

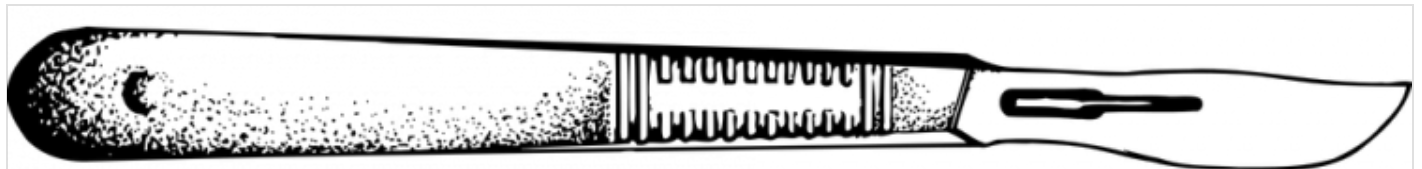


(<https://kazarin.online/>)

KAZARIN OnLine

(<https://kazarin.online/>)

Blog about IT, Me, education, etc...



Linux. Глубокое понимание работы системы. Часть 1 — трассировка библиотечных и системных вызовов.

20.07.2017 (https://kazarin.online/index.php/2017/07/20/linux-deep-in-system-work_part-1/) / Kazarin Kirill

(<https://kazarin.online/index.php/author/wpwebmaster/>) / DevOps

(<https://kazarin.online/index.php/category/it-blog/devops/>), IT Blog

(<https://kazarin.online/index.php/category/it-blog/>), Linux

(<https://kazarin.online/index.php/category/it-blog/linux/>), system engineer
(<https://kazarin.online/index.php/category/it-blog/system-engineer/>)

UPD: Запись перенесена из старого блога, опубликована в 2015 году.

Предисловие

Системному администратору зачастую очень важно понимать как устроена система, как настроить/отладить те ли иные вещи и как искать ошибки в работе. Если устройству и настройке можно достаточно легко научиться из различных руководств и справочных страниц, где написано что система сделана вот так вот, а настраивается она вот этими кнопками и командами, то вот с отладкой и поиском проблем зачастую возникают проблемы. Для многих, к сожалению, система представляет собой черный ящик, с входом и выходом и магией внутри. Я покажу как немного пролить свет на эту темноту внутри ящика.

Он будет развернут у меня на виртуальной машине в VirtualBox (<https://ru.wikipedia.org/wiki/VirtualBox>), с использованием снапшотов. Это нужно для того, чтобы смело экспериментировать и не бояться поломать свою систему.

Поехали...

Представим что нам просто выдали сервер с предустановленной ОС. Мы знаем лишь то, что «под капотом» стоит некий Linux. Необходимо провести разведку

```
uname -r
```

```
uname -a
```

мы узнаем версию ядра ОС и полную краткую информацию — версия ядра, то, что это ядро PAE (http://en.wikipedia.org/wiki/Physical_Address_Extension) (позволяет на 32 битной архитектуре видеть более 4 гб памяти. Видим что это Ubuntu (в основе)



([http://2.bp.blogspot.com/-](http://2.bp.blogspot.com/-VBXzcdUk2rs/V0Ggjn_wU5I/AAAAAAAAAe4/ysmxM9NnpGI/s1600/%D0%A1%D0%BD%D0%B8%D0%BC%D0%BE%D0%BA%2B%D1%8D%D0%BA%D1%80%D0%B0%D0%BD%D0%B0%2B%D0%BE%D1%82%2B2015-02-16%2B10%3A47%3A24.png)

[VBXzcdUk2rs/V0Ggjn_wU5I/AAAAAAAAAe4/ysmxM9NnpGI/s1600/%D0%A1%D0%BD%D0%B8%D0%BC%D0%BE%D0%BA%2B%D1%8D%D0%BA%D1%80%D0%B0%D0%BD%D0%B0%2B%D0%BE%D1%82%2B2015-02-16%2B10%3A47%3A24.png](http://2.bp.blogspot.com/-VBXzcdUk2rs/V0Ggjn_wU5I/AAAAAAAAAe4/ysmxM9NnpGI/s1600/%D0%A1%D0%BD%D0%B8%D0%BC%D0%BE%D0%BA%2B%D1%8D%D0%BA%D1%80%D0%B0%D0%BD%D0%B0%2B%D0%BE%D1%82%2B2015-02-16%2B10%3A47%3A24.png))

Выясним что же это за дистрибутив

```
lsb_release -a
```



(<http://4.bp.blogspot.com/-7NXCqIDUExc/V0Gg3hhTb7I/AAAAAAAAAfA/jqsVN53uKbA/s1600/%D0%A1%D0%BD%D0%B8%D0%BC%D0%BE%D0%BA%2B%D1%8D%D0%BA%D1%80%D0%B0%D0%BD%D0%B0%2B%D0%BE%D1%82%2B2015-02-16%2B10%3A48%3A41.png>)

LSB – Linux Standart Base. Он показывает нам что это дистрибутив Elenemtary OS, релиз 0.2 с кодовым именем Luna.

Этой информации достаточно для того чтобы начать поиск документации и описания различных знаний, специфичных для этой системы (например управление через GUI, хорошие практики настройки и эксплуатации и пр).

Однако, узнав что это по сути, один из стандартных Linux дистрибутивов мы можем смело сказать, что в составе уже есть минимально-необходимый комплект инструментов для отладки (универсальный для всех Linux систем, за исключением каких-нибудь очень специфичных). Продолжим знакомство теперь уже с ними:

Трассировщики и доп. утилиты

Для того чтобы наблюдать за программой которая обращается к ядру ос, ее библиотекам, системным вызовам, ресурсам и пр нам понадобятся трассировщики.

Покажу Вам парочку стандартных инструмнтов:

1. strace (<http://en.wikipedia.org/wiki/Strace>)или System trace – простой трассировщик системных вызовов.



(<http://1.bp.blogspot.com/-2XRFiaJo8nE/V0GinH3bkBI/AAAAAAAAAfM/EPt1TPTQs1Q/s1600/%D0%A1%D0%BD%D0%B8%D0%BC%D0%BE%D0%BA%2B%D1%8D%D0%BA%D1%80%D0%B0%D0%BD%D0%B0%2B%D0%BE%D1%82%2B2015-02-16%2B10%3A56%3A15.png>)

2. ltrace (<http://en.wikipedia.org/wiki/Ltrace>) – library tracer – трассировщик библиотек. Строит трассы вызова системных библиотек в ходе запуска программы.



(<http://4.bp.blogspot.com/-s1zftzIVyWY/V0GjEBrH64I/AAAAAAAAAfU/VKvXvxbhz6U/s1600/%D0%A1%D0%BD%D0%B8%D0%BC%D0>

%BED0%BA%2B%D1%8D%D0%BA%D1%80%D0%B0%D0%BD%D0%B0%2B%D0%BE%D1%82%2B2015-02-16%2B10%3A58%3A11.png)

3. `file` (https://ru.wikipedia.org/wiki/File_%28Unix%29)— позволяет заглядывать внутрь указанного файла и сигнатурным способом строит предположения о том, что же это за файл. В `unix` системах у файлов нет суффикса- расширения. Поэтому работа ведется по внутреннему содержанию, хотя ни одна файловая система не содержит никаких признаков.

Поэтому эта утилита очень полезна нам в работе, позволяя так или иначе заглядывать внутрь файла и узнавать, что же он из себя представляет.



(<http://4.bp.blogspot.com/-CtrEaynE6S0/VOGIId97GWI/AAAAAAAAAfo/-kzbDsSrWQ0/s1600/%D0%A1%D0%BD%D0%B8%D0%BC%D0%BE%D0%BA%2B%D1%8D%D0%BA%D1%80%D0%B0%D0%BD%D0%B0%2B%D0%BE%D1%82%2B2015-02-16%2B11%3A06%3A58.png>)

4. `which` (https://ru.wikipedia.org/wiki/File_%28Unix%29) – команда, позволяющая узнать где находится исполняемый файл той или иной программы.



([http://1.bp.blogspot.com/-](http://1.bp.blogspot.com/-Zpnesd0EQIA/VOGknei60fI/AAAAAAAAAfg/1M_3xgoxbEA/s1600/%D0%A1%D0%BD%D0%B8%D0%BC%D0%BE%D0%BA%2B%D1%8D%D0%BA%D1%80%D0%B0%D0%BD%D0%B0%2B%D0%BE%D1%82%2B2015-02-16%2B11%3A04%3A49.png)

[Zpnesd0EQIA/VOGknei60fI/AAAAAAAAAfg/1M_3xgoxbEA/s1600/%D0%A1%D0%BD%D0%B8%D0%BC%D0%BE%D0%BA%2B%D1%8D%D0%BA%D1%80%D0%B0%D0%BD%D0%B0%2B%D0%BE%D1%82%2B2015-02-16%2B11%3A04%3A49.png](http://1.bp.blogspot.com/-Zpnesd0EQIA/VOGknei60fI/AAAAAAAAAfg/1M_3xgoxbEA/s1600/%D0%A1%D0%BD%D0%B8%D0%BC%D0%BE%D0%BA%2B%D1%8D%D0%BA%D1%80%D0%B0%D0%BD%D0%B0%2B%D0%BE%D1%82%2B2015-02-16%2B11%3A04%3A49.png))

Обычно мы вводим команду в командном интерпретаторе и получаем ответ, не задумываясь- а это была встроенная команда или сторонняя программа, запуск которой был вызван вводом команды. А ведь то, что в одном дистрибутиве является встроенной командой оболочки, в другом может повлечь вызов программы. А место вызова программы может быть переопределено (в том числе и злоумышленником) и вместо простого вызова `sudo`, Вы запустите стороннюю утилиту, которая сохранит и отправит Ваш пароль кому-нибудь еще. `which` позволяет узнать где лежит та или иная программа.

5. «`type -a`» (http://en.wikipedia.org/wiki/Type_%28Unix%29) – команда, позволяющая узнать, чем для нас является та или иная команда- встроенной командой интерпретатора или вызовом внешней программы.



(<http://3.bp.blogspot.com/-kS0V8ITmLs8/V0Gm7ASpupI/AAAAAAAAAf8/ZyQBGZBUYqI/s1600/%D0%A1%D0%BD%D0%B8%D0%BC%D0%BE%D0%BA%2B%D1%8D%D0%BA%D1%80%D0%B0%D0%BD%D0%B0%2B%D0%BE%D1%82%2B2015-02-16%2B11%3A14%3A36.png>)

В зависимости от ее вывода, мы можем понять- что чем является. То, что лежит в:

- `/bin` – является неотъемлемыми системными компонентами, исполняемыми бинарными файлами, либо
- `/usr/bin` – прикладными, которые можно изъять из операционной системы без потери ее функциональности

либо является встроенной командой.

Теперь, раз мы уже так много знаем и умеем, давайте разберемся в самих файлах и программах чуть глубже- спросим у утилиты `file` про нее же саму и посмотрим, что она нам скажет?



(http://4.bp.blogspot.com/-aKfM8UrLgts/V0GoJyk-S8I/AAAAAAAAAgI/7bmKyI_3Vbk/s1600/%D0%A1%D0%BD%D0%B8%D0%BC%D0%BE%D0%BA%2B%D1%8D%D0%BA%D1%80%D0%B0%D0%BD%D0%B0%2B%D0%BE%D1%82%2B2015-02-16%2B11%3A19%3A54.png)

нам скажут что это:

- исполняемый файл (ищем в выводе команды слово «... executable ...»)
- нам скажут что он формата ELF – т.е. это не jpeg, не mpeg – т.е. не картинка, не видео (явно),
- это 32 битный, исполняемый код («... 32 bit LSB executable...»), который упакован в формат ELF (executable and linking format)
- версия формата («...Intel 80386, version 1 ...»)
- динамически слинкован с библиотеками («... dynamically linked...»)
- использует разделяемые библиотеки (http://www.opennet.ru/base/dev/shared_lib_intro.txt.html)
- создан для ядра Linux, начиная с версии 2.6.24
- и некоторая контрольная сумма исполняемого файла («...[sha1]=...»)
- и что из него удалена отладочная информация (stripped), которая могла бы быть использована отладчиком (<http://ru.wikipedia.org/wiki/%CE%E2%EB%E0%E4%F7%E8%EA>).

Отладчик необходим, если мы не можем понять, что происходит внутри программы а исходных текстов у нас нет. Или есть, но мы подозреваем, что компилятор (<http://ru.wikipedia.org/wiki/%CA%E0%E8%EF%E8%EB%FF%F2%E0%F0>) произвел некорректную оптимизацию, внося ошибки в результирующий бинарный код, хотя в тексте программы ошибок не было.

Но это приведет нас к выводу, что нам необходимы исходные тексты, умение их читать и компилировать. Либо если у нас проприетарное ПО, поработав отладчиком мы сможем найти сбойное место и отправить информацию производителю, чтобы они выпустили патч.

Продолжим исследование программы file

6. ldd (<http://xgu.ru/wiki/ldd>) – просмотрщик зависимостей запускаемой программы от различных разделяемых библиотек (http://www.opennet.ru/base/dev/shared_lib_intro.txt.html). Например, библиотеки от которых зависит программа file, т.е. кирпичики, которые начинают цепляться один за другой при запуске основной программы, т.е. программа file запускаться и работать вообще не сможет



(<http://2.bp.blogspot.com/-XudtZT7r74k/V0GsKmSFliI/AAAAAAAAAgk/Qwz8aoKIOrQ/s1600/%D0%A1%D0%BD%D0%B8%D0%BC%D0%BE%D0%BA%2B%D1%8D%D0%BA%D1%80%D0%B0%D0%BD%D0%B0%2B%D0%BE%D1%82%2B2015-02-16%2B11%3A37%3A03.png>)

XudtZT7r74k/V0GsKmSFliI/AAAAAAAAAgk/Qwz8aoKIOrQ/s1600/%D0%A1%D0%BD%D0%B8%D0%BC%D0%BE%D0%BA%2B%D1%8D%D0%BA%D1%80%D0%B0%D0%BD%D0%B0%2B%D0%BE%D1%82%2B2015-02-16%2B11%3A37%3A03.png)

- linux-gate.so.1 – библиотека системных вызовов ядра ОС
- libmagic.so.1 – ее собственная библиотека, содержащая сигнатуры для определения типов исследуемых файлов.
- libc.so.6 – библиотека языка C нужна ей, т.к. будучи написанной на C она использует ее функции для банального открытия файлов
- libz.so.1 – нужна для работы с сжатыми файлами (архивами)
- /lib/ld-linux.so.2 – библиотека линковщик- умеет/позволяет «прицеплять» другие библиотеки. Это и есть часть того Loader-а или загрузчика, который строит окружение программы-процесса, помещая в память необходимые компоненты. Саму ОС мало интересует от чего зависит программа- она просто

выделяет память, создает контекст процесса
(<http://emanual.ru/download/9532.html>) и передает управление этой
библиотеке, которая и делает за нее грязную работу.

После этого встает вопрос- а как же библиотека загружает другие библиотеки если
ее тоже кто-то должен загрузить и загрузить ее библиотеки?

Вспомним, что библиотеки это тоже файлы-программы. Между ними нет никакой
разницы кроме одного маленького исключения. Посмотрим его (тут я переключусь на
свою локальную консоль, ибо так будет удобней):



(<http://3.bp.blogspot.com/-cmqR1yLfNcg/V0GzGpX5skI/AAAAAAAAAH/3vyqNc6q7Ho/s1600/%D0%A1%D0%BD%D0%B8%D0%BC%D0%BE%D0%BA%2B%D1%8D%D0%BA%D1%80%D0%B0%D0%BD%D0%B0%2B%D0%BE%D1%82%2B2015-02-16%2B12%3A06%3A34.png>)

Посмотрим, что из себя представляет наша «магическая» библиотека – оказывается
это символическая ссылка на библиотеку определенной версии в системе. Тогда мы
даем команду `file` с ключом `-L` – сходи по ссылке и выясни таки, что там валяется.

Он показывает нам, что это тоже бинарный файл, формата ELF и т.д. и т.п. Только в
отличие от программы (смотрим пример для самой программы `file`), библиотека
является не `executable` объектом, а `shared object` (выделил на скриншоте). Что это
означает? – Запустить программу можно, а вот библиотеку – нет. В ней нет точки
входа/старта. И если мы посмотрим остальные библиотеки, увидим такую же картину.



(<http://3.bp.blogspot.com/-9w0u-43UzeA/VOHt-pnIq8I/AAAAAAAAAhY/5uUGzvGdIR0/s1600/%D0%A1%D0%BD%D0%B8%D0%BC%D0%BE%D0%BA%2B%D1%8D%D0%BA%D1%80%D0%B0%D0%BD%D0%B0%2B%D0%BE%D1%82%2B2015-02-16%2B16%3A17%3A54.png>)

И у всех у них в свою очередь тоже есть зависимости, потому что между программой и библиотекой от которой она зависит, нет почти никакой разницы. Посмотрим `ldd` для библиотеки `C` — она в свою очередь зависит от другой библиотеки, а вот у нее уже нет никаких зависимостей, т.к. она статически слинкована — т.е. все что необходимо для нее, уже входит в ее состав. Т.е. она не зависит ни от каких других библиотек и может быть использована напрямую.



([http://3.bp.blogspot.com/-](http://3.bp.blogspot.com/-yQA8wRu1_4U/VOHu1ixbzbI/AAAAAAAAAhk/p_Y0eS4_86o/s1600/%D0%A1%D0%BD%D0%B8%D0%BC%D0%BE%D0%BA%2B%D1%8D%D0%BA%D1%80%D0%B0%D0%BD%D0%B0%2B%D0%BE%D1%82%2B2015-02-16%2B16%3A21%3A33.png)

[yQA8wRu1_4U/VOHu1ixbzbI/AAAAAAAAAhk/p_Y0eS4_86o/s1600/%D0%A1%D0%BD%D0%B8%D0%BC%D0%BE%D0%BA%2B%D1%8D%D0%BA%D1%80%D0%B0%D0%BD%D0%B0%2B%D0%BE%D1%82%2B2015-02-16%2B16%3A21%3A33.png](http://3.bp.blogspot.com/-yQA8wRu1_4U/VOHu1ixbzbI/AAAAAAAAAhk/p_Y0eS4_86o/s1600/%D0%A1%D0%BD%D0%B8%D0%BC%D0%BE%D0%BA%2B%D1%8D%D0%BA%D1%80%D0%B0%D0%BD%D0%B0%2B%D0%BE%D1%82%2B2015-02-16%2B16%3A21%3A33.png))

Вот вам ответ как она загружается- она не зависит от других библиотек и загружается сама сразу. Один нюанс – раз команда `file` говорит что `/lib/ld-linux.so.2` динамически слинкована, а `ldd` говорит что статически, значит кто-то из них врет. Скорее всего врет `file`, т.к. она строит предположения на основе сигнатур, а `ldd` четко проверяет зависимости.



([http://1.bp.blogspot.com/-FTScIPj-](http://1.bp.blogspot.com/-FTScIPj-LqI/VOHvXfhNwVI/AAAAAAAAAhs/X2I1YECvPRM/s1600/%D0%A1%D0%BD%D0%B8%D0%BC%D0%BE%D0%BA%D1%8D%D0%BA%D1%80%D0%B0%D0%BD%D0%B0%2B%D0%BE%D1%82%2B2015-02-16%2B16%3A23%3A47.png)

[LqI/VOHvXfhNwVI/AAAAAAAAAhs/X2I1YECvPRM/s1600/%D0%A1%D0%BD%D0%B8%D0%BC%D0%BE%D0%BA%D1%8D%D0%BA%D1%80%D0%B0%D0%BD%D0%B0%2B%D0%BE%D1%82%2B2015-02-16%2B16%3A23%3A47.png](http://1.bp.blogspot.com/-FTScIPj-LqI/VOHvXfhNwVI/AAAAAAAAAhs/X2I1YECvPRM/s1600/%D0%A1%D0%BD%D0%B8%D0%BC%D0%BE%D0%BA%D1%8D%D0%BA%D1%80%D0%B0%D0%BD%D0%B0%2B%D0%BE%D1%82%2B2015-02-16%2B16%3A23%3A47.png))

Кстати, существуют и просто программы, которые статически линкуются. Например это сам командный интерпретатор – он не имеет зависимостей при запуске, на тот случай, если все сломалось и нам надо загрузиться в ограниченном режиме и починить систему. К тому же, вам должно быть известно, что когда запускается система, она использует сценарии, написанные на языке командного интерпретатора. При установке, администратор мог вынести некоторые каталоги на отдельные разделы, которые не могут быть подмонтированы в самом начале. А это значит, что если сценарий использует программы и библиотеки, расположенные на этих разделах, он не сможет их запустить.

Либо более простая ситуация- произошло что-то плохое с диском, что повлияло на блок, в котором располагалась часть кода библиотеки. В следствии чего она не смогла быть корректно запущена. Посмотрим на наш командный интерпретатор



([http://3.bp.blogspot.com/--](http://3.bp.blogspot.com/--GP1SpIRx1U/V0H4UMPr03I/AAAAAAAAAh8/Bqz5G0WN414/s1600/%D0%A1%D0%BD%D0%B8%D0%BC%D0%BE%D0%BA%2B%D1%8D%D0%BA%D1%80%D0%B0%D0%BD%D0%B0%2B%D0%BE%D1%82%2B2015-02-16%2B17%3A01%3A58.png)

[GP1SpIRx1U/V0H4UMPr03I/AAAAAAAAAh8/Bqz5G0WN414/s1600/%D0%A1%D0%BD%D0%B8%D0%BC%D0%BE%D0%BA%2B%D1%8D%D0%BA%D1%80%D0%B0%D0%BD%D0%B0%2B%D0%BE%D1%82%2B2015-02-16%2B17%3A01%3A58.png](http://3.bp.blogspot.com/--GP1SpIRx1U/V0H4UMPr03I/AAAAAAAAAh8/Bqz5G0WN414/s1600/%D0%A1%D0%BD%D0%B8%D0%BC%D0%BE%D0%BA%2B%D1%8D%D0%BA%D1%80%D0%B0%D0%BD%D0%B0%2B%D0%BE%D1%82%2B2015-02-16%2B17%3A01%3A58.png))

И увидим что он так же имеет зависимости. Но как же так? Ведь мы не сможем без переинсталляции починить систему в случае такого краха. Нет, сможем. Потому что в каталоге `/bin` лежит версия интерпретатора `sh`, статически скомпилированная. Она называется `static-sh`. На самом деле это символическая ссылка на другой инструмент – `busybox` (<http://ru.wikipedia.org/wiki/BusyBox>) – эдакий швейцарский нож. В нем реализовано много других программ, необходимых для починки ОС.



(http://3.bp.blogspot.com/-qe1nmP4_wIY/V0H-lrZULHI/AAAAAAAAiM/d76n00KlcYY/s1600/%D0%A1%D0%BD%D0%B8%D0%BC%D0%BE%D0%BA%2B%D1%8D%D0%BA%D1%80%D0%B0%D0%BD%D0%B0%2B%D0%BE%D1%82%2B2015-02-16%2B17%3A28%3A45.png)

Мы видим, что он не от чего не зависит — вот он наш минимальный инструмент, который всегда запустится.

Пример использования 1. окружение

Теперь давайте попробуем применить полученные знания на практике. Мы уже начали исследовать саму программу `file` — с ней и продолжим. Итак, вспомним вновь основные приемы — смотрим что это, где лежит и от чего зависит



(<http://2.bp.blogspot.com/-2fc4bAcu-Ro/VOICgmDKn5I/AAAAAAAAAiY/ugBwppoIqF8/s1600/%D0%A1%D0%BD%D0%B8%D0%BC%D0%BE%D0%BA%2B%D1%8D%D0%BA%D1%80%D0%B0%D0%BD%D0%B0%2B%D0%BE%D1%82%2B2015-02-16%2B17%3A45%3A26.png>)

Теперь немного поработаем ей- попробуем определить типы различных файлов



(<http://1.bp.blogspot.com/-SE3rvc4yamk/VOIC8EVdGDI/AAAAAAAAAig/vYh9IJM8es0/s1600/%D0%A1%D0%BD%D0%B8%D0%BC%D0%BE%D0%BA%2B%D1%8D%D0%BA%D1%80%D0%B0%D0%BD%D0%B0%2B%D0%BE%D1%82%2B2015-02-16%2B17%3A45%3A26.png>)

Как это работает- они либо смотрит, чем является файл на файловой системе, либо заглядывает внутрь и пытается выяснить тип файла сигнатурным способом.



(http://3.bp.blogspot.com/-7J-ZQxsjpTs/VOIEr9M_OWI/AAAAAAAAAis/b6YVwR9wChg/s1600/%D0%A1%D0%BD%D0%B8%D0%BC%D0%BE%D0%BA%2B%D1%8D%D0%BA%D1%80%D0%B0%D0%BD%D0%B0%2B%D0%BE%D1%82%2B2015-02-16%2B17%3A54%3A50.png)

Мы уже видели, что она зависит от библиотеки `libmagic`. Логично было бы убедиться в том, что она использует этот сигнатурный способ на практике, а не основываясь на моих словах. Сделать мы это можем путем предположения, что именно эта библиотека и отвечает за анализ сигнатур (ведь остальные являются стандартными системными библиотеками, а эта как то выделяется). Аналогичным способом мы могли бы выяснить что `Libz` отвечает за работу с архивами. Как нам это сделать?

Еще в самом начале мы выяснили, что у нас система основана на `ubuntu`, а значит, поковырявшись в справочной информации мы узнаем что это ветка развития `Debian` а значит использует его менеджер пакетов `dpkg` (<http://ru.wikipedia.org/wiki/Dpkg>) (как раз та самая дистрибутив-специфичная информация).

Так давайте спросим- из какого пакета была инсталлирована та или иная библиотека.

```
$ dpkg -S /usr/lib/i386-linux-gnu/libmagic.so.1
```

Он скажет что она была инсталлирована из пакета `libmagic1` для платформы `intel 386`. Спросим теперь что это за зверь?

```
dpkg -s libmagic1
```

Он нам ответить что это библиотека (читаем описание на агл), может быть использована для классификации файлов с использованием специального теста «магических чисел» + дается ссылка на страницу проекта.



(<http://4.bp.blogspot.com/--zypEHPaskQ/V0IFl0XqY9I/AAAAAAAAAi0/38Ko7fj0P88/s1600/%D0%A1%D0%BD%D0%B8%D0%BC%D0%BE%D0%BA%2B%D1%8D%D0%BA%D1%80%D0%B0%D0%BD%D0%B0%2B%D0%BE%D1%82%2B2015-02-16%2B17%3A58%3A36.png>)

Все становится более-менее понятно. И видим что программа `file` не может жить без библиотеки, а значит пакет с программой зависит от пакета с библиотекой. Вот вам живая иллюстрация зависимостей программного обеспечения.

Что мы видим? что в этом пакете не только библиотека но еще и пара файлов (и их контрольные суммы):

Conffiles:

`/etc/magic 272913026300e7ae9b5e2d51f138e674`

`/etc/magic.mime 272913026300e7ae9b5e2d51f138e674`

Хотя если это не очевидно, то мы можем запросить список файлов, входящих в пакет.

```
dpkg -L libmagic1
```



(http://4.bp.blogspot.com/-FFZqfjsTim8/VOIGb0v_J4I/AAAAAAAAAi8/QA00IZRq3UQ/s1600/%D0%A1%D0%BD%D0%B8%D0%BC%D0%BE%D0%BA%2B%D1%8D%D0%BA%D1%80%D0%B0%D0%BD%D0%B0%2B%D0%BE%D1%82%2B2015-02-16%2B18%3A02%3A14.png)

Тут сама библиотека, каталоги, отладочная информация и справочные руководства man! (Страница 5: /usr/share/man/man5/magic.5.gz) – значит мы можем почитать описание этой библиотеки:

```
man 5 magic
```



(http://1.bp.blogspot.com/-JPnVU_jbaIo/VOIHFIGIM9I/AAAAAAAAAJQ/U03D0gR3lbg/s1600/%D0%A1%D0%BD%D0%B8%D0%BC%D0%BE%D0%BA%2B%D1%8D%D0%BA%D1%80%D0%B0%D0%BD%D0%B0%2B%D0%BE%D1%82%2B2015-02-16%2B18%3A05%3A00.png)

Тут мы можем подробно почитать о том что это и зачем нужно. Из чего он состоит и как утилита file его использует. Но нам же интересно и на сам файл с сигнатурами посмотреть?

```
file -L /usr/share/misc/magic.mgc
```



(<http://3.bp.blogspot.com/-SrGmqXazLqU/VOIHmgshSJI/AAAAAAAAAJY/2SPikcdM3QE/s1600/%D0%A1%D0%BD%D0%B8%D0%BC%D0%BE%D0%BA%2B%D1%8D%D0%BA%D1%80%D0%B0%D0%BD%D0%B0%2B%D0%BE%D1%82%2B2015-02-16%2B18%3A07%3A15.png>)

Сама утилита файл подтверждает полученную ранее информацию- да, это файл, содержащий те самые магические числа.

А мы можем его посмотреть? Попробуем, воспользовавшись утилитой `less` для просмотра текстовых файлов



(<http://2.bp.blogspot.com/-q6M0sJxabHs/VOIIF5HpEuI/AAAAAAAAAjk/qDcJfdRw-3E/s1600/%D0%A1%D0%BD%D0%B8%D0%BC%D0%BE%D0%BA%2B%D1%8D%D0%BA%D1%80%D0%B0%D0%BD%D0%B0%2B%D0%BE%D1%82%2B2015-02-16%2B18%3A09%3A19.png>)

К сожалению, она нам говорит что не уверена что он текстовый и даже если мы прикажем его посмотреть- не сможет это сделать. А может утилита `cat` нам поможет? Нет, увы- она замусорит нам консоль.



(http://1.bp.blogspot.com/-nRNWRqb9DzE/VOIIeegSIoI/AAAAAAAAAjs/lsY_LYZMU6I/s1600/%D0%A1%D0%BD%D0%B8%D0%BC%D0%BE%D0%BA%2B%D1%8D%D0%BA%D1%80%D0%B0%D0%BD%D0%B0%2B%D0%BE%D1%82%2B2015-02-16%2B18%3A10%3A50.png)

То есть очевидно что там все же не совсем текстовое содержимое, а скорее сигнатуры и их описание. Отлично- вот как много мы уже выяснили, зная лишь про одну утилиту `file` и исследуя ее окружение.

Теперь, давайте попробуем посмотреть то, что она делает и как она это делает.

Пример использования 2. Трассировка

Итак, посмотрим за поведением утилиты `file` во время ее работы, для этого используем упомянутые выше утилиты `strace` и `ltrace` и построим трассы вызовов.

```
strace file /usr/bin/file
```



(<http://3.bp.blogspot.com/-JlacPTrnB8A/VOJbPLWXSoI/AAAAAAAAAKI/1lSgIvel0ds/s1600/%D0%A1%D0%BD%D0%B8%D0%BC%D0%BE%D0%BA%2B%D1%8D%D0%BA%D1%80%D0%B0%D0%BD%D0%B0%2B%D0%BE%D1%82%2B2015-02-17%2B00%3A03%3A43.png>)

Получим большой-большой вывод. Ознакомимся с ним — что мы видим:

- выполняется системный вызов
(http://ru.wikipedia.org/wiki/%D1%E8%F1%F2%E5%EC%ED%FB%E9_%E2%FB%E7%EE%E2%D1%E8%F1%F2%E5%EC%ED%FB%E9_%E2%FB%E7%EE%E2))
execve, запускающий программу «/usr/bin/file», с аргументом «/usr/bin/file»
- системный вызов access, который узнает- если ли такой файл (искомый программой)
- системный вызов open — попытка открыть найденный файл
- и т.д.

В конечном итоге это превращается в попытку открыть библиотеки от которых зависит программа и считать их содержимое.



(http://4.bp.blogspot.com/-1Y3Yqjwutz0/V0Jc3dM5H1I/AAAAAAAAAKU/_NQDZUSQFXY/s1600/%D0%A1%D0%BD%D0%B8%D0%BC%D0%BE%D0%BA%2B%D1%8D%D0%BA%D1%80%D0%B0%D0%BD%D0%B0%2B%D0%BE%D1%82%2B2015-02-17%2B00%3A10%3A46.png)

Т.е. мы уже видим типичный запуск любой программы (а если видите в первый раз-запоминайте) – поиск и открытие всех библиотек и др составных частей рабочей программы, т.е. формирование среды ее работы.

Далее, программа открывает свой конфигурационный файл и файл, который как мы предполагали, содержит сигнатуры



(http://2.bp.blogspot.com/-brvLywvx64A/V0Jd8oYYi9I/AAAAAAAAAKg/Q04E_Osu1H8/s1600/%D0%A1%D0%BD%D0%B8%D0%BC%D0

%BED0%BA%2B%D1%8D%D0%BA%D1%80%D0%B0%D0%BD%D0%B0%2B%D0%BE%D1%82%2B2015-02-17%2B00%3A15%3A21.png)

А дальше она пытается открыть и обработать тот файл, который мы попросили проанализировать:

Разбирает список аргументов



(http://3.bp.blogspot.com/-KMZEaKfw4SA/VOJesEW1E1I/AAAAAAAAAko/TDg1BNh_HG8/s1600/%D0%A1%D0%BD%D0%B8%D0%BC%D0%BE%D0%BA%2B%D1%8D%D0%BA%D1%80%D0%B0%D0%BD%D0%B0%2B%D0%BE%D1%82%2B2015-02-17%2B00%3A18%3A33.png)

открывает файл



(http://2.bp.blogspot.com/-fz4X9k-CRoU/VOJe0fx6-wI/AAAAAAAAAkW/cZhsVFNRwVg/s1600/%D0%A1%D0%BD%D0%B8%D0%BC%D0%BE%D0%BA%2B%D1%8D%D0%BA%D1%80%D0%B0%D0%BD%D0%B0%2B%D0%BE%D1%82%2B2015-02-17%2B00%3A19%3A08.png)

читает его заголовок



(<http://4.bp.blogspot.com/-D85vyFr0iWQ/V0Je-ptVS6I/AAAAAAAAAk4/8j5AT0bz9Vs/s1600/%D0%A1%D0%BD%D0%B8%D0%BC%D0%BE%D0%BA%2B%D1%8D%D0%BA%D1%80%D0%B0%D0%BD%D0%B0%2B%D0%BE%D1%82%2B2015-02-17%2B00%3A19%3A47.png>)

находит в заголовке какие-то только ей понятные искомые структуры данных



(http://1.bp.blogspot.com/-L52lDC0Eaz8/V0JfH-ALRzI/AAAAAAAAA1A/UXYVP6L6A_I/s1600/%D0%A1%D0%BD%D0%B8%D0%BC%D0%BE%D0%BA%2B%D1%8D

%D0%BA%D1%80%D0%B0%D0%BD%D0%B0%2B%D0%BE%D1%82%2B2015-02-17%2B00%3A20%3A22.png)

Далее, считав что нужно, она его анализирует (это мы к сожалению не увидим, т.к. обработка происходит внутри программы без использования системных вызовов) и в конце концов выдает результат:



(<http://4.bp.blogspot.com/-xcuu01ryc-Q/V0JfguRqNII/AAAAAAAAAI/gtV8PyPsHyA/s1600/%D0%A1%D0%BD%D0%B8%D0%BC%D0%BE%D0%BA%2B%D1%8D%D0%BA%D1%80%D0%B0%D0%BD%D0%B0%2B%D0%BE%D1%82%2B2015-02-17%2B00%3A22%3A02.png>)

Дополнительные детали работы программы можно увидеть, построив трассу вызова библиотек – например, какие библиотечные функции использует наша программа:

```
ltrace file /usr/bin/file
```



(<http://2.bp.blogspot.com/-XShvCriIgFI/V0Ji-ab0txI/AAAAAAAAAU/YoOnbJENyrU/s1600/%D0%A1%D0%BD%D0%B8%D0%BC%D0%BE%D0%BA%2B%D1%8D%D0%BA%D1%80%D0%B0%D0%BD%D0%B0%2B%D0%BE%D1%82%2B2015-02-17%2B00%3A25%3A59.png>)

- Устанавливает текущие настройки локали
- Ищет первое вхождение символа / в строку аргументов, чтобы установить точку отсчета пути
- дальше, загрузив библиотеку magic, с помощью вызовов open и load загружает информацию из сигнатурного файла и с помощью своих функций начинает обработку
- найдя необходимую информацию в файле сигнатур и проанализировав обрабатываемый файл, она вызовом magic_file «выковыривает» описание и

вызовом `puts` выдает нам на экран результат



(<http://4.bp.blogspot.com/-S1CogYt7xTY/V0JkTAwXoBI/AAAAAAAAAlg/bMIbLaGm8qY/s1600/%D0%A1%D0%BD%D0%B8%D0%BC%D0%BE%D0%BA%2B%D1%8D%D0%BA%D1%80%D0%B0%D0%BD%D0%B0%2B%D0%BE%D1%82%2B2015-02-17%2B00%3A42%3A25.png>)

Итого, путем всех этих препарирований, мы четко для себя выяснили схему работы программы:

- загрузиться в память и загрузить все библиотеки
- получить список аргументов и разобрать его
- найти анализируемый файл
- открыть и прочитать свои `magic` файлы
- обратиться к своей библиотеке
- проанализировать информацию
- напечатать результат

А мы все увидели это с помощью трассировщиков.

Работа со справочной информацией

Но откуда нам знать что делают те или иные функции? Очень просто- почитать встроенную справку. Например мы видели функцию `puts`, почитаем справку про нее:

```
man 3 puts
```



(<http://1.bp.blogspot.com/-DBLOZK0yt1o/VOJlqyGDIBI/AAAAAAAAAIs/Y5MEiLgcxVU/s1600/%D0%A1%D0%BD%D0%B8%D0%BC%D0%BE%D0%BA%2B%D1%8D%D0%BA%D1%80%D0%B0%D0%BD%D0%B0%2B%D0%BE%D1%82%2B2015-02-17%2B00%3A48%3A16.png>)

Откуда я узнал, что нужно открыть именно 3 раздел руководства? прочитал справку о справке – map map, где описана структура справочного руководства.

Но если мы попробуем найти хоть какую-то информацию о наших магических функциях, нас ждет разочарование:



(<http://4.bp.blogspot.com/-tJd2xZe6Tck/VOJm06Egv1I/AAAAAAAAA10/CaRJJrQ-OoM/s1600/%D0%A1%D0%BD%D0%B8%D0%BC%D0%BE%D0%BA%2B%D1%8D%D0%BA%D1%80%D0%B0%D0%BD%D0%B0%2B%D0%BE%D1%82%2B2015-02-17%2B00%3A50%3A35.png>)

Почему так произошло? Да очень просто- установив пакет с библиотекой, мы установили минимально необходимый для запуска программы набор компонентов, оставив в стороне набор компонентов разработчика, такой как заголовочные файлы и страницы справочного руководства. Их так же можно доустановить из пакетов так же, как мы ставим все прочее программное обеспечение.

Попробуем провести поиск. Мы знаем что это функция входит в «магическую» библиотеку. Последовательно проследим от каких библиотек зависит запускаемая нами утилита `file` (ведь в ее трассировке мы увидели этот библиотечный вызов), в каком пакете содержится эта библиотека, есть ли для нее версия для разработчиков и т.д.



(http://1.bp.blogspot.com/-79sZVNc40bU/V0Jx6_TL9hI/AAAAAAAAAmE/oCNclhDaEzA/s1600/%D0%A1%D0%BD%D0%B8%D0%BC%D0%BE%D0%BA%2B%D1%8D%D0%BA%D1%80%D0%B0%D0%BD%D0%B0%2B%D0%BE%D1%82%2B2015-02-17%2B01%3A40%3A35.png)

Среди списка рекомендуемых пакетов в описании увы нет ничего похожего (например одноименные библиотеки с суффиксами `doc` или `dev`), поэтому проведем поиск по имени пакета среди доступных:



(<http://2.bp.blogspot.com/-hRvCejk0QjE/V0JzMM5kP-I/AAAAAAAAAmQ/AUT8rwtatTk/s1600/%D0%A1%D0%BD%D0%B8%D0%BC%D0%BE%D0%BA%2B%D1%8D%D0%BA%D1%80%D0%B0%D0%BD%D0%B0%2B%D0%BE%D1%82%2B2015-02-17%2B01%3A45%3A55.png>)

Отлично, мы нашли нужный пакет и убедились в том что это он, прочтя описание. Установим и поковыряемся в нем. Установка производится командой

```
sudo aptitude install libmagic-dev
```



(<http://4.bp.blogspot.com/-4t5CvUup41I/VOJ0MGNwivI/AAAAAAAAAmg/inpfvQRsPfc/s1600/%D0%A1%D0%BD%D0%B8%D0%BC%D0%BE%D0%BA%2B%D1%8D%D0%BA%D1%80%D0%B0%D0%BD%D0%B0%2B%D0%BE%D1%82%2B2015-02-17%2B01%3A50%3A15.png>)

Отлично, у нас появилась документация разработчика- посмотрим раздел 3 справочного руководства по библиотеке libmagic и мы найдем там описание нужной функции:



(<http://4.bp.blogspot.com/-nkWikwUTIF8/VOJ0r7edC0I/AAAAAAAAAmw/Y2ZsC1LJRSQ/s1600/%D0%A1%D0%BD%D0%B8%D0%BC%D0%BE%D0%BA%2B%D1%8D%D0%BA%D1%80%D0%B0%D0%BD%D0%B0%2B%D0%BE%D1%82%2B2015-02-17%2B01%3A52%3A24.png>)

На этом первая часть работы с инструментарием закончена, продолжение будет в следующей статье.

П.С. Данная статья (и последующие аналогичные) подготовлена на основе конспекта лекций (<https://dl.spbstu.ru/enrol/index.php?id=3925>) Кетова Дмитрия Владимировича

(<http://www.avalon.ru/HigherEducation/MasterNetworking//Teachers/About/?TeacherID=1>). От себя я добавил редактуру, несколько измененный стиль изложения, свои правки и комментарии.

Tagged DevOps (<https://kazarin.online/index.php/tag/devops/>), Go Deep (<https://kazarin.online/index.php/tag/go-deep/>), Linux (<https://kazarin.online/index.php/tag/linux/>), System (<https://kazarin.online/index.php/tag/system/>)

SSL И LET'S ENCRYPT
([HTTPS://KAZARIN.ONLINE/INDEX.PHP/2017/07/20/SSL-I-LETS-ENCRYPT/](https://kazarin.online/index.php/2017/07/20/SSL-I-LETS-ENCRYPT/))

LINUX. ГЛУБОКОЕ ПОНИМАНИЕ РАБОТЫ СИСТЕМЫ.
ЧАСТЬ 2 – ТРАССИРОВКА СТАНДАРТНЫХ И
СЕТЕВЫХ СЛУЖБ
([HTTPS://KAZARIN.ONLINE/INDEX.PHP/2017/07/21/LINUX-DEEP-IN-SYSTEM-WORK_PART-2/](https://kazarin.online/index.php/2017/07/21/LINUX-DEEP-IN-SYSTEM-WORK_PART-2/))


Поиск статей в блоге





Обо мне


DevOps Teamlead, SRE. Linux администратор, фанат поковырять что то


Свежие записи

 Заметки начинающего Тимлида:
Performance Review и с чем его едят
(https://kazarin.online/index.php/2021/08/07/junior_team_lead_performance_review_basic/)


 Как я баг через mongodb explain
нашел — или зачем надо быть немного
DBA
(https://kazarin.online/index.php/2021/04/07/stroy_about_explain/)


 20 лучших практик для Dockerfile
(<https://kazarin.online/index.php/2021/04/01/top-20-dockerfile-best-practices/>)


 10 Антипаттернов использования
Docker
(<https://kazarin.online/index.php/2021/03/17/docker-antipatterns/>)


 20 Признаков того, что в вашей
инфраструктуре что-то идет не так.
(https://kazarin.online/index.php/2020/05/17/20_infrastructure_antipatterns/)


Архивы

 Август 2021
(<https://kazarin.online/index.php/2021/08/>) (1)

 Апрель 2021
(<https://kazarin.online/index.php/2021/04/>) (2)

 Март 2021
(<https://kazarin.online/index.php/2021/03/>) (1)

 Май 2020
(<https://kazarin.online/index.php/2020/05/>) (2)

 Февраль 2020
(<https://kazarin.online/index.php/2020/02/>)

0/02/) (1)

📅 Октябрь 2019

(<https://kazarin.online/index.php/2019/10/>) (1)

📅 Сентябрь 2019

(<https://kazarin.online/index.php/2019/09/>) (1)

📅 Июнь 2019

(<https://kazarin.online/index.php/2019/06/>) (4)

📅 Февраль 2019

(<https://kazarin.online/index.php/2019/02/>) (3)

📅 Январь 2019

(<https://kazarin.online/index.php/2019/01/>) (1)

📅 Ноябрь 2018

(<https://kazarin.online/index.php/2018/11/>) (3)

📅 Апрель 2018

(<https://kazarin.online/index.php/2018/04/>) (2)

📅 Декабрь 2017

(<https://kazarin.online/index.php/2017/12/>) (1)

📅 Ноябрь 2017

(<https://kazarin.online/index.php/2017/11/>) (4)

📅 Октябрь 2017

(<https://kazarin.online/index.php/2017/10/>) (6)

📅 Сентябрь 2017

(<https://kazarin.online/index.php/2017/09/>) (1)

📅 Август 2017

(<https://kazarin.online/index.php/2017/08/>) (4)

📅 Июль 2017

(<https://kazarin.online/index.php/2017/07/>) (13)

Рубрики

1C

(<https://kazarin.online/index.php/category/1c/>)

Ansible

(<https://kazarin.online/index.php/category/it-blog/devops/ansible-devops/>)

Ansible

(<https://kazarin.online/index.php/category/it-blog/linux/ansible/>)

backup

(<https://kazarin.online/index.php/category/it-blog/backup/>)

Bash

(<https://kazarin.online/index.php/category/it-blog/linux/bash/>)

Books

(<https://kazarin.online/index.php/category/books/>)

databases

(<https://kazarin.online/index.php/category/it-blog/databases/>)

DevOps

(<https://kazarin.online/index.php/category/it-blog/devops/>)

Docker

(<https://kazarin.online/index.php/category/it-blog/docker/>)

Education

(<https://kazarin.online/index.php/category/it-blog/education/>)

Git

(<https://kazarin.online/index.php/category/it-blog/devops/git/>)

IT Blog

(<https://kazarin.online/index.php/category/it-blog/>)

KVM

(<https://kazarin.online/index.php/category/it-blog/kvm/>)

- 📁 lessons
(<https://kazarin.online/index.php/category/it-blog/lessons/>)
- 📁 Library
(<https://kazarin.online/index.php/category/library/>)
- 📁 Linux
(<https://kazarin.online/index.php/category/it-blog/linux/>)
- 📁 lvm
(<https://kazarin.online/index.php/category/it-blog/linux/lvm/>)
- 📁 mdadm
(<https://kazarin.online/index.php/category/it-blog/linux/mdadm/>)
- 📁 Private
(<https://kazarin.online/index.php/category/it-blog/private/>)
- 📁 Programming
(<https://kazarin.online/index.php/category/it-blog/programming/>)
- 📁 Security
(<https://kazarin.online/index.php/category/it-blog/security/>)
- 📁 system engineer
(<https://kazarin.online/index.php/category/it-blog/system-engineer/>)
- 📁 Team Lead
(<https://kazarin.online/index.php/category/team-lead/>)
- 📁 Terraform
(<https://kazarin.online/index.php/category/it-blog/devops/terraform/>)
- 📁 tools
(<https://kazarin.online/index.php/category/it-blog/linux/tools/>)
- 📁 Uncategorized
(<https://kazarin.online/index.php/category/uncategorized/>)
- 📁 Vagrant
(<https://kazarin.online/index.php/category/it-blog/vagrant/>)

Virtualization

(<https://kazarin.online/index.php/category/it-blog/virtualization/>)

Web

(<https://kazarin.online/index.php/category/it-blog/web/>)

Лайфхаки

(<https://kazarin.online/index.php/category/%d0%bb%d0%b0%d0%b9%d1%84%d1%85%d0%b0%d0%ba%d0%b8/>)

Новости

(<https://kazarin.online/index.php/category/%d0%bd%d0%be%d0%b2%d0%be%d1%81%d1%82%d0%b8/>)

Обзор

(<https://kazarin.online/index.php/category/it-blog/%d0%be%d0%b1%d0%b7%d0%be%d1%80/>)

Реклама

(<https://kazarin.online/index.php/category/it-blog/%d1%80%d0%b5%d0%ba%d0%bb%d0%b0%d0%bc%d0%b0/>)



Сайт работает на WordPress (<http://wordpress.org/>) | Тема: Amadeus
(<http://themeisle.com/themes/amadeus/>) от Themeisle.