f  data_28d2f:
00028d2f                                             68                 h
00028d30  49 53 54 20 6d 75 73 74-20 62 65 20 61 66 74 65  IST must be afte
```

# Patching out the CRC check


libpng16.so.16.28.0 — Binary Ninja
libpng16.so.16.28.0 (ELF Graph)
Xrefs

```
png_set_gray_to_rgb
png_set_rgb_to_gray_fix…
png_set_rgb_to_gray
png_set_read_user_trans…
sub_13a30
sub_149d0
sub_149e3
sub_14cc0
sub_18030
sub_181a0
sub_18240
png_get_uint_31
png_get_uint_32
png_get_int_32
png_get_uint_16
sub_18320
sub_183f0
sub_18430
sub_18500
sub_185e0
sub_18700
sub_18750
sub_188d0
sub_18b50
sub_18c80
sub_18e50
sub_19110
sub_19180
sub_19260
sub_194b0
sub_196c0
sub_197e0
sub_19e80
sub_1a180
sub_1a3d0
sub_1a610
sub_1a780
sub_1a8c0
sub_1aa60
sub_1adf0
```

```
00018594 in sub_18500
  je       0x185cb
```

```
uint64_t sub_18500(int64_t arg1, int32_t arg2)

00018567  4889ef            mov     rdi, rbp
0001856a  e8c1feffff        call    sub_18430
0001856f  85c0              test    eax, eax
00018571  742d              je      0x185a0

00018573  f6851b02000020    test    byte [rbp+0x21b], 0x20
0001857a  8b8530010000      mov     eax, dword [rbp+0x130]
00018580  753e              jne     0x185c0

000185c0  c1e809            shr     eax, 0x9
000185c3  83f001            xor     eax, 0x1
000185c6  83e001            and     eax, 0x1
000185c9  ebbd              jmp     0x18588

00018582  c1e80a            shr     eax, 0xa
00018585  83e001            and     eax, 0x1

00018588  84c0              test    al, al
0001858a  488d3514070100    lea     rsi, [rel data_28ca5]  {"CRC error"}
00018591  4889ef            mov     rdi, rbp
00018594  7435              je      0x185cb

000185cb  e8d8c5feff        call    png_chunk_error

00018596  e82dc9feff        call    png_chunk_warning
0001859b  b801000000        mov     eax, 0x1

000185a0  488b8c2408040000  mov     rcx, qword [rsp+0x408 {var_20}]
000185a8  6448330c25280000… xor     rcx, qword [fs:0x28]
000185b1  751d              jne     0x185d0

000185b3  4881c410040000    add     rsp, 0x410
000185ba  5b                pop     rbx {__saved_rbx}
000185bb  5d                pop     rbp {__saved_rbp}
000185bc  415c              pop     r12 {__saved_r12}
000185be  c3                retn

000185d0  e8dbc6feff        call    __stack_chk_fail
```

## Patching out the CRC check

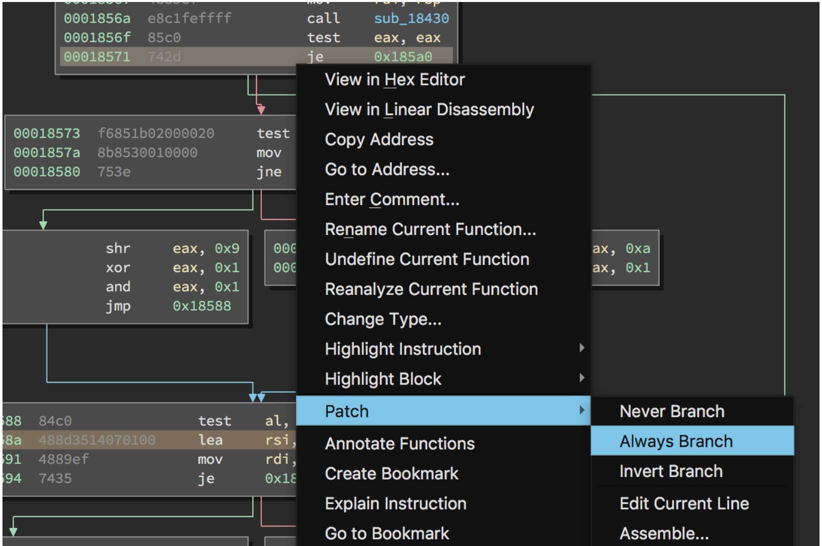
A quick look at this function shows our "CRC error" string being loaded immediately prior to a call to `png_chunk_error` or `png_chunk_warn`. We hope all we need to do is avoid the blocks which drive us over `png_chunk_error` and `png_chunk_warn`.

If we can always take the branch at `0x18571`, hopefully we can avoid libpng dying on checksum errors, and we will be able to more efficiently fuzz imagemagick.

FOR ALL SECURE

## Libpng in Binary Ninja

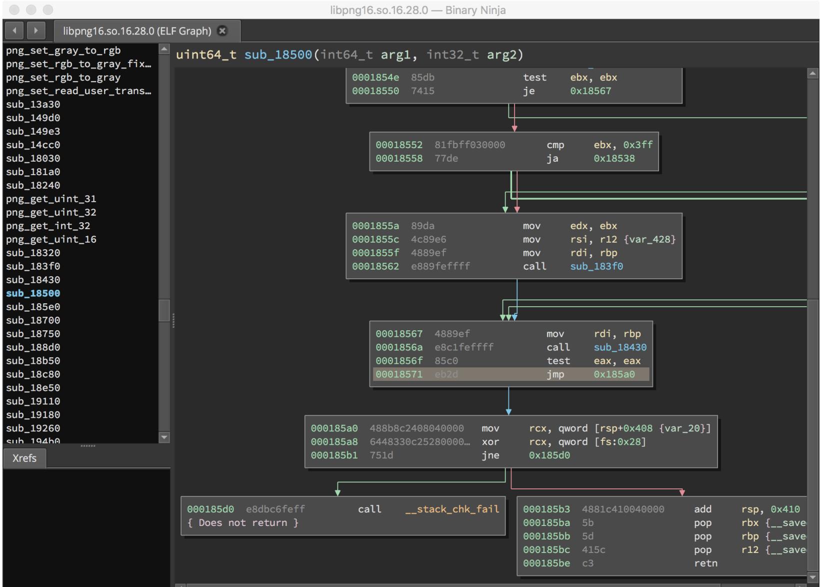
Right-click the instruction at `0x18571`, and choose `Patch -> Always Branch`.



```
0001856a  e8c1feffff        call  sub_18430
0001856f  85c0              test  eax, eax
00018571  742d              je    0x185a0

00018573  f6851b02000020    test
0001857a  8b8530010000      mov
00018580  753e              jne

          shr   eax, 0x9
          xor   eax, 0x1
          and   eax, 0x1
          jmp   0x18588

88   84c0            test  al,
8a   488d3514070100  lea   rsi,
91   4889ef          mov   rdi,
94   7435            je    0x18
```

# Libpng in Binary Ninja

Our patched libpng binary, with errors on CRC mismatches removed.


libpng16.so.16.28.0 — Binary Ninja

libpng16.so.16.28.0 (ELF Graph)

```
png_set_gray_to_rgb
png_set_rgb_to_gray_fix…
png_set_rgb_to_gray
png_set_read_user_trans…
sub_13a30
sub_149d0
sub_149e3
sub_14cc0
sub_18030
sub_181a0
sub_18240
png_get_uint_31
png_get_uint_32
png_get_int_32
png_get_uint_16
sub_18320
sub_183f0
sub_18430
sub_18500
sub_185e0
sub_18700
sub_18750
sub_188d0
sub_18b50
sub_18c80
sub_18e50
sub_19110
sub_19180
sub_19260
```

Xrefs

```
uint64_t sub_18500(int64_t arg1, int32_t arg2)

0001854e  85db              test    ebx, ebx
00018550  7415              je      0x18567

00018552  81fbff030000      cmp     ebx, 0x3ff
00018558  77de              ja      0x18538

0001855a  89da              mov     edx, ebx
0001855c  4c89e6            mov     rsi, r12 {var_428}
0001855f  4889ef            mov     rdi, rbp
00018562  e889feffff        call    sub_183f0

00018567  4889ef            mov     rdi, rbp
0001856a  e8c1feffff        call    sub_18430
0001856f  85c0              test    eax, eax
00018571  eb2d              jmp     0x185a0

000185a0  488b8c2408040000  mov     rcx, qword [rsp+0x408 {var_20}]
000185a8  6448330c25280000… xor     rcx, qword [fs:0x28]
000185b1  751d              jne     0x185d0

000185d0  e8dbc6feff        call    __stack_chk_fail
{ Does not return }

000185b3  4881c410040000    add     rsp, 0x410
000185ba  5b                pop     rbx {__save
000185bb  5d                pop     rbp {__save
000185bc  415c              pop     r12 {__save
000185be  c3                retn
```

FOR ALL SECURE

## Test our patched libpng

In Binary Ninja, go to `File -> Save Contents As`, and save your new `lbpng.so.16.28.0`. Make sure that overwrites the original `libpng.so`, and rerun our convert program.

```
root@a67f5575a117:~# convert /tutorials/binary-patching/
    imagemagic/nsa-insignia-crc-error.png /tmp/out.png
convert-im6.q16: IDAT: CRC error `/tutorials/binary-patching
    /imagemagic/nsa-insignia-crc-error.png' @ error/png.c/
    MagickPNGErrorHandler/1628.
root@a67f5575a117:~# cp /tutorials/binary-patching/
    imagemagic/libpng16.so.16.28.0.patched /usr/lib/x86_64-
    linux-gnu/libpng16.so.16.28.0
root@a67f5575a117:~# convert /tutorials/binary-patching/
    imagemagic/nsa-insignia-crc-error.png /tmp/out.png
convert-im6.q16: PNG unsigned integer out of range `/
    tutorials/binary-patching/imagemagic/nsa-insignia-crc-
    error.png' @ error/png.c/MagickPNGErrorHandler/1628.
```

Perfect! Now we don't have to worry about imagemagick throwing out inputs just because they have incorrect checksums.

## Packaging for MAYHEM

We can go ahead and package up this binary, and then send it off to MAYHEM for analysis.

```
root@a67f5575a117:~# mayhem package /usr/bin/convert -o convert
INFO:mayhem.mdbclient.commands.package:Packaging application: /usr
INFO:mayhem.mdbclient.commands.package:Packaging dependency: /usr/
INFO:mayhem.mdbclient.commands.package:Packaging dependency: /usr/
INFO:mayhem.mdbclient.commands.package:Packaging dependency: /usr/
...
INFO:mayhem.mdbclient.commands.package:Packaging dependency: /lib/
INFO:mayhem.mdbclient.commands.package:Packaging dependency: /lib/
INFO:mayhem.mdbclient.commands.package:Generating default configur
INFO:mayhem.mdbclient.commands.package:Packaged /usr/bin/convert u
root@a67f5575a117:~#
```

## Packaging for MAYHEM

We need to give MAYHEM the proper commandline invocation for `convert`. Open `convert/config.json` with your favorite text editor, and change:

```
"target_args": [
    "@@"
]
```

to

```
"target_args": [
    "@@",
    "/dev/null"
]
```

## Upload to MAYHEM

All that's left is uploading the package to MAYHEM!

```
root@a67f5575a117:~# mayhem upload -u http
    ://192.168.99.100:30128/ --start-sword convert
WARNING:mayhem.mdbclient.client:Using local environment to
    figure out the image
INFO:mayhem.mdbclient.commands.upload:Application
    successfully uploaded: 1 with harness 1
INFO:mayhem.mdbclient.commands.upload:Default job added (
    job_id: 1). Use that job_id to post results to the API.
```

## Questions & Comments

Questions & Comments