Patching Binaries to Improve Fuzzing

Alex Eubanks <alex.eubanks@forallsecure.com>

June 11, 2018

FOR ALL SECURE

Agenda

1 Agenda

2 Introduction
- Requirements
- Why Patch Binaries
- Example Scenarios

3 A Checksum Story: Imagemagick
- Getting set up with Imagemagick
- Finding the CRC Check
- Modifying libpng
- Packaging for MAYHEM

4 Questions & Comments

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.

FOR ALL SECURE

Requirements

- 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].
- A hex editor. We will be using Hex Fiend[2], a free hex editor for Mac OS X.
- The example files that are referenced with the slide deck.

[1] https://binary.ninja
[2] https://ridiculousfish.com/hexfiend/

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"[3].

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.

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

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!

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[4] 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!

---

[4] 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.

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
...
```

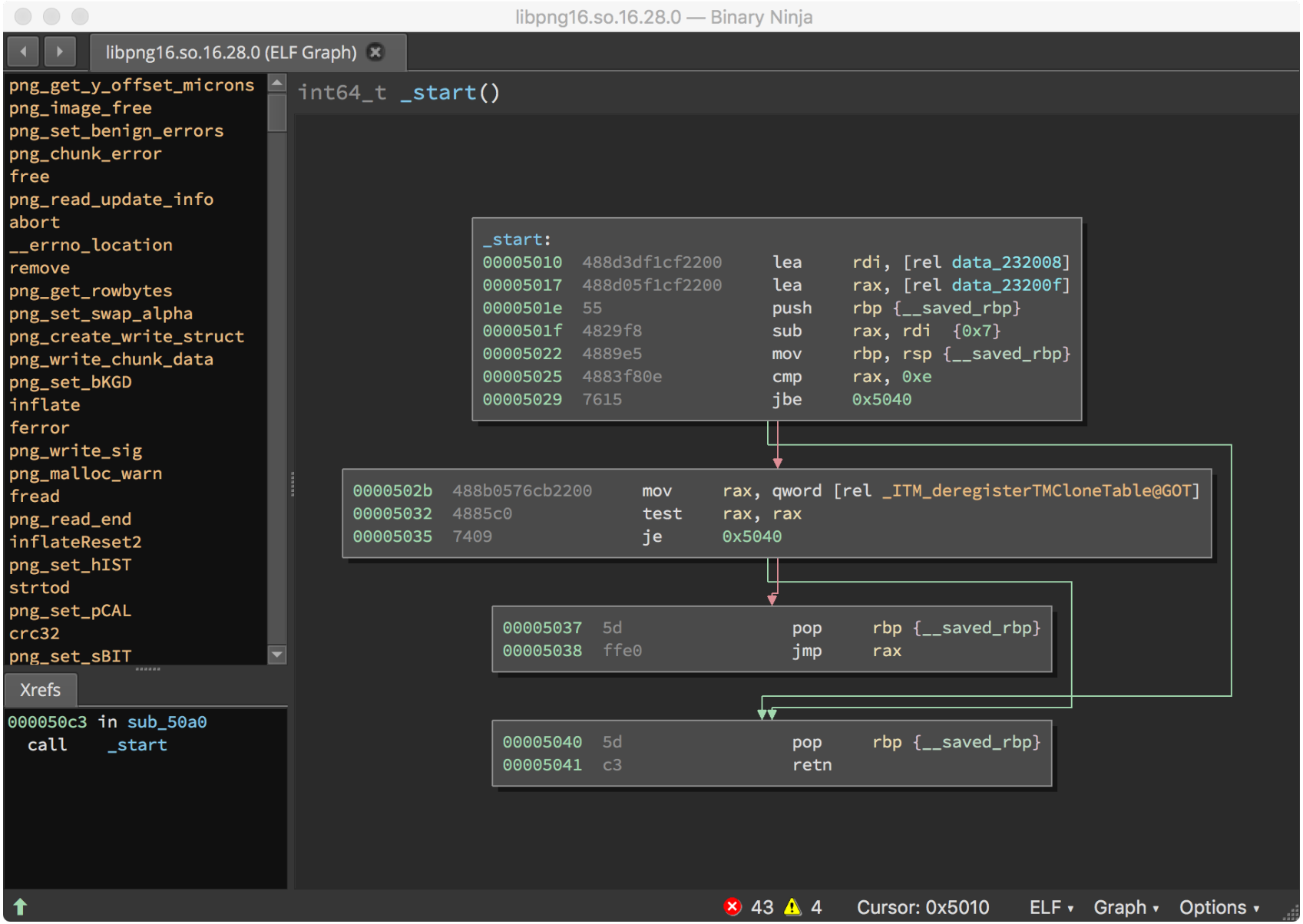
It looks like we need to patch `libpng16.so.16.28.0`. Your linux distribution may have a different version of libpng, but you should be fine to play along with whichever version you have.

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

FOR ALL SECURE

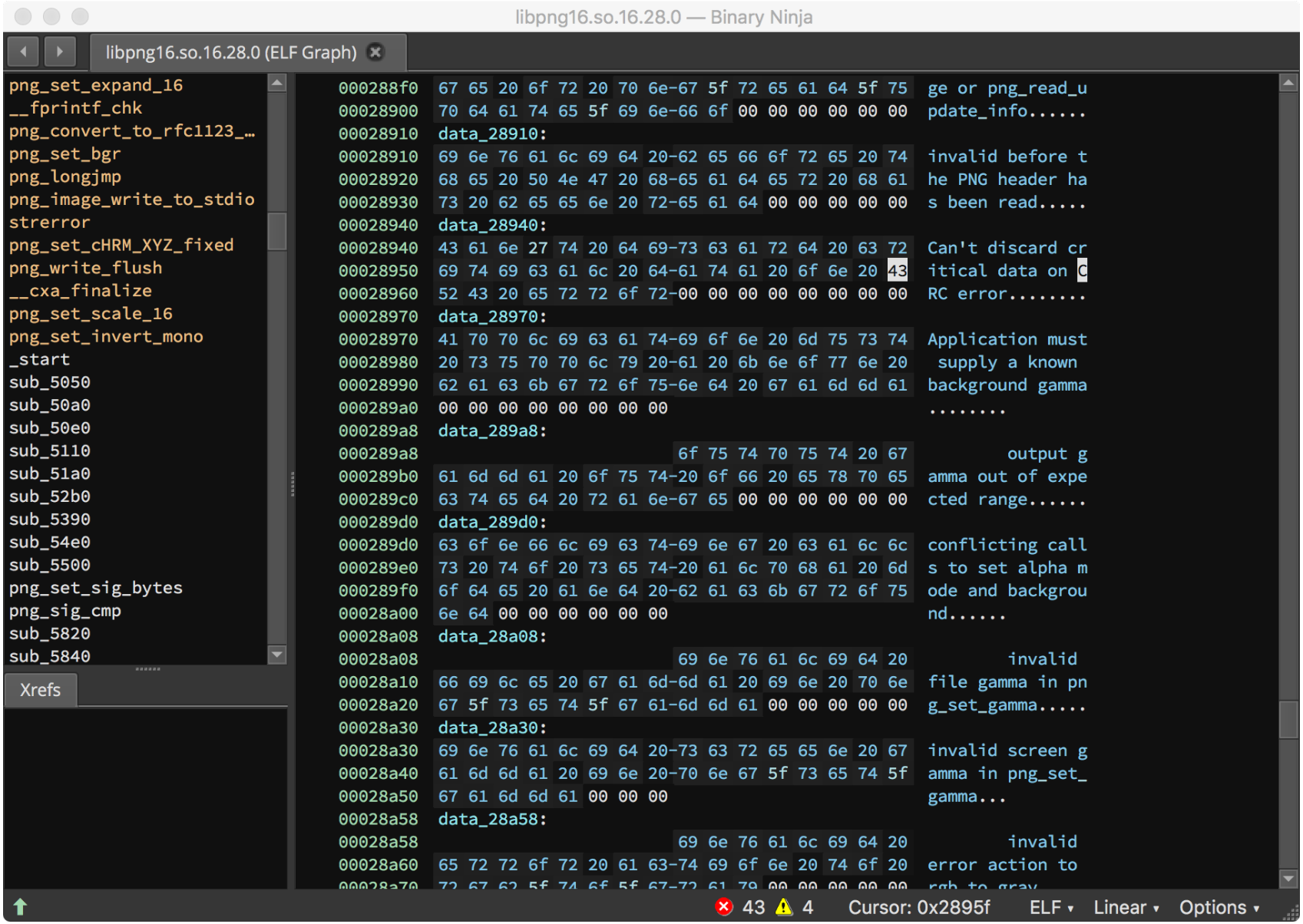
Libpng in Binary Ninja

Here we have the libpng library open in Binary Ninja.



Libpng in Binary Ninja

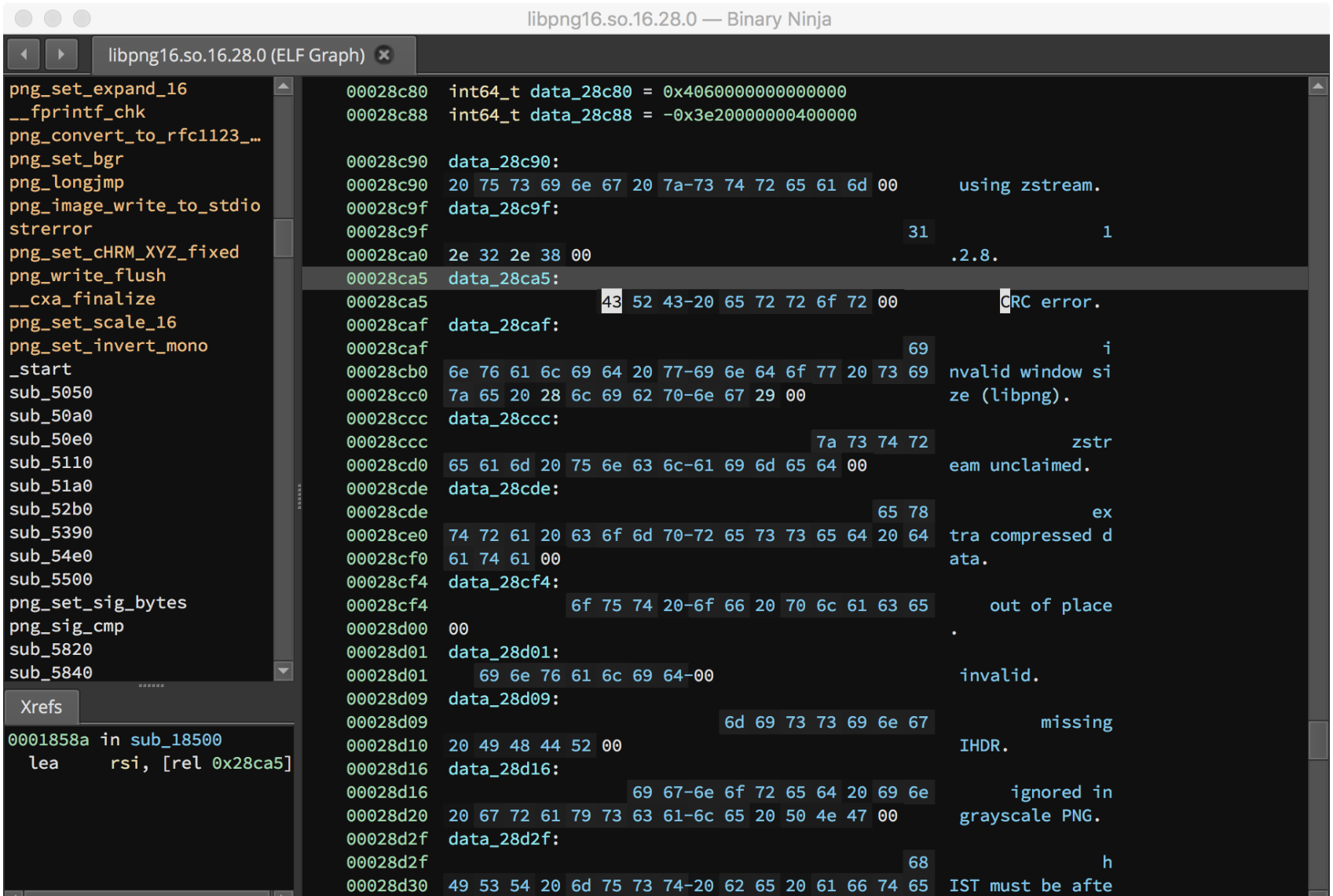
We begin searching for "CRC error". It looks like we found a substring.



FOR ALL SECURE

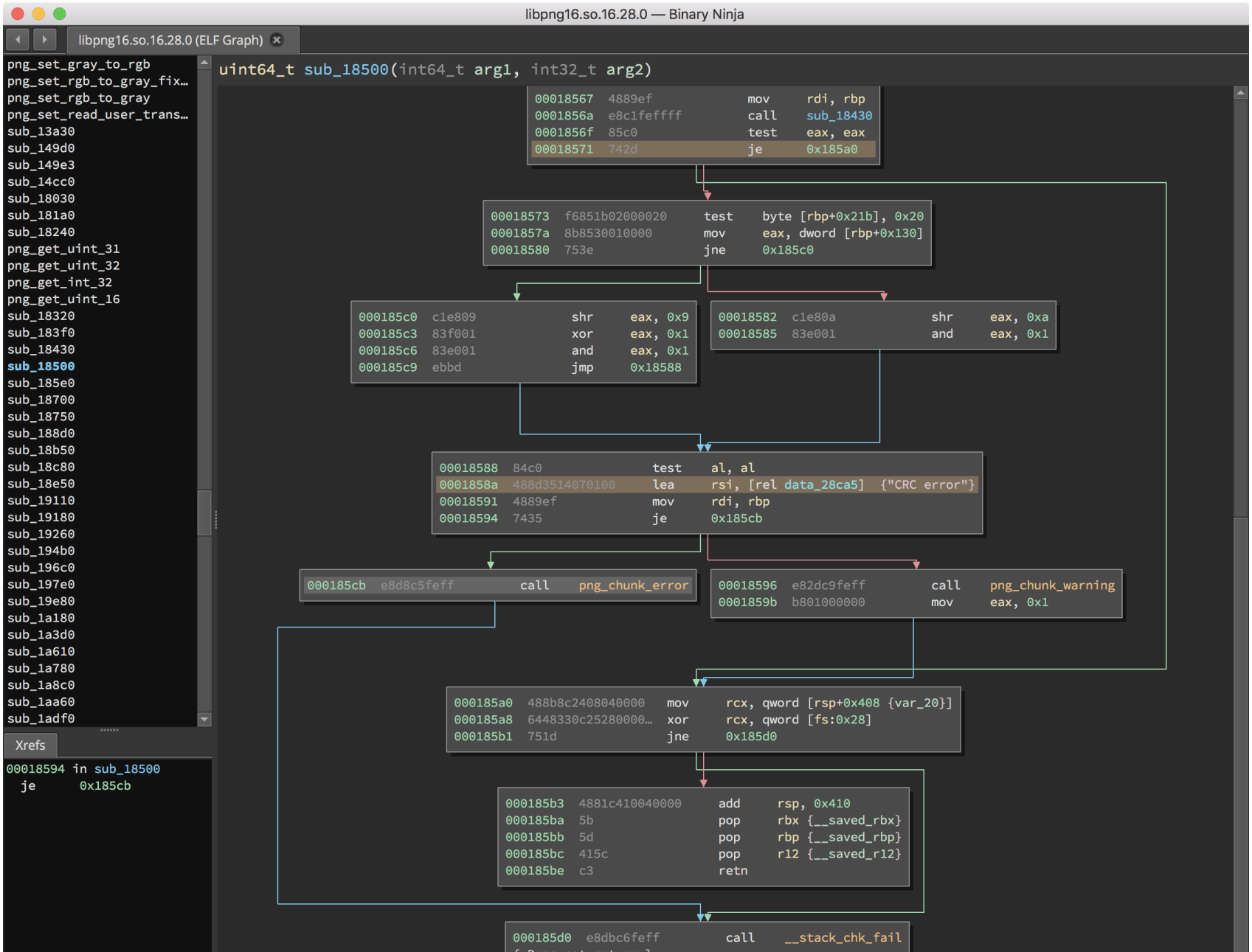
Searching for "CRC error"

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



FOR ALL SECURE

Patching out the CRC check



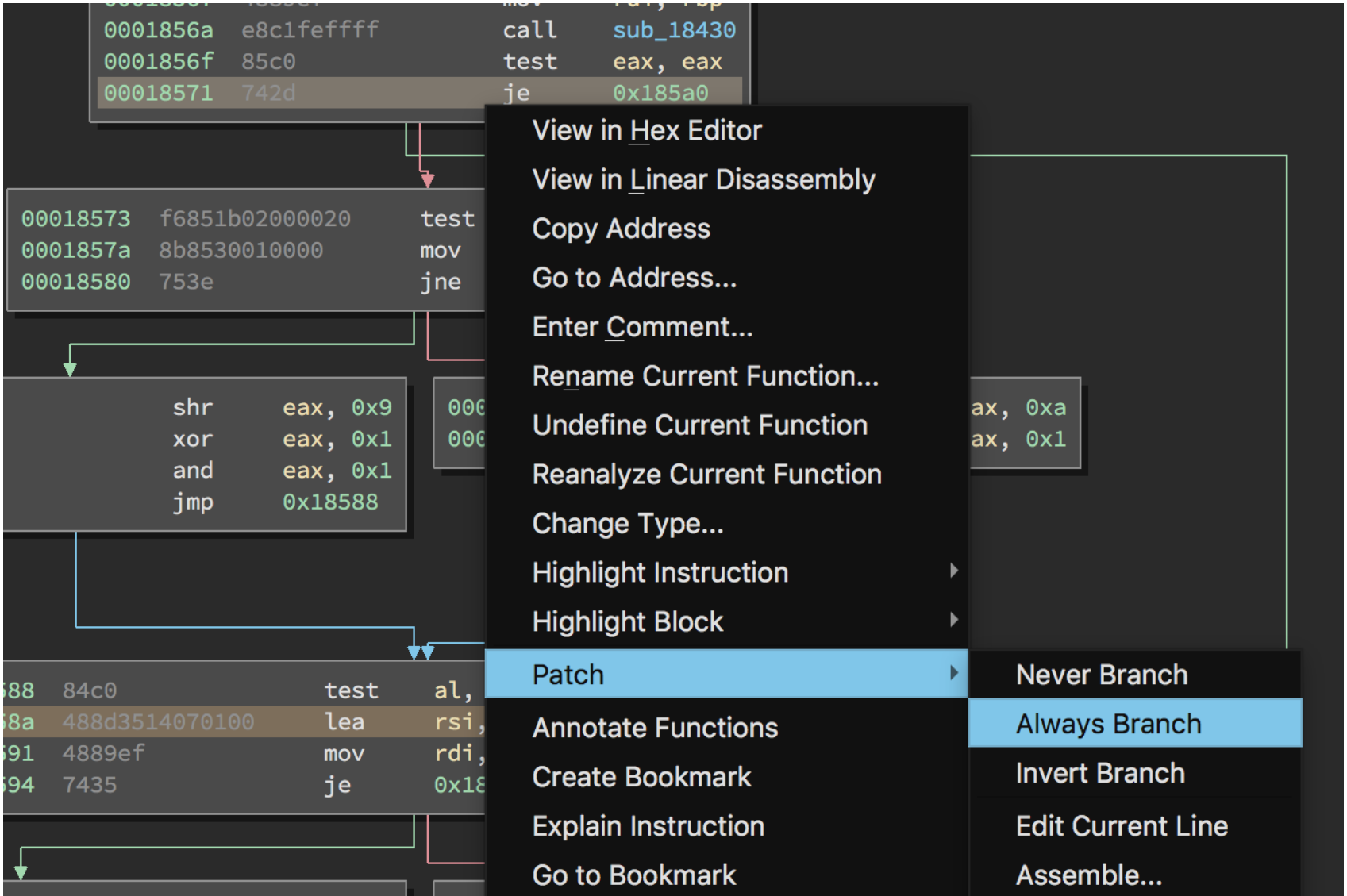
Patching out the CRC check

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.

Libpng in Binary Ninja

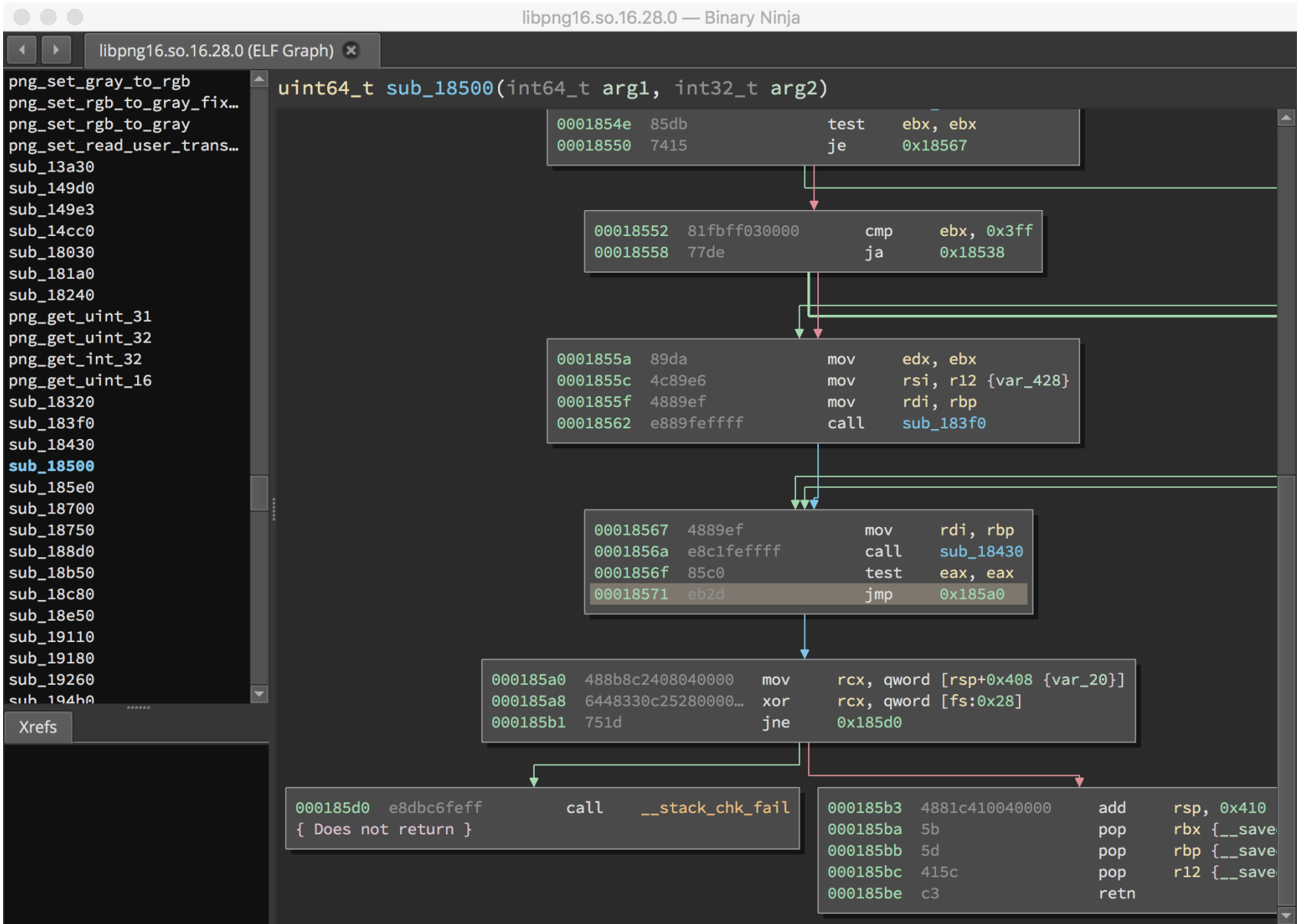
Right-click the instruction at `0x18571`, and choose `Patch -> Always Branch`.



FOR ALL SECURE

Libpng in Binary Ninja

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



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