binfmt_misc - execute it!!!

March 04, 2022

By Aaron Grothe
https://www.grothe.us

Introduction

The idea for this talk goes back to when I was playing around with some esoteric languages.

"How do you get your Linux box to run LOLCODE programs without having to always put the interpreter on the command line?"

This led me down a rabbit hole and led to this talk :-)

Introduction

Looking at the Linux Source code tree - https://github.com/torvalds/linux/tree/master/fs

Linux supports 6 type of formats directly

4 are executable types (a.out, elf, elf-fdpic, and flat)

2 are “interesting” to use (scripting and misc)

Introduction

Executable

- Binfmt_aout.c - a.out format
- Binfmt_elf.c - elf format
- Binfmt_elf_fdpic.c - elf format - position independent code
- Binfmt_flat.c - flat binaries

“Interesting”

- Binfmt_misc.c
- Binfmt_script.c

Is binfmt_script the answer?

Binfmt_script sounds promising. It is a way to call scripting options.

However it has some limitations which prevent it from working for us.

It expects every file to start with a #! - hash-bang and then the interperter

Why this won't work for me

An example LOLCode program

```
HAI 1.2
  CAN HAS STDIO?
  VISIBLE "HAI WORLD!!!1!"
KTHXBYE
```

By the definition of the language every LOLCode program starts with "HAI <Version>". There is no real way to get a # as the first line. The comments of LOLCode are in the form "BTW comment", so that doesn't work.

Why this won't work for me

Couple of potential workarounds

Modify LOLCode to support #! and change the interpreter and modify all the programs I'd like to play with (thank you no.)

Write a helper program with a #! at the top that will call the LOLcode interpreter and then run it. That sounds complicated.

Implement a naming convention, name all the files with an extension like .lol and writing an alias to execute it

Why this Won't work for me

All of them seem to be a bit of pain, what does this binfmt_misc do?

From the Wikipedia page for binfmt_misc - https://en.wikipedia.org/wiki/Binfmt_misc

binfmt_misc (Miscellaneous Binary Format) is a capability of the Linux kernel which allows arbitrary executable file formats to be recognized and passed to certain user space applications, such as emulators and virtual machines

That sounds promising :-)

Binfmt_misc

Is it enabled on my machine?

```
grothe@binfmt:~$ findmnt  binfmt_misc
TARGET                 SOURCE     FSTYPE   OPTIONS
/proc/sys/fs/binfmt_misc binfmt_misc binfmt_m
rw,nosuid,nodev,noexec,relat
```

It is enabled on my system. Good. There are other ways to validate it as well, but the above works pretty well.

LOLCode

So we'll try and run a simple LOLCode program

```
HAI 1.2
  CAN HAS STDIO?
  VISIBLE "HAI WORLD!!!1!"
KTHXBYE
```

Save it as hello.lol

LOLCode

We've got the program

#1. Lets run it with the lci interpreter

% lci ./hello.lol

Works.

LOLCode

We'll chmod it to be executable and try and run it and see what we get

```
% chmod +x ./hello.lol
% ./hello.lol
```

BOO!!! - Doesn't work

LOLCode

So we need to register the format with the binfmt_misc to know what to do with the executable.

Two ways to do this

#1. By file extension - for .lol files use lci to run them

#2. By magic header - for files that begin with HAI use lci to run them

LOLCode

Lets add the file extension to binfmt_misc

Once again referring to the wikipedia page

The register file contains lines which define executable types to be handled. Each line is of the form:

:name:type:offset:magic:mask:interpreter:flags

name is the name of the binary format.

LOLCode

type is either E or M

If it is E, the executable file format is identified by its filename extension: magic is the file extension to be associated with the binary format; offset and mask are ignored.

LOLCode

type is either E or M

If it is M, the format is identified by magic number at an absolute offset (defaults to 0) in the file and mask is a bitmask (defaults to all 0xFF) indicating which bits in the number are significant.
interpreter is a program that is to be run with the matching file as an argument.

flags (optional) is a string of letters, each controlling a certain aspect of interpreter invocation:

LOLCode

Example line to run lci for program ending with .lol

```
echo ':lolcat:E::lol::/usr/local/bin/lci:OC' | sudo tee /proc/sys/fs/binfmt_misc/register
```

Now we try and run hello.lol

```
./hello.lol
```

SUCCESS!!!!

LOLCode

Where is this registered in the system?

The answer as it is with almost all things Linux is in proc. In this case it is in the /proc/sys/fs/binfmt_misc folder

cd /proc/sys/fs/binfmt_misc
ls - shows all the formats currently registered with the system

We see an entry for lolcat - lets go ahead and cat that

cat lolcat

LOLCode

enabled
interpreter /usr/local/bin/lci
flags: OC
extension .lol

LOLCode by Magic

That's good and all but how would we do it by magic, instead of relying upon the magic header of the program.

First off we need to deregister our current registration for lolcat

% echo -1 | sudo tee /proc/sys/fs/binfmt_misc/lolcat

Now let's verify that isn't processing the .lol files anymore

% ./hello.lol

LOLCode by Magic

Now we need to register for the lolcat by magic

% echo ':lolcat:M::HAI::/usr/local/bin/lci:' | sudo tee /proc/sys/fs/binfmt_misc/register

Format looks similar to the extension except for the the 'M' instead of 'E', the Magic Value, and not having extension registered

Let's validate that works

% ./hello.lol

SUCCESS!!!

Next Step - Persistence

So now we've proven we can extend the system to support additional file formats, by either extension of magic headers.

One problem with this is. While we've been able to add these extensions they won't persist between reboots. For that we need to add the file to the system so it persists.

On debian this is under the /var/lib/binfmts folder

Next Step - Persistence

ls /var/lib/binfmts

Contains a bunch of registered entries for the system.

We'll want to add one for lolcode

vi /var/lib/binfmts/lolcode

Next Step - Persistence

Example File

lolcat
magic
0
HAI

/usr/local/bin/lci

Next Step - Persistence

Is the contents that we were registering with the system spread out to 10 lines

Lets go ahead and reboot the box and see if the change persists :-)

SUCCESS!!!

What's Next?

So we've shown how the system can be extended to run just about anything you want to.

Now we'll create a sparc64 executable on my x64 system and run it transparently to the user. This is where the fun starts.

This will show some of the real power of the binfmt_misc system.

Creating an executable

First need a program

```
#include <stdio.h>

int main ()
{
   printf ("hello world\n");
}
```

Creating an executable

Compile the program/Run and file the program

```
grothe@binfmt:~$ gcc a.c -l static

grothe@binfmt:~$ ./a.out
hello world

grothe@binfmt:~$ file a.out
a.out: ELF 64-bit LSB pie executable, x86-64, version 1
(SYSV), dynamically linked, interpreter
/lib64/ld-linux-x86-64.so.2,
BuildID[sha1]=96d0683b2e66b689e108d112bf022dfd952e
b157, for GNU/Linux 3.2.0, not stripped
```

Creating a executable

Compile the program/Run and file the program

```
grothe@binfmt:~$ gcc a.c -static

grothe@binfmt:~$ file a.out
a.out: ELF 64-bit LSB executable, x86-64, version 1
(GNU/Linux), statically linked,
BuildID[sha1]=42eda8241518f40dc462285f12e37e853d78
6ff2, for GNU/Linux 3.2.0, not stripped

grothe@binfmt:~$ ./a.out
hello world
```

Creating a Sparc64 executable

Compile the program, file the program

```
grothe@binfmt:~$ sparc64-linux-gnu-gcc-11 a.c -static

grothe@binfmt:~$ file a.out
a.out: ELF 64-bit MSB executable, SPARC V9, Sun UltraSPARC1 Extensions Required, relaxed memory ordering, version 1 (SYSV), statically linked, BuildID[sha1]=9230a5ea86c14c9bf7d45c450c15a99bdeba0d72, for GNU/Linux 3.2.0, not stripped
```

Transparent Execution

and run

```
grothe@binfmt:~$ ./a.out
hello world
```

SUCCESS!!

Transparent Execution

How did we do that?

The answer lies in /proc/sys/fs/binfmt_misc/qemu-sparc64

Time to take a look

% cat /proc/sys/fs/binfmt_misc/qemu-sparc64

Transparent Execution

Contents of the file

enabled
interpreter /usr/libexec/qemu-binfmt/sparc64-binfmt-P
flags: POC
offset 0
magic
7f454c4602020100000000000000000002002b
mask ffffffffffffffcfffffffffffffffffeffff

So it goes by the magic string for the executable and knows to pass it to the qemu interpreter

References

A Couple of References I need to point out.

“Using Go as a scripting language in Linux by Cloudflare”

https://blog.cloudflare.com/using-go-as-a-scripting-language-in-linux/

References

"Transparently running binaries from any architecture in Linux with QEMU and binfmt_misc"

https://ownyourbits.com/2018/06/13/transparently-running-binaries-from-any-architecture-in-linux-with-qemu-and-binfmt_misc/

References

Imgact_linux a project to add similar functionality to Mac OS X

https://github.com/georghe-crihan/imgact_linux

Wikipedia page for binfmt_misc

https://en.wikipedia.org/wiki/Binfmt_misc

Summary & Thank You

What've shown is that you can extend Linux to do almost anything you want to with binary programs. The ability of binfmt_misc allows you to call a userspace program to handle almost anything.

I'm hoping a few of you after this talk will go "Hey. What about this?"

Thanks for Listening,

Questions???