

Книга Radare2

1-е издание

Содержание

Введение	9
История проекта	9
Пакет программ radare2	10
Программа radare2	10
Программа rabin2	10
Программа rasm2	10
Примеры	10
Программа rahash2	11
Примеры	11
Программа radiff2	11
Программа rafind2	11
Программа ragg2	11
Примеры	11
Программа rarun2	11
Пример скрипта rarun2	12
Подключение программы к сокету	12
Отладка программы, перенаправив stdio в другой терминал	12
Программа rax2	12
Примеры	12
Загрузка radare2	12
Сборка при помощи meson + ninja	13
Вспомогательные скрипты	13
Очистка директория сборки	13
Компиляция и переносимость	13
Статическая сборка	14
Сборка при помощи Meson	14
Установка в Docker	14
Деинсталляция старых установок Radare2	14
Windows	14
Необходимые ресурсы и инструменты	14
Последовательность шагов	14
Установка Visual Studio 2015 или более поздней версии	14
Установка Python 3 и Meson при помощи пакетного менеджера Conda	14
Создание среды (environment) Python для Radare2	14
Установка Meson	15
Установка Git for Windows	15
Загрузка исходного кода Radare2	15
Компилирование Radare2 Code	15
Заметки о параметрах сборки	15
Как убедится, что Radare2 запускается из любой папки	16
Примечания по настройке общесистемных (system-wide) переменных среды	16
Android	16
Необходимые инструменты	16
Последовательность шагов	16
Загрузка и распаковка Android NDK	16
Использование Make	16
Указание месторасположение NDK	16
Компилирование и создание архива tar.gz, загрузка в подключенное устройство android	16
Использование Meson	16
Создание кросс-файл для мезона	16
Компилирование при помощи meson + ninja	17
Копирование файлов на свое устройство Android, запуск приложения	17
Интерфейсы пользователя	17
Базовое использование Radare2	18
Аргументы командной строки	19
Частые варианты использования	20
Формат команды	20
Выражения	21
Основные команды отладчика	22

Улучшение книги Radare2	23
Книга Radare2	23
Конфигурация	24
Цветовая раскраска в интерфейсе	25
Темы	25
Переменные конфигурации	25
Переменная asm.arch	25
Переменная asm.bits	25
Переменная asm.syntax	25
Переменная asm.pseudo	27
Переменная asm.os	27
Переменная asm.flags	27
Переменная asm.lines.call	27
Переменная asm.lines.out	27
Переменная asm.linestyle	27
Переменная asm.offset	27
Переменная asm.trace	27
Переменная asm.bytes	28
Переменная asm.sub.reg	28
Переменная asm.sub.jmp	28
Переменная asm.sub.rel	28
Переменная asm.sub.section	28
Переменная asm.sub.varonly	28
Переменная cfg.bigendian	28
Переменная cfg.newtab	28
Переменная scr.color	28
Переменная scr.seek	29
Переменная scr.scrollbar	29
Переменная scr.utf8	29
Переменная cfg.fortunes	29
Переменная cfg.fortunes.type	29
Переменная stack.size	29
Файлы radare2	29
Файлы RC	29
Целевой файл	29
Основные команды	30
Установка базового смещения	30
Открыть файл	31
Перемещение смещения на любые адреса	32
Размер блока	32
Секции	33
Отображение файлов	33
Режимы отображения данных	34
Шестнадцатеричный вид	35
Шестнадцатеричный дамп слов (32 бита)	35
8-битовый шестнадцатеричный набор байтов	36
Шестнадцатеричный дамп из Quad-слов (64 бита)	36
Форматы даты/времени	36
Основные типы	37
Высокоуровневые представления в синтаксисе языков программирования	38
Строки	38
Печать содержимого памяти	39
Дизассемблирование	39
Выбор целевой архитектуры	40
Настройка дизассемблера	40
Синтаксис дизассемблера	40
Флаги	40
Локальные флаги	41
Зоны флагов	42
Сохранение данных	42

Сохранение поверх существующих данных	43
Масштабирование	43
Сохранение/Вставка	44
Сравнение байтов	45
База данных строк	46
Пример использования	46
Применения базы данных	47
Еще примеры	47
Dietline	48
Автодополнение	48
Режим Emacs (по умолчанию)	48
Перемещение	48
Удаление	48
Killing and Yanking	48
История	48
Режим Vi	49
Вход в командные режимы	49
Перемещение	49
Удаление и вставка	49
Визуальный режим	49
Навигация	49
режимы отображения (панели)	49
Получение инструкций	51
Визуальный дизассемблер	51
Навигация	51
Команда d - определить	52
Использование курсора для вставки/исправления.	52
Перекрасные ссылки	53
Отображение аргументов функции	53
Добавление комментария	54
Введите другие команды	54
Поиск	54
HUD-ы	54
UserFriendly HUD	54
HUD для флагов/комментариев/функций/	54
Настройка дизассемблирования	54
Визуальный редактор конфигурации	54
Примеры	54
asm.arch: Изменение архитектуры && asm.bits: Размер слова в битах на ассемблере	54
asm.pseudo: Включить псевдосинтаксис	58
asm.syntax: Выбор синтаксиса дизассемблирования (intel, att, masm...)	58
asm.describe: Показать описание оп-кода	58
Визуальный ассемблер	58
Визуальный редактор конфигурации	58
Визуальные панели	58
Концепция	58
Обзор	58
Команды	58
Базовые функции	65
Разделение экрана	65
Команды оконного режима	65
Изменение значений	65
Вкладки	65
Сохранение макетов	65

Поиск байтов	65
Обычный поиск	66
Настройка параметров поиска	67
Поиск сопоставлением с шаблоном	67
Автоматизация обработки найденных вхождений	68
Поиск в обратном направлении	68
Поиск в результатах ассемблирования	69
Поиск криптографических данных	69
Поиск ключей AES	69
Поиск закрытых ключей и сертификатов	69
Энтропийный анализ	69
Дизассемблирование	70
Добавление метаданных к тексту дизассемблера	71
ESIL	72
Использование ESIL	73
ESIL-команды	73
Набор инструкций ESIL	73
Флаги ESIL	79
Синтаксис и команды	79
Порядок аргументов для неассоциативных операций	80
Специальные указания	80
Экспресс-анализ	80
Флаги ЦП	80
Переменные	80
Битовые массивы	80
Арифметика	81
Битовая арифметика	81
Поддержка устройств с плавающей запятой	81
Обработка префикса x86 REP в ESIL	81
Пример использования:	81
Нереализованные/необработанные инструкции	81
Пример дизассемблирования ESIL:	81
Интроспекция	82
API HOOKS	82
Анализ данных и кода	82
Анализ кода	83
Анализ функций	84
Анализ функции вручную	85
Рекурсивный анализ	87
Конфигурация	89
Конфигурация потока управления	89
Управление ссылками	89
Диапазоны анализа	89
Таблицы переходов	89
Элементы управления, специфичные для платформы	90
Визуальный подход	90
Подсказки по анализу	90
Управление переменными	93
Вывод типа	94
Типы данных	95
Загрузка типов	96
Распечатка структур	96
Типы ссылок	97
Structure Immediates	97
Управление перечислениями	98
Внутреннее представление	98
Структуры	99
Юнионы	99

Прототипы функций	99
Соглашения вызова функции	100
Виртуальные таблицы	101
Системные вызовы	101
Эмуляция ESIL	102
Эмуляция в цикле анализа	103
Символы	104
Сигнатуры	105
Поиск лучших совпадений zb	107
Команды графа управления	108
Ascii Art ** (например, agf)	108
Интерактивный Ascii-art (например, agfv)	108
Крошечный Ascii-art (например, agft)	108
Graphviz dot (например, agfd)	109
JSON (например, agfj)	109
Язык моделирования графа (например, agfg)	109
Ключ-значение SDB (например, agfk)	109
Пользовательские команды R2 для графа (например, agf*)	109
Веб / изображение (например, agfw)	109
Скрипты	109
Итераторы	111
Макросы	113
Псевдонимы	113
Программа R2pipe	114
Примеры	114
Python	114
NodeJS	114
Go	115
Rust	115
Ruby	115
Perl	116
Erlang	116
Haskell	116
Dotnet	116
Java	117
Swift	117
Vaal	118
NewLisp	118
Dlang	118
Отладчик	118
Начало работы	119
Небольшая сессия в отладчике Radeone2	119
Переход с IDA, GDB и WinDBG	119
Как запустить программу с помощью отладчика	119
.	119
Как присоединиться/отсоединиться от исполняющегося процесса? (gdb -p)	120
.	120
Как установить аргументы/переменные среды/загрузить динамическую библиотеку в сеансе отладки radare	120
Как запускать скрипты в radare2 ?	120
Как показать исходный код подобно gdb?	120

сочетания клавиш	120
Эквивалент команды <code>gdb ``set-follow-fork-mode``</code>	123
Общие функции	123
Регистры	123
Карты памяти	125
Куча	126
Файлы	127
Отладка в обратном направлении	127
Сообщения Windows	128
Возможности удаленного доступа	129
Отладка при помощи <code>gdbserver</code>-а	130
Отладка в режиме ядра с WinDBG KD	131
Настройка KD в Windows	132
Последовательный порт	132
Сеть	132
Подключение к интерфейсу KD на r2	132
Последовательный порт	132
Сеть	132
Использование KD	133
WinDBG бэкенд для Windows (DbgEng)	133
Использование плагина	133
Инструменты	133
Программа Rax2	134
Программа rafind2	135
Программа Rarun2	136
Rabin2 --- свойства двоичного файла	137
Идентификация свойств файла	138
Точки ввода в код	138
Импорты	138
Экспорты	139
Символы (Экспорты)	139
Список библиотек	140
Строки	140
Секции программ	141
Программа Radiff2	142
Бинарное сравнение	142
Утилита Rasm2	143
Ассемблирование	146
Визуальный режим	147
Дизассемблер	147
pd N	147
pD N	147
pda	147
pi, pI	147
Конфигурирование дизассемблера	147
Программа ragg2	148
Пример компиляции ragg2	148

Маленькие двоичные файлы	149
Синтаксис языка	149
Пропроцессор	149
Псевдонимы	149
Включения в текст	149
Хашбанг	149
Main	149
Определение функции	149
Сигнатуры функций	150
Типы функций	150
Системные вызовы	150
Библиотеки	150
Основная библиотека	150
Переменные	150
Массивы	151
Отслеживание	151
Указатели	151
Виртуальные регистры	151
Математические операции	151
Возвращаемые значения	151
Ловушки (traps)	151
Встроенный ассемблер	151
Метки	152
Управление исполнением	152
Комментарии	152
Программа rahash2	152
Блочное хеширование	152
Хеширование при помощи rabin2	152
Получение хэшей в сеансе radare2	152
Примеры	153
Плагины	154
Типы плагинов	154
Получение списка плагинов	154
Примечания	155
Плагины ввода-вывода	155
Реализация плагина дизассемблирования	156
Перемещение плагина в дерево исходников radare2	157
Реализация нового плагина анализа	157
Реализация нового формата двоичного файла	159
Включение виртуальной адресации	159
Оформление папки с именем формата файла в libr/bin/format	159
Примеры	160
Реализация плагина отладчика	161
То ли еще будет!	161
Реализация новой псевдоархитектуры	161
Плагины Python	161
Реализация плагина формата в Python	163
Отладка	164
Тестирование плагина	165
Реализация пакета r2pm для плагина	165
Задачи Crackmes	165
Задачки IOLI CrackMes	165
IOLI 0x00	166

IOLI 0x01	166
IOLI 0x02	168
IOLI 0x03	171
IOLI 0x04	174
0x04	174
IOLI 0x05	175
IOLI 0x06	178
IOLI 0x07	181
IOLI 0x08	184
IOLI 0x09	184
Avatao R3v3rs3 4	184
.radare2	185
.first_steps	186
.main	187
.vmloop	195
instr_A	204
instr_S	209
instr_I	211
instr_D	211
instr_P	211
instr_X	212
instr_J	212
.bytecode	214
.outro	214
Справочная карта Radare2	217
Руководство по выживанию	217
Флаги	217
Пространство флагов	217
Информация	217
Печать строк	218
Визуальный режим	218
Поиск	219
Сохранения (сломано)	219
Использование переменных в выражениях	219
Авторы и активные участники	220
Книга radare2	220

Введение

Книга представляет собой обновленную версию оригинальной книги о radare2, написанной пользователем pancake. Обновление начато пользователем `maijin`. Текст книги активно поддерживается и обновляется многими участниками через Интернет.

На сервере Github опубликованы ее исходники, добавляйте новое содержимое или исправляйте опечатки:

- Github: <https://github.com/radareorg/radare2-book>
- PDF: GHA Artifacts
- EPUB: GHA Artifacts
- Английская версия Online: <https://book.rada.re/>

История проекта

В 2006 году Серхи Альварес (Sergi Àlvarez, он же pancake) работал судебным аналитиком. Поскольку не разрешалось использовать программное обеспечение компании для своих личных нужд, он решил написать небольшой инструмент - шестнадцатеричный редактор, реализующий следующие требования:

- быть чрезвычайно переносимым (unix friendly, command line, c, small),
- открывать дисковые устройства, использование 64-битных смещений,
- поиск строк и шестнадцатеричных кодов,
- просмотр и сброс результатов на диск.

Редактор изначально разрабатывался для восстановления удаленных файлов из раздела HFS+.

После этого pancake решил усовершенствовать инструмент: реализовать подключаемый io, возможность подключаться к процессам. Для этого он добавил возможности отладчика, поддержку нескольких архитектур и анализ кода.

С тех пор проект эволюционировал. Теперь он оснащает полную инфраструктуру для анализа двоичных файлов, используя основные концепции UNIX. Концепции включают в себя знаменитые парадигмы «все является файлом», «взаимодействие программ через объединение stdin/stdout» и «не усложняйте без необходимости».

Потребность в запуске сценариев выявила проблемы в первоначальном дизайне: монолитный инструмент усложнил использование API, потребовался глубокий рефакторинг. В 2009 году создан radare2 (r2) как ответвление radare1. Рефакторинг добавил желанной гибкости и динамические функции. Это позволило значительно улучшить интеграцию, проложив путь к использованию r2 из разных языков программирования. Позже API r2pipe обеспечил доступ к radare2 при помощи каналов из любого языка.

То, что начиналось как проект одного человека с небольшими эпизодическими дополнениями, постепенно превратилось в большой открытый проект в 2014 году. Число пользователей быстро росло, и автору и главному разработчику приходилось метаться между кодированием и менеджментом проекта, интегрируя результаты различных разработчиков, присоединившихся к проекту.

Инструктирование пользователей о необходимости сообщать о своих проблемах позволило проекту определять новые направления развития. Исходный код находится в GitHub radare2, а обсуждение - в Telegram-канале.

Проект остается активным и сейчас, в момент написания этой книги. Есть и несколько побочных проектов, предоставляющих, среди прочего, графический пользовательский интерфейс (Cutter), декомпилятор (r2dec, radeco), интеграцию с Frida (r2frida), Yara, Unicorn, Keystone и многое другое. Проекты доступны в r2pm (менеджер пакетов radare2).

С 2016 года разработчики собирается раз в год на r2con, конгресс по проблемам radare2, проводимый в Барселоне.

Пакет программ radare2

Программная система Radare2 представляет собой набор небольших утилит командной строки, используемых вместе или по отдельности. Глава дает общее представление о том, как они функционируют, в специальных разделах в конце книги есть информация по каждому инструменту.

Программа radare2

Основной инструмент программной системы. Он основан на ядре, включающем шестнадцатеричный редактор и отладчик. Программа Radare2 позволяет загружать и отображать данные, полученные из различных источников данных (ввода/вывода), включая диски, сетевые подключения, драйверы ядра, отлаживаемые процессы и т.д., как будто это обычные файлы.

Возможности реализованного в ней интерфейса командной строки позволяют перемещаться по файлу, проводить анализ данных, дизассемблировать, вносить исправления в двоичные файлы, сравнивать данные, осуществлять поиск и замену, а также визуализацию. Также можно управлять этим интерфейсом из различных языков программирования, включая Python, Ruby, JavaScript, Lua и Perl.

Программа rabin2

Программа для извлечения информации из исполняемых двоичных файлов, таких как ELF, PE, Java CLASS, Mach-O, а также любого другого формата, поддерживаемого плагинами r2. Программа rabin2 используется ядром для получения данных о формате файла, например, перечень экспортированных символов, импортов, информации о формате файлов, перекрестных ссылках (xrefs), зависимостей библиотек и разделов (секций загрузчика).

Программа rasm2

Ассемблер и дизассемблер, управляемый из командной строки операционной системы, поддерживающий огромное множество архитектур, включая Intel x86 и x86-64, MIPS, ARM, PowerPC, Java.

Примеры

```
$ rasm2 -a java 'nop'
00
```

```
$ rasm2 -a x86 -d '90'
nop
$ rasm2 -a x86 -b 32 'mov eax, 33'
b821000000
$ echo 'push eax;nop;nop' | rasm2 -f -
509090
```

Программа rahash2

Реализация блочного хеширования. Она позволяет хэшировать текстовые строки и большие пространства на диске, rahash2 поддерживает много различных алгоритмов, включая MD4, MD5, CRC16, CRC32, SHA1, SHA256. Программа rahash2 используется для проверки целостности или отслеживания изменений больших файлов, дампов памяти или дисков.

Примеры

```
$ rahash2 file
file: 0x00000000-0x00000007 sha256: 887cfbd0d44aaff69f7bdbedebd282ec96191cce9d7fa7336298a18efc3c7a5a
$ rahash2 -a md5 file
file: 0x00000000-0x00000007 md5: d1833805515fc34b46c2b9de553f599d
```

Программа radiff2

Утилита для сравнения двоичных файлов, реализующая множество алгоритмов. Она поддерживает побайтовое сравнение или представление выявленных различий двоичных файлов, а также сравнение результатов анализа кода, полученных из радара, для поиска изменений в блоках кода.

Программа rafind2

Программа для поиска байтов в файлах по заданным шаблонам.

Программа ragg2

Представляет функции r_egg в виде утилиты командной строки. Программа ragg2 компилирует программы, представленные на простом языке высокого уровня, в крошечные двоичные файлы для архитектур x86, x86-64 и ARM.

Примеры

```
$ cat hi.r
/* hello world in r_egg */
write@syscall(4); //x64 write@syscall(1);
exit@syscall(1); //x64 exit@syscall(60);

main@global(128) {
    .var0 = "hi!\n";
    write(1,.var0, 4);
    exit(0);
}
$ ragg2 -O -F hi.r
$ ./hi
hi!

$ cat hi.c
main@global(0,6) {
    write(1, "Hello0", 6);
    exit(0);
}
$ ragg2 hi.c
$ ./hi.c.bin
Hello
```

Программа ragen2

Предназначена для запуска программ в разных средах, с другими аргументами, разрешениями, в других каталогах и переопределеными файловыми дескрипторами по умолчанию. Программа ragen2 используется для

- Решения задачек crackme
- Fuzzing
- Организации различного рода тестов

Пример скрипта rarun2

```
$ cat foo.rr2
#!/usr/bin/rarun2
program=./pp400
arg0=10
stdin=foo.txt
chdir=/tmp
#chroot=.
./foo.rr2
```

Подключение программы к сокету

```
$ nc -l 9999
$ rarun2 program=/bin/ls connect=localhost:9999
```

Отладка программы, перенаправив stdio в другой терминал 1 - Откройте новый терминал и введите `tty', чтобы получить название терминала:

```
$ tty ; clear ; sleep 999999
/dev/ttys010
```

2 - Создайте новый файл, содержащий следующий профиль rarun2 с именем foo.rr2:

```
#!/usr/bin/rarun2
program=/bin/ls
stdio=/dev/ttys010
```

3 - Теперь запустите команду radare2:

```
r2 -r foo.rr2 -d /bin/ls
```

Программа rax2

Минималистичный вычислитель математических выражений, используемый в командной строке операционной системы. Он полезен для выполнения базовых преобразований значений чисел с плавающей запятой в их шестнадцатеричное представление, шестнадцатеричных строк в ASCII, восьмеричных чисел в десятичные целые и многое другое. Программа также поддерживает установку порядка следования байтов в представлении, используется также в качестве интерактивной оболочки если аргументы не были в командной строке заданы.

Примеры

```
$ rax2 1337
0x539

$ rax2 0x400000
4194304

$ rax2 -b 01111001
y

$ rax2 -S radare2
72616461726532

$ rax2 -s 617765736f6d65
awesome
```

Загрузка radare2

Исходный код программы radare2 находится в GitHub-репозитории: <https://github.com/radareorg/radare2>

Бинарные пакеты доступны для операционных систем Ubuntu, Maemo, Gentoo, Windows, iPhone и др. Настоятельно рекомендуется именно загружать исходный код и самостоятельно компилировать его. Это дает лучшее понимание зависимостей, делает примеры более доступными и, конечно же, позволяет вам иметь у себя самую последнюю версию программы.

Новый стабильный релиз обычно публикуется в Интернет каждый месяц. Основная ветка разработки часто более стабильна, чем «стабильные» релизы. Загрузка последней версии:

```
$ git clone https://github.com/radareorg/radare2.git
```

Загрузка, вероятно, займет некоторое время, поэтому сделайте перерыв на кофе и продолжайте читать эту книгу.

Можно обновить локальную копию репозитория, используя `git pull` в любой папке дерева кода radare2:

```
$ git pull
```

Если код вами был локально модифицирован, то эти правки можно отменить (потерять их!):

```
$ git reset --hard HEAD
```

Или пришлите нам патч ваших исправлений:

```
$ git diff > radare-foo.patch
```

Наиболее удобным способом обновления и установки r2 в масштабах всей операционной системы является использование:

```
$ sys/install.sh
```

Сборка при помощи meson + ninja

Существует также недоделанная настройка для Meson.

Использование clang и ld.gold ускоряет сборку:

```
CC=clang LDFLAGS=-fuse-ld=gold meson . release --buildtype=release --prefix ~/.local/stow/radare2
ninja -C release
# ninja -C release install
```

Вспомогательные скрипты

Ознакомьтесь со скриптами в `sys/`, они используются для автоматизации этапов сборки, связанных с синхронизацией, построением и установкой r2 и его привязок к различным системам программирования. Наиболее важным из них является `sys/install.sh`. Он обновит, очистит, соберет и установит (symstall) r2 в операционную систему.

Symstalling - это процесс установки всех программ, библиотек, документации и файлов данных с использованием символьных ссылок, вместо копирования файлов. По умолчанию он будет установлен в `/usr/local`, но можно указать другой префикс, используя аргумент `--prefix`. Разработчикам так бывает очень удобно поскольку позволяет просто запускать 'make' и тестировать результаты изменения без необходимости запускать 'make install'.

Очистка директория сборки

Очистка дерева исходного кода позволяет избегать проблем связывания со старыми файлами объектов, необновления объектов после изменения ABI.

Следующие команды обновляют локальный клон git:

```
$ git clean -xdf
$ git reset --hard origin/master
$ git pull
```

Удаление предыдущих установок из системы выполняется следующими командами:

```
$ ./configure --prefix=/usr/local
$ make purge
```

Компиляция и переносимость

В настоящее время ядро radare2 можно скомпилировать для многих операционных систем и архитектур, но основная разработка ведется на GNU/Linux с помощью GCC, на MacOS X с помощью clang. Radare также компилируется на других системах и архитектурах, включая TCC и SunStudio.

Разработчики часто используют radare в качестве отладчика при обратном проектировании (реверс-инжениринге). Сейчас подсистему отладчика можно использовать в Windows, GNU/Linux (Intel x86, x86_64, MIPS и ARM), OS X, FreeBSD, NetBSD и OpenBSD (Intel x86 и x86_64).

По сравнению с ядром функция отладчика является ограниченно-переносимой. Если отладчик еще не портирован на вашу любимую платформу, можно отключить модуль отладки при помощи параметра `--without-debugger` скрипта `configure` при компиляции Radare2.

Напомним, что в программной системе есть плагины ввода-вывода, которые используют функции GDB, WinDbg или Wine, они полагаются на наличие соответствующих инструментов (в случае удаленной отладки --- только на целевой машине).

Чтобы собрать программную систему, используя acr и GNU Make, например, в системах *BSD:

```
$ ./configure --prefix=/usr
$ gmake
$ sudo gmake install
```

Существует и простой скрипт, собирающий проект автоматически:

```
$ sys/install.sh
```

Статическая сборка

Можно собрать radare2 статически вместе со всеми инструментами с помощью команды:

```
$ sys/static.sh
```

Сборка при помощи Meson

Для сборки можно использовать meson + ninja:

```
$ sys/meson.py --prefix=/usr --shared --install
```

Для локальной сборки:

```
$ sys/meson.py --prefix=/home/$USER/r2meson --local --shared --install
```

Установка в Docker

В репозитории исходного кода radare2 есть Dockerfile, используемый с системой Docker.

Этот dockerfile также используется в дистрибутиве Remnux из SANS, а также на сайте hub.docker.com, registryhub.

Деинсталляция старых установок Radare2

```
./configure --prefix=/old/r2/prefix/installation  
make purge
```

Windows

Компилирование r2 в Windows требует использования системы сборки Meson. Вообще существует возможность сборки r2 в Windows с использованием cygwin, mingw или wsl, используя в качестве системы сборки aot/make, однако этот подход не является рекомендуемым/официальным/поддерживаемым и может привести к неожиданным результатам.

Готовые двоичные сборки загружаются со страницы релизов, кроме того, можно использовать артефакты GITHUB CI, создаваемые на каждый коммит как для 32-разрядной, так и 64-разрядной версий Windows.

- <https://github.com/radareorg/radare2/releases>

Необходимые ресурсы и инструменты

- Требуется 3 ГБ свободного места на диске,
- Visual Studio 2019 (or higher),
- Python 3,
- Meson,
- Ninja,
- Git.

Последовательность шагов

Установка Visual Studio 2015 или более поздней версии Visual Studio должна включать компилятор Visual C++, поддерживающий библиотеки C++, и соответствующий пакет Windows SDK для целевой версии платформы.

- Убедитесь, что задан Programming Languages > Visual C++

Если у вас не установлена Visual Studio, то грузите какую-нибудь версию community-edition, эти версии бесплатны и отлично работают.

- Загрузка Visual Studio 2019

Установка Python 3 и Meson при помощи пакетного менеджера Conda Conda - это, вероятно, лучший дистрибутив Python для Windows. Пропустите следующие шаги, если у вас уже установлен Python

- <https://docs.conda.io/en/latest/miniconda.html>
- <https://repo.anaconda.com/archive/>

Создание среды (environment) Python для Radare2 Следующие действия создают и активируют среду Conda с именем r2. Все дальнейшие инструкции приводятся в предположении, что имя среды r2, при желании имя можно изменить.

1. Пуск > Anaconda Prompt
2. conda create -n r2 python=3
3. activate r2

Всякий раз, когда вы входите в среду, открыв Anaconda Prompt, надо запускать `activate r2`. И наоборот, запуск `deactivate` отключает от среды.

Установка Meson Убедитесь, что установлен Meson версии 0.48 или выше (`meson -v`)

```
pip install meson
```

Установка Git for Windows Разработка Radare2 ведется в системе управления версиями Git, исходный код опубликован на GitHub.

Установка Git для Windows требует выполнения следующих действий.

Загрузка Windows-версии Git

- <https://git-scm.com/download/win>

Во время установки проверьте следующие параметры.

- Использовать шрифт TrueType во всех окнах консоли,
- Использовать Git из командной строки Windows,
- Использовать библиотеку Windows Secure Channel, вместо OpenSSL,
- Установить флаг Windows-style, коммит `- Unix-style` для кодирования концов строк(`core.autocrlf=true`),
- Использовать окно консоли Windows по умолчанию, вместо Mintty,
- Убедитесь, что `git --version` срабатывает после инсталляции.

Загрузка исходного кода Radare2 Следующие действия приводят к клонированию git-репозитория Radare2.

```
git clone https://github.com/radareorg/radare2
```

Компилирование Radare2 Code Следующие действия - компиляция кода Radare2.

- **Visual Studio 2017:**

Замечание 1: Замените `Community` на `Professional` или `Enterprise` в следующем примере командной строки в зависимости от используемой вами версии.

Замечание 2: Замените `vcvars32.bat` на `vcvars64.bat` в этой строке для сборки 64-битовой версии.

```
"%ProgramFiles(x86)%\Microsoft Visual Studio\2017\Community\VC\Auxiliary\Build\vcvars32.bat
```

4. Создание системы сборки с помощью Meson:

В командной строке Meson можно задавать аргументы для настройки типа сборки. Однако, `r2` должен собраться без дополнительных флагов `meson-a`. При использовании Visual Studio. Замечание: Замените `Debug` на `Release` в следующей строке в зависимости от намерения собрать версию для отладки или релиза.

```
meson b --buildtype debug --backend vs2019 --prefix %cd%\dest  
msbuild build\radare2.sln /p:Configuration=Debug /m
```

При использовании Ninja интерфейс Visual Studio не требуется, достаточно установить инструменты компилятора msvc:

```
meson b  
ninja -C b
```

Наконец, надо запустить следующую строку для установки `r2` в заданный каталог (абсолютный префикс):

```
meson install -C build --no-rebuild
```

Заметки о параметрах сборки Параметр `/m[ахсруискоунт]` создает один рабочий процесс MSBuild для каждого процессорного ядра компьютера. Указание числового значения (например, `/m:2`) ограничивает количество рабочих процессов. (Не следует путать с параметром компилятора Visual C++ `/MP`.)

Иногда при попытке 32-разрядной установки можно получить сообщение такого вида: `error MSB4126: The specified solution configuration "Debug|x86" is invalid.` Проблема решается добавлением следующего аргумента: `/p:Platform=Win32`.

Проверить версию собранного Radare2: `dest\bin\radare2.exe -v`

Как убедится, что Radare2 запускается из любой папки В системах UNIX r2 --- это просто символьная ссылка на исполняемый файл radare2. В Windows если вы хотите, чтобы у вас был именно r2, то надо просто скопировать в него radare2.exe: copy radare2.exe r2.exe. Также надо добавить каталог с этими файлами в переменную PATH в настройках Проводника Windows.

Откройте консоль cmd.exe и наберите r2 -v. Если не получите ошибку, то весь процесс сборки и установки прошел успешно.

Примечания по настройке общесистемных (system-wide) переменных среды

1. В проводнике windows перейдите в папку Radare2, т.е., в которую только что мы устанавливали пакет.
2. Из этой папки перейдите в dest > bin и оставьте окно открытым.
3. Перейдите в раздел ``Свойства системы'' (System Properties): в строке поиска Windows введите sysdm.cpl.
4. Перейдите в Дополнительно > Переменные среды (Advanced > Environment Variables).
5. Щелкните на переменной PATH, затем нажмите кнопку edit. Если PATH присутствует в настройках и для пользователя, и среди перечня системных переменных, то лучше править пользовательскую.
6. Проверьте наличие в переменной PATH пути к папке в оставленном ранее окне. Если это не так, добавьте эту папку и нажмите OK.
7. Выйдите из сеанса Windows.
8. Откройте командную строку Windows: введите cmd в строке поиска. Убедитесь, что текущая папка не папка Radare2.
9. Проверьте работоспособность Radare2 в командной строке: radare2 -v.

Android

Программа Radare2 также может быть кросс-компилирована для других архитектур и операционных систем, например в Android.

Необходимые инструменты

- Python 3,
- Meson,
- Ninja,
- Git,
- Android NDK.

Последовательность шагов

Загрузка и распаковка Android NDK Загрузите Android NDK с официального сайта и распакуйте его где-нибудь у вас в системе, например в директории /tmp/android-ndk.

Использование Make

Указание месторасположение NDK

```
$ echo NDK=/tmp/android-ndk > ~/.r2androidrc
```

Компилирование и создание архива tar.gz, загрузка в подключенное устройство android

```
./sys/android-build.sh arm64-static
```

Можно собрать пакет для какой-либо другой архитектуры, изменив аргумент на ./sys/android-build.sh. Запустить скрипт без аргументов, он покажет использованные значения.

Использование Meson

Создание кросс-файл для мезона Meson-у нужен конфигурационный файл, описывающий среду кросс-компиляции, например, meson-android.ini. Его можно донастроить при необходимости. Следующий пример - хорошая отправная точка:

```
[binaries]  
c      = '/tmp/android-ndk/toolchains/llvm/prebuilt/linux-x86_64/bin/aarch64-linux-android28-clang'  
cpp    = '/tmp/android-ndk/toolchains/llvm/prebuilt/linux-x86_64/bin/aarch64-linux-android28-clang++'  
ar     = '/tmp/android-ndk/toolchains/llvm/prebuilt/linux-x86_64/bin/aarch64-linux-android-ar'  
as     = '/tmp/android-ndk/toolchains/llvm/prebuilt/linux-x86_64/bin/aarch64-linux-android-as'  
ranlib = '/tmp/android-ndk/toolchains/llvm/prebuilt/linux-x86_64/bin/aarch64-linux-android-ranlib'  
ld     = '/tmp/android-ndk/toolchains/llvm/prebuilt/linux-x86_64/bin/aarch64-linux-android-ld'  
strip  = '/tmp/android-ndk/toolchains/llvm/prebuilt/linux-x86_64/bin/aarch64-linux-android-strip'  
pkgconfig = 'false'
```

```
[properties]
sys_root = '/tmp/android-ndk/sysroot'

[host_machine]
system = 'android'
cpu_family = 'arm'
cpu = 'aarch64'
endian = 'little'
```

Компилирование при помощи meson + ninja Настроить каталог сборки с meson-ом, как обычно:

```
$ CFLAGS="-static" LDFLAGS="-static" meson --default-library static --prefix=/tmp/android-dir -Dblob=true build --cross-file ./meson-android.ini
```

Поясним значения всех перечисленных выше переменных: * `CFLAGS=static`, `LDFLAGS=static`, `--default-library static` обеспечивает компилирование библиотек и запускаемых файлов в статическом режиме, теперь нет необходимости указывать значения переменных `LD_*` в вашей среде Android-а. Переменные указывают, где находятся подходящие библиотеки. В статическом режиме они не нужны. У двоичных (запускаемых) файлов есть все, что им нужно внутри. * `-Dblob=true` указывает meson-у компилировать только один единый запускаемый файл со всем требуемым кодом для всех программ `radare2`, `rabin2`, `rasm2`, и т.д. В результате для каждой программы будет создана символьная ссылка на этот единый файл. Это позволяет избежать создания множества больших статически скомпилированных двоичных файлов, а просто создается один единый, включающий все функции. Мы получим наши `rabin2`, `rasm2`, `rax2`, и др., но они будут только ссылками на `radare2`. * `--cross-file ./meson-android.ini` описывает процесс компилирования `radare2` для ОС Android

Затем, надо скомпилировать проект, и установить его на устройство:

```
$ ninja -C build
$ ninja -C build install
```

Копирование файлов на свое устройство Android, запуск приложения Последний этап - копирование сгенерированных файлов в `/tmp/android-dir` на ваше Android-устройство, запуск на нем `radare2`:

```
$ cd /tmp && tar -cvf radare2-android.tar.gz android-dir
$ adb push radare2-android.tar.gz /data/local/tmp
$ adb shell
DEVICE:/ $ cd /data/local/tmp
DEVICE:/data/local/tmp $ tar xvf radare2-android.tar.gz
DEVICE:/data/local/tmp $ ./android-dir/bin/radare2
Usage: r2 [-ACdfLMnNgStuvwxyzX] [-P patch] [-p prj] [-a arch] [-b bits] [-i file]
        [-s addr] [-B baddr] [-m maddr] [-c cmd] [-e k=v] file|pid|-|--|=
```

Интерфейсы пользователя

Для Radare2 на протяжении многих лет разработано много различных пользовательских интерфейсов.

Идея графического интерфейса далеко от использованной идеологии основного механизма инструментария обратного проектирования: предпочтительно иметь отдельный проект и сообщество, позволяющее обоим проектам сотрудничать и совершенствоваться вместе, вместо того, чтобы заставлять разработчиков интерфейса командной строки думать о проблемах графического интерфейса и прыгать взад-вперед между графическим аспектом и низкоуровневой логикой реализаций.

В прошлом было, по крайней мере, пять различных пользовательских интерфейсов (`ragui`, `r2gui`, `gradare`, `r2net`, `bokken`), но ни один из них не получил достаточной поддержки, чтобы набрать достаточную популярность, в результате и все они ``сложили ласты''.

Программа `r2` снабжена встроенным веб-сервером и предоставляет несколько простых пользовательских интерфейсов, реализованных при помощи HTML/JS. Данный режим запускается следующим образом:

```
$ r2 -c=H /bin/ls
```

После трех лет самостоятельной разработки Уго Тесо, автор проекта Bokken (python-gtk-интерфейс для `r2`), выпустил для общественности еще один интерфейс для `r2`, но на этот раз написанный на `c++` и `qt`, что очень понравилось сообществу.

Разработанный графический интерфейс назван `Iaito`, однако Уго несколько лениво поддерживал проект, что привело и созданию на основе `Iaito` нового проекта под именем `Cutter` (так проголосовало сообщество), возглавил проект пользователь Xarkes. Вот как он выглядит:

- <https://github.com/radareorg/cutter>.

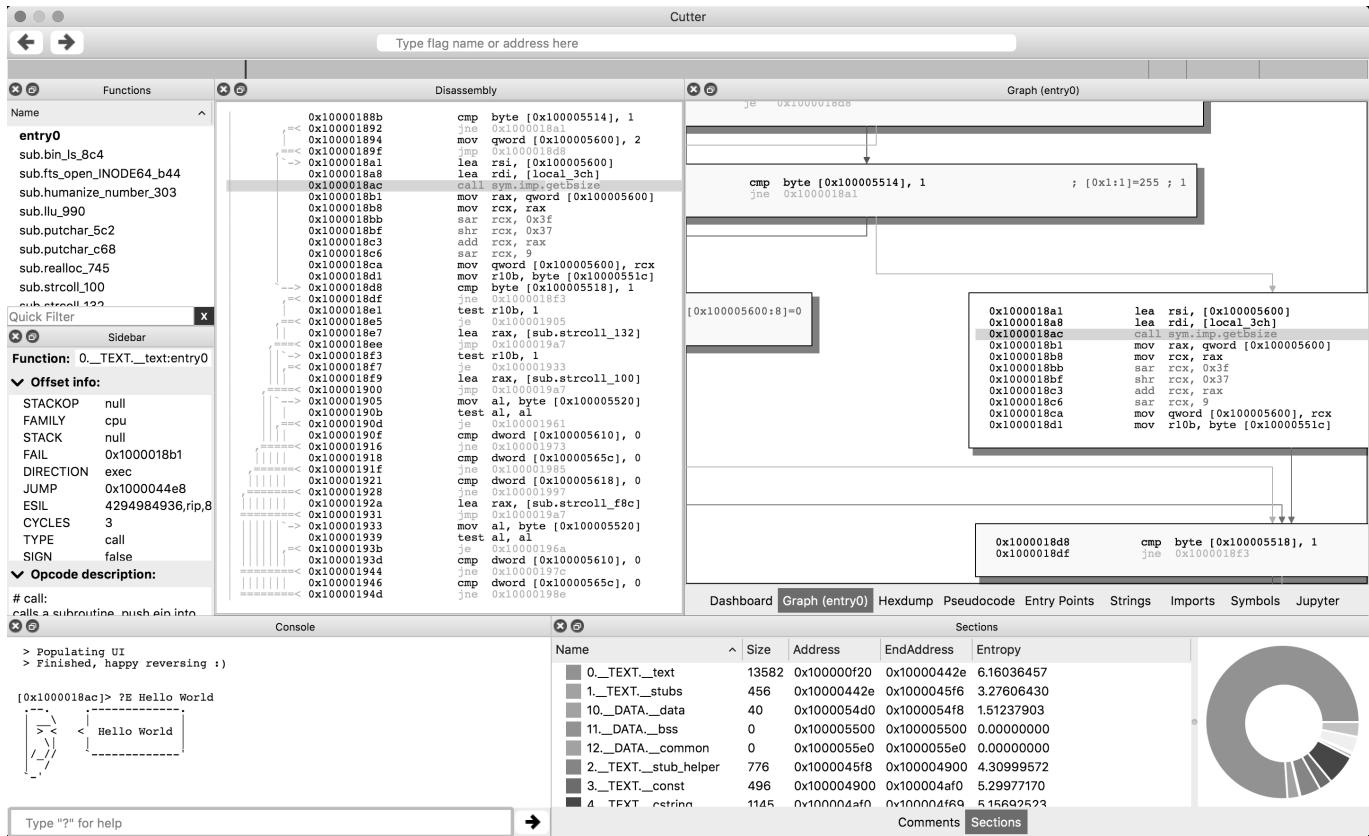


Рис. 1: Cutter

Базовое использование Radare2

Кривая обучения обычно пологая только в самом начале. После часа использования уже можно легко понять, как работает большинство функций, и как можно комбинировать различные инструменты радара. Вам предлагается прочитать оставшуюся часть этой книги, чтобы понять, как работают некоторые нетривиальные вещи, и в конечном итоге улучшить свои навыки.

Перемещение по адресам, исследование и правка загруженного двоичного файла выполняются при помощи трех простых действий: перемещение по адресному пространству (в некоторый адрес), отображение (буфера) и внесение изменений (запись, добавление).

Команда «seek» обозначается аббревиатурой **S**, она принимает выражение в качестве аргумента. Выражение может быть следующего вида -10, +0x25, или даже [0x100+ptr_table]. Если вы хотите работать с блочными файлами, вы можете зафиксировать размер блока с помощью команды **b** и осуществлять перемещение по памяти вперед или назад по адресам байтов. Использование **S++** и **S--** аналогично позволяет перемещаться по памяти.

Если в radare2 открыть исполняемый файл, то по умолчанию он откроется в режиме виртуальной адресации (VA), и секции файла будут отображены в их виртуальные адреса. В режиме VA основа перемещения по адресам - этот виртуальный адрес, а начальная позиция устанавливается в точку входа исполняемого файла. Использование ключа -P позволяет отменить загрузку в данном режиме, при этом radare2 откроет файл в не-VA режиме. В режиме не-VA перемещение по адресам осуществляется относительно начала файла.

Команда 'print' также имеет сокращенный вариант **p** и включает несколько вариантов функционирования, вторая буква указывает как раз требуемый режим отображения. Наиболее часто встречаются **px**, отображающий данные в виде шестнадцатеричного дампа, **pd** - дизассемблирование кода.

Чтобы radare2 мог сохранять изменения в исследуемых файлах, надо указать флаг **-w** в командной строке при открытии файла. Команда **w** используется для записи строк в исходный файл, шестнадцатеричных кодов (режим **X**) или даже ассемблерные коды операций (режим **a**). Примеры:

```
> w hello world      ; строка
> wx 90 90 90 90    ; шестнадцатеричные данные
> wa jmp 0x8048140   ; ассемблирование
> wf inline.bin      ; запись содержимого из файла
```

Добавление символа **?** к любой команде показывает подсказку, например, **p??.** Если написать **?***, то получим список команд, начинающихся с заданной строки, например, **p?*.**

R2 LEARNING CURVE

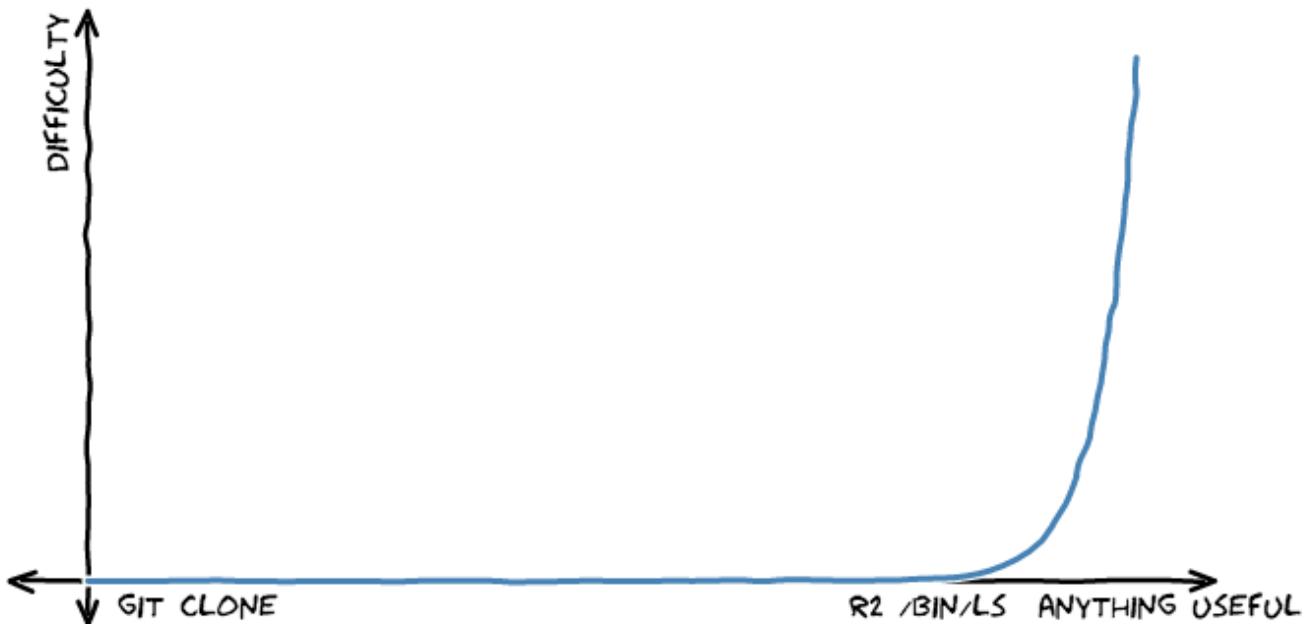


Рис. 2: learning_curve

Визуальный режим запускается вводом `V<enter>`. По клавише `q` можно выйти из этого режима и вернуться в привычный режим командной строки.

В визуальном режиме для перемещений используются клавиши `HJKL` (влево, вниз, вверх и вправо, соответственно). Также можно использовать эти клавиши в режиме курсора, режим переключается клавишей `C`. Чтобы выбрать диапазон байтов в режиме курсора, одновременно нажмите `Shift` и клавиши навигации `HJKL`, в результате radare2 выделит область.

В визуальном режиме также можно переписывать байты, нажав `i`. Нажатием клавиши `TAB` осуществляется перемещение между панелями шестнадцатеричных кодов (посередине) и строчным представлением (справа). Нажатие `q` в панели шестнадцатеричного представления приводит к возвращению в визуальный режим. Нажатием `r` или `P` приводит к циклическому переключению различных режимов визуального представления. Существует второй по важности визуальный режим --- интерфейс панелей, похожий на `curses`, доступный при помощи комбинации `V!`.

Аргументы командной строки

Ядро Radare управляется множеством флагов, аргументов командной строки.

Вот кусочек сообщения справки по использованию флагов:

```
$ radare2 -h
Usage: r2 [-ACdfLMnNqStuvwzX] [-P patch] [-p proj] [-a arch] [-b bits] [-i file]
          [-s addr] [-B baddr] [-m maddr] [-c cmd] [-e k=v] file|pid|-|-|=

--          запустить radare2 без открытия файла
-          тоже, что 'r2 malloc://512'
=          читать файл со стандартного ввода (используйте -i и -c для выполнения команд)
-=         флаг !=! позволяет удаленно запускать команды
-0         напечатать \x00 перед первой и после каждой команды
-2         закрыть файловый дескриптор stderr (тихий режим без вывода сообщений)
-a [arch]   установить asm.arch
-A         запуск команды 'aaa' анализа ссылок в коде
-b [bits]   задать asm.bits
-B [baddr]  задать базовый адрес для PIE-бинариков
-c 'cmd..'  запуск команды radare
-C         отобразить в файл host:port (alias for -c+=http://%s/cmd/)
-d         отлаживать запускаемый 'file' или процесс по 'pid'
-D [backend] установить режим отладки (е cfg.debug=true)
-e k=v     вычислить переменную конфигурации
-f         разместить приравнять к размеру файла
-F [binplug] использовать известный всем плагин rbin насилино
```

```

-h, -hh      показать сообщение о флагах и их предназначении, -hh - подробнее
-H ([var])  напечатать переменную
-i [file]    запустить файл-сценарий
-I [file]    запустить файл-сценарий ПЕРЕД открытием основного файла
-k [OS/kern] задать asm.os (linux, macos, w32, netbsd, ...)
-l [lib]     загрузить плагин из файла
-L          перечислить поддерживаемые плагины ввода-вывода
-m [addr]   отобразить файл по указанному адресу (loadaddr)
-M          не чудить с именами символов
-n, -nn    не загружать информацию RBin (-nn загружает только двоичные структуры)
-N          не загружать установки и сценарии пользователя
-q          тихий режим без командной строки, выйти после исполнения -i
-Q          тихий режим без командной строки, быстро затем выйти (quickLeak=true)
-p [prj]    использовать проект, список, если не задан аргумент, загрузить, если не указан основной файл
-P [file]   применить rapatch-файл и выйти
-r [rарun2] указать профиль rарun2 для загрузки (то же самое, что -e dbg.profile=X)
-R [rr2rule] указать дополнительную директиву rарun2
-s [addr]   начальный адрес смещения (seek)
-S          запуск r2 в режиме "песочницы"
-t          загрузить информацию rabin2 в "thread"
-u          задать bin.filter=false для получения "грубых" имен sym/sec/cls
-v, -V     показать версию radare2 (-V показывает версии библиотек)
-w          открыть файл в режиме перезаписи
-x          открыть без флага exec (asm.emu не будет работать), смотри io.exec
-X          то же самое, что -e bin.usetrue=false (используется в dyldcache)
-z, -zz    не загружать строки, или загружать их в грубом формате

```

Частые варианты использования

Открыть файл в режиме записи без анализа формата его заголовков.

```
$ r2 -nw file
```

Войти в оболочку r2, не открывая ни одного файла.

```
$ r2 -
```

Указать конкретный суббинарный файл в открываемом fatbin-файле:

```
$ r2 -a ppc -b 32 ls.fat
```

Запустить сценарий перед переходом в основную командную строку:

```
$ r2 -i patch.r2 target.bin
```

Выполнить команду и выйти, не входя в интерактивный режим:

```
$ r2 -qc ij hi.bin > imports.json
```

Установка значения переменной конфигурации:

```
$ r2 -e scr.color=0 blah.bin
```

Отладить программу:

```
$ r2 -d ls
```

Использовать существующий файл проекта:

```
$ r2 -p test
```

Формат команды

Общий формат команд radare2 выглядит следующим образом:

```
[.][times][cmd][~grep][@[@iter]addr!size][|>pipe] ;
```

Программисты, часто использующие Vim и знакомые с его командами, чувствуют теперь себя как дома. Этот формат используется на протяжении всей книги. Команды обозначаются одним символом [a-zA-Z], регистр учитывается.

Чтобы выполнить команду несколько раз, надо добавить к ней префикс с номером:

```
px      # выполнить px
3px    # выполнить px три раза
```

Предфикс ! используется для выполнения команды в оболочке операционной системы. If you want to use the cmd callback from the I/O plugin you must prefix with !=.

Обратите внимание, что один восклицательный знак запускает команду и распечатывает вывод через API RCons. Выполнение будет с блокировкой radare, интерактивный режим не будет работать. Двойные восклицательные знаки !! служат для запуска стандартных системных вызовов.

Все сокетные, файловые и исполнительные API можно контролировать (ограничивать) с помощью переменной конфигурации `cfg.sandbox`.

Примеры:

```
ds ; вызова команды отладчика 'step'  
px 200 @ esp ; показать 200 байт по адресу в регистре esp  
pc > file.c ; выдать в файл file.c буфер в виде массива байтов на языке С  
wx 90 @@ sym.* ; записать инструкцию por по адресу каждого символа  
pd 2000 | grep eax ; отфильтровать grep-ом коды инструкций, использующие регистр 'eax'  
px 20 ; pd 3 ; px 40 ; несколько команд в одной строке
```

Стандартный канал UNIX | также доступен в программе radare2. Он используется для фильтрации вывода команд r2 внешней программой, способной считывать со стандартного ввода, например, grep, less, wc. Если нет необходимости создавать лишние процессы, или это по какой-то причине невозможно, или же в оперативной системе нет необходимых вам инструментов UNIX (например, вы в Windows или используете встроенное приложение), вам доступен встроенный grep (~).

Используйте ~? для детальной документации команды.

Команда-фильтр `~` подобно `grep` используется для фильтрации вывода любой другой команды:

`pd 20~call` : дизассемблировать 20 инструкций, и отфильтровать grep-ом строки с 'call'

Кроме того, можно получать grep-ом строки и столбцы по номерам:

```
pd 20~call:0 ; первая строка  
pd 20~call:1 ; вторая строка  
pd 20~call[0] ; первая колонка  
pd 20~call[1] ; вторая колонка
```

Или даже комбинировать такие запросы:

`pd 20~call:0[0]`; отфильтровать grep-ом первую колонку первой строки, где встречается 'call'

Внутренняя функция `grep` является ключевой при создании сценариев `radare2`, она может использоваться для организации перебора списка смещений или данных, генерированных дизассемблером, диапазонов, или вывода других команд. Узнать об этом больше можно, изучив раздел Итераторы.

Символ **д** используется для указания параметра - адреса смещения для команды, которая записана в строке слева от него. После исполнения команды в такой форме, исходный адрес восстанавливается.

Например, pd 5 @ 0x100000fce дизассемблирует пять инструкций по адресу 0x100000fce.

Большинство команд поддерживают автодополнение по клавише `<TAB>`, например, `Seek` или `flags`. Автодополнение позволяет просмотреть все возможные варианты, в данном случае, имена флагов. Можно просмотреть историю команд с помощью команды `!~...`, она в визуальном режиме прокручивает список выполненных ранее команд `radare2`.

Также можно расширить объем предлагаемых в автодополнении вариантов до обработки большего из количества, а также включить автодополнение для ваших собственных команд, определенных в ядре, плагинах ввода-вывода. Вот эта команда - `!!`.

Выражения

Выражение --- это математическое представление 64-битных числовых значений. Результаты выражений отображаются в разных форматах, сравниваются и могут быть использованы со всеми командами, принимающими числовые аргументы. Выражения используют традиционные арифметические операции, а также двоичные и логические значения. Чтобы вычислить математическое выражение в команде, надо добавить к ней ?:

Поддерживаемые арифметические операции:

- + : сумма
- - : вычитание
- * : умножение
- / : деление
- % : остаток от деления по модулю
- >> : сдвиг вправо
- << : сдвиг влево

```
[0x00000000]> ?vi 1+2+3
```

6

При использовании двоичного ИЛИ, всю команду следует заключать в кавычки, иначе выполнится связывание потоков | :

```
[0x00000000]> "? 1 | 2"
```

```
hex      0x3
octal   03
unit     3
segment 0000:0003
int32    3
string  "\x03"
binary   0b00000011
fvalue: 2.0
float:  0.000000f
double: 0.000000
trits   0t10
```

Числа могут отображаться в одном из следующих форматов:

```
0x033  : шеснадцатеричное число просто отображается
3334   : десятичное
sym.fo : преобразуется в смещение флага
10K    : килобайты (KBytes) 10*1024
10M    : мегабайты (MBytes) 10*1024*1024
```

Также можно использовать переменные и адреса, включая текущее смещение, при построении сложных выражений.

Последовательность символов ?\$? выдает список всех доступных команд. Можно также ознакомиться с кратким перечнем команд в конце книги.

```
$$  here (текущий виртуальный адрес, смещение)
$l  длина кода инструкции (opcode length)
$s  размэр файла
$j  адрес перехода (например, jmp 0x10, jz 0x10 => 0x10)
$f  адрес перехода, если ложь (например, jz 0x10 => адрес следующей инструкции)
$m  ссылка на адрес памяти, выраженная кодом инструкции (opcode) (например, mov eax,[0x10] => 0x10)
$b  размэр блока
```

Еще несколько примеров:

```
[0xA13B8C0]> ? $m + $l
140293837812900 0x7f98b45df4a4 03771426427372244 130658.0G 8b45d000:04a4 140293837812900 10100100 140293837812900.0
0.000000
```

```
[0xA13B8C0]> pd 1 @ +$l
0x4A13B8C2  call 0x4a13c000
```

Основные команды отладчика

Запуск радара в режиме отладки осуществляется при помощи флага -d в командной строке операционной системы. Обычно производится запуск новой программы, указывая ее имя и параметры, можно также и присоединиться к выполняющемуся процессу, указав его PID.

```
$ pidof mc
32220
$ r2 -d 32220
$ r2 -d /bin/ls
$ r2 -a arm -b 16 -d gdb://192.168.1.43:9090
...
```

Во втором случае отладчик создаст дочерний процесс и загрузит в него программу ls для отладки.

Программа остановится в момент загрузки ld.so динамическим компоновщиком. В этот момент точка входа еще не будет видна и разделяемые библиотеки тоже.

Переопределить такое функционирование можно, задав другой адрес для останова - адрес точки входа по ее имени. Делается это добавлением команды радара `e dbg.bep=entry` или `e dbg.bep=main` в ваш startup-сценарий, обычно он находится в файле `~/.config/radare2/radare2rc`.

Другой способ - запустить продолжения исполнения до определенного адреса - команда `dcu`. Команда означает «отладку продолжать до тех пор, пока» (`debug continue until`) не будет достигнут адрес, где нужно остановить исполнение. Например:

```
dcu main
```

Имейте в виду, что определенные вредоносные или другие замороченные программы могут выполнять некоторый код до запуска `main()`, что не даст возможности контролировать вам исполнение этого кода. (Также и конструкторы программ и TLS-инициализации)

Ниже приведен список наиболее распространенных команд, используемых с отладчиком:

```
> d?          ; показать подсказку по командам отладчика  
> ds 3        ; сделать три шага (step)  
> db 0x8048920 ; установить точку останова (set breakpoint)  
> db -0x8048920 ; удаление точки останова  
> dc          ; продолжить исполнение процесса (continue)  
> dcs         ; продолжить исполнение до запуска системного вызова (continue until syscall)  
> dd          ; проводить манипуляции с файловыми дескрипторами  
> dm          ; показать карты процесса (show process maps)  
> dmp A S rwx ; поменять ограничения для страницы по адресу A размером S  
> dr eax=33    ; присвоить значение регистру (set register value). eax = 33
```

Есть еще один вариант отладки в радаре, который проще - использование визуального режима.

При его использовании не нужно ни запоминать команды, ни держать в уме состояние программы.

Чтобы перейти в режим визуального отладчика, используйте `Vpp`:

```
[0xb7f0c8c0]> Vpp
```

Начальное вид экрана в визуальном режиме представляет собой шестнадцатеричный вид текущего счетчика отлаживаемой программы (например, EIP для архитектуры x86). Нажатие `p` переключает между разными представлениями в визуальном режиме. Можно нажимать `p` и `P` для смены режимов различных вариантов визуализации. Используйте `F7` или `S`, чтобы войти внутрь функции и `F8` или `S` для перешагивания через текущую инструкцию. Ключ `C` позволяет переключаться в режим курсора. В этом режиме можно выделять диапазоны байтов, например, чтобы потом перезаписать их пор-пами. Установка точки останова - клавиша `F2`.

В визуальном режиме можно вводить обычные команды радара, добавляя к ним префикс `:`. Например, чтобы сбросить один блок содержимого памяти по адресу в регистре `ESI`:

```
<Нажать ':'>  
x @ esi
```

Получение справки по визуальному режиму - клавиша `?`. Для прокрутки экрана справки используются стрелки. Если надо выйти из режима справки, используйте клавишу `q`.

Часто используемая команда - `dr`, она используется для чтения или установки значений регистров общего назначения. Еще есть более компактное представление значения регистра - команда `dr=`. Можно манипулировать также аппаратными регистрами, расширенными регистрами, регистрами для чисел с плавающей запятой.

Улучшение книги Radare2

Книга Radare2

Если есть желание внести свой вклад в книгу Radare2, можно сделать это на сайте Github. Предлагаемые варианты развития:

- Обзоры упражнений Crackme,
- Обзоры CTF,
- Документация по использованию Radare2,
- Документация по разработке для Radare2,
- Презентации на конференциях и семинарах о Radare2,
- Портирование отсутствующего контента из книги Radare1 в Radare2.

Не забудьте получить разрешение на перенос контента, которым вы не владеете лично и лично не создавали, прежде чем поместить его в книгу Radare2.

Ознакомьтесь с общей инструкцией <https://github.com/radareorg/radare2/blob/master/DEVELOPERS.md> о развитии проекта Radare2.

Конфигурация

Конфигурация radare2 выполняется с помощью команд установки переменных среды. Ядро считывает и интерпретирует `~/.config/radare2/radare2rc` во время запуска. Можно добавлять е-команды в этот файл для настройки конфигурации Radare2 по своему вкусу. Для предотвращения загрузки этого файла при запуске, установите флаг командной строки `-N`. Типичный файл конфигурации запуска выглядит так:

```
$ cat ~/.radare2rc
e scr.color = 1
e dbg.bep = loader
```

Конфигурацию также можно изменять с помощью флагов формата `-e <config=value>` в строке параметров при запуске. Таким образом, можно подправить конфигурацию из командной строки, сохранив файл `.radare2rc` не тронутым. Например, чтобы начать с пустой конфигурации, а затем настроить `scr.color` а также `asm.syntax` можно использовать следующую строку:

```
$ radare2 -N -e scr.color=1 -e asm.syntax=intel -d /bin/ls
```

Внутри r2 конфигурация хранится в хеш-таблице. Переменные сгруппированы в пространства имен: `cfg.`, `file.`, `dbg.`, `scr.` и так далее.

Чтобы получить список всех переменных конфигурации, просто введите `e` в командной строке r2. Чтобы ограничить вывод выбранным пространством имен, передайте его с точкой в конце `e <имя пространства>`. Например, `e file.` отобразит все переменные, определенные внутри пространства имен «`file`».

Чтобы получить инструкции о команде `e - e?`:

```
Usage: e [var[=value]] # Переменные
| e?asm.bytes      показать описание
| e??              перечислить конфигурационные переменные с описаниями
| e a              получить значение переменной 'a'
| e a=b            установить значение переменной 'a' равным 'b'
| e var=?          показать все возможные значения переменной
| e var==?         показать все возможные значения переменной с описаниями
| e.a=b            аналогично 'e a=b', но без использования пробела
| e,k=v,k=v,k=v   разделение запятой k[=v]
| e-              сбросить значения к начальным
| e*              сделать дамп переменных конфигурации в виде команд r
| e!a              инвертировать логическое значение переменной 'a'
| ec [k] [color]  установить цвет для заданного ключа (prompt, offset, ...)
| eevar           открыть редактор для изменения переменных
| ed              открыть редактор для изменения ~/.radare2rc
| ej              перечислить конфигурационные переменные в формате JSON
| env [k[=v]]     получить/установить значение переменной среды
| er [key]         установить режим доступа к конфигурационному ключу "только для чтения", и ... нет пути назад
| es [space]       показать все пространства конфигурационных переменных (не ключей)
| et [key]         показать тип заданной конфигурационной переменной
| ev [key]         перечислить конфигурационные переменные в подробном виде
| evj [key]        перечислить конфигурационные переменные в подробном виде в формате JSON
```

Более простая альтернатива команде `e` доступна из визуального режима. Наберите `Ve` для входа в этот режим, используйте стрелки (вверх, вниз, влево, вправо) для навигации по конфигурации и `Q` - выход из этого режима. В начале экрана визуального редактирования конфигурации выглядит так:

[EvalSpace]

```
> anal
asm
scr
asm
bin
cfg
diff
dir
dbg
cmd
fs
hex
http
graph
hud
scr
search
io
```

Для значений конфигурации, которые могут принимать одно из нескольких значений, вы можете использовать `=?`, оператор для получения списка допустимых значений:

```
[0x00000000]> e scr.nkey = ?
scr.nkey = fun, hit, flag
```

Цветовая раскраска в интерфейсе

Консольный доступ заключен в API, позволяющий отображать выходные данные любой команды при помощи ANSI, W32 Console или HTML. Это позволяет ядру радара работать в средах с ограниченными возможностями отображения, таких как командная строка и встроенные устройства. С них по-прежнему можно получать данные в предпочтитаемом формате.

Чтобы включить поддержку цветов по умолчанию, добавьте соответствующий параметр конфигурации в конфигурационный файл .radare2:

```
$ echo 'e scr.color=1' >> ~/.radare2rc
```

Обратите внимание, что включение цветов не является логическим параметром, а числом, обозначающим разные уровни глубины цвета, *режимы*:

- 0: Черно-белый,
- 1: 16 основных цветов ANSI,
- 2: 256 цветов в виде шкалы,
- 3: 24-битный truecolor.

Причина наличия таких настроек заключается в том, что нет стандартного или портативного способа для терминальных программ запрашивать консоль для определения наилучшей конфигурации, то же самое касается кодировок наборов символов, поэтому r2 позволяет выбрать это вручную.

Обычно последовательные консоли работают с режимами 0 или 1, в то время как xterms могут поддерживать до уровня 3. RCons попытается найти наиболее близкую цветовую схему для вашей темы, когда вы выбираете новую тему с помощью команды `e c0`.

Есть возможность настроить цвет практически любого элемента вывода дизассемблера. Для терминалов *NIX r2 принимает спецификацию цвета в формате RGB. Для изменения цветовой палитры консоли используйте команду `e cs`. Команда `e cs` отображает цветовую палитру, помогающую с выбором цветов:

Темы

Можно создать свою собственную цветовую тему, хотя у radare2 итак есть уже предопределенные. Используйте команду `e c0`, чтобы перечислить и выбрать одну из их. После выбора одной можно сравнить ее цветовую схему с текущей темой, нажав клавиши **CTRL+SHIFT**, а затем клавишу со стрелкой вправо для переключателя. В визуальном режиме используйте клавишу **R** для генерирования случайной палитры цветов или выбора следующей темы в списке.

Переменные конфигурации

Ниже приведен список наиболее часто используемых конфигурационных переменных. Полный список получается, выполнив команду `e` без аргументов. Например, чтобы посмотреть все переменные, определенные в пространстве имен ``cfg'', задайте `e cfg.` (обратите внимание на конечную точку). Получить инструкцию по любой переменной конфигурации - `e? cfg..` Команда `e??` показывает описание всех ценных переменных конфигурации radare2. Количество выдаваемых данных этой команды довольно велико, можно фильтровать их внутренним `grep`-ом `~`:

Визуальный интерфейс включает браузер переменных среды, доступный с помощью команды `Vbe`.

Переменная `asm.arch`

Эта переменная задает архитектуру целевого процессора для дизассемблирования (`pd`, `pD`) и анализа кода (`a`). Список возможных значений для конкретной переменной показывается при помощи `e asm.arch=?` или `rasm2 -L`.

Добавить новые архитектуры для разборки и анализа кода довольно просто. Для этого есть специальный интерфейс. Для x86 он используется для присоединения ряда сторонних дизассемблерных механизмов, включая GNU binutils, Udis86 и несколько реализованных вручную.

Переменная `asm.bits`

Определяет размер регистров в битах для текущей архитектуры. Поддерживаемые значения: 8, 16, 32, 64. Обратите внимание, что не все целевые архитектуры поддерживают все комбинации `asm.bits`.

Переменная `asm.syntax`

Выбирает синтаксический диалект дизассемблера между Intel и AT&T. На данный момент этот параметр влияет на дизассемблер Udis86 только для архитектур Intel 32/Intel 64. Поддерживаются значения `intel` и `att`.

[0x00000000] > ecs					
black					
red					
white					
green					
magenta					
yellow					
cyan					
blue					
gray					
Greyscale:					
rgb:000	rgb:111	rgb:222	rgb:333	rgb:444	rgb:555
rgb:666	rgb:777	rgb:888	rgb:999	rgb:aaa	rgb:bbb
rgb:ccc	rgb:ddd	rgb:eee	rgb:fff		
RGB:					
rgb:000	rgb:030	rgb:060	rgb:090	rgb:0c0	rgb:0f0
rgb:003	rgb:033	rgb:063	rgb:093	rgb:0c3	rgb:0f3
rgb:006	rgb:036	rgb:066	rgb:096	rgb:0c6	rgb:0f6
rgb:009	rgb:039	rgb:069	rgb:099	rgb:0c9	rgb:0f9
rgb:00c	rgb:03c	rgb:06c	rgb:09c	rgb:0cc	rgb:0fc
rgb:00f	rgb:03f	rgb:06f	rgb:09f	rgb:0cf	rgb:0ff
rgb:300	rgb:330	rgb:360	rgb:390	rgb:3c0	rgb:3f0
rgb:303	rgb:333	rgb:363	rgb:393	rgb:3c3	rgb:3f3
rgb:306	rgb:336	rgb:366	rgb:396	rgb:3c6	rgb:3f6
rgb:309	rgb:339	rgb:369	rgb:399	rgb:3c9	rgb:3f9
rgb:30c	rgb:33c	rgb:36c	rgb:39c	rgb:3cc	rgb:3fc
rgb:30f	rgb:33f	rgb:36f	rgb:39f	rgb:3cf	rgb:3ff
rgb:600	rgb:630	rgb:660	rgb:690	rgb:6c0	rgb:6f0
rgb:603	rgb:633	rgb:663	rgb:693	rgb:6c3	rgb:6f3
rgb:606	rgb:636	rgb:666	rgb:696	rgb:6c6	rgb:6f6
rgb:609	rgb:639	rgb:669	rgb:699	rgb:6c9	rgb:6f9
rgb:60c	rgb:63c	rgb:66c	rgb:69c	rgb:6cc	rgb:6fc
rgb:60f	rgb:63f	rgb:66f	rgb:69f	rgb:6cf	rgb:6ff
rgb:900	rgb:930	rgb:960	rgb:990	rgb:9c0	rgb:9f0
rgb:903	rgb:933	rgb:963	rgb:993	rgb:9c3	rgb:9f3
rgb:906	rgb:936	rgb:966	rgb:996	rgb:9c6	rgb:9f6
rgb:909	rgb:939	rgb:969	rgb:999	rgb:9c9	rgb:9f9
rgb:90c	rgb:93c	rgb:96c	rgb:99c	rgb:9cc	rgb:9fc
rgb:90f	rgb:93f	rgb:96f	rgb:99f	rgb:9cf	rgb:9ff
rgb:c00	rgb:c30	rgb:c60	rgb:c90	rgb:cc0	rgb:cf0
rgb:c03	rgb:c33	rgb:c63	rgb:c93	rgb:cc3	rgb:cf3
rgb:c06	rgb:c36	rgb:c66	rgb:c96	rgb:cc6	rgb:cf6
rgb:c09	rgb:c39	rgb:c69	rgb:c99	rgb:cc9	rgb:cf9
rgb:c0c	rgb:c3c	rgb:c6c	rgb:c9c	rgb:ccc	rgb:cfc
rgb:c0f	rgb:c3f	rgb:c6f	rgb:c9f	rgb:ccf	rgb:cff
rgb:f00	rgb:f30	rgb:f60	rgb:f90	rgb:fc0	rgb:ff0
rgb:f03	rgb:f33	rgb:f63	rgb:f93	rgb:fc3	rgb:ff3
rgb:f06	rgb:f36	rgb:f66	rgb:f96	rgb:fc6	rgb:ff6
rgb:f09	rgb:f39	rgb:f69	rgb:f99	rgb:fc9	rgb:ff9
rgb:f0c	rgb:f3c	rgb:f6c	rgb:f9c	rgb:fcc	rgb:ffc
rgb:f0f	rgb:f3f	rgb:f6f	rgb:f9f	rgb:fcf	rgb:fff
[0x00000000] >					

Рис. 3: img

```
[0x100001200]> e??~color
graph.gv.graph: Graphviz global style attributes. (bgcolor=white)
graph.gv.node: Graphviz node style. (color=gray, style=filled shape=box)
scr.color: Enable colors (0: none, 1: ansi, 2: 256 colors, 3: truecolor)
scr.color.args: Colorize arguments and variables of functions
scr.color.bytes: Colorize bytes that represent the opcodes of the instruction
scr.color.grep: Enable colors when using -grep
scr.color.ops: Colorize numbers and registers in opcodes
scr.pipecolor: Enable colors when using pipes
scr.rainbow: Shows rainbow colors depending of address
scr.randpal: Random color palette or just get the next one from 'eco'
```

Рис. 4: e??~color

Переменная `asm.pseudo`

Это логическое значение для задания псевдо-синтаксиса при дизассемблировании. Значение «`false`» указывает на ``родной'', определенный текущей архитектурой, «`true`» активирует формат строк псевдокода. Например, переменная ``преобразует'':

0x080483ff	e832000000	call 0x8048436
0x08048404	31c0	xor eax, eax
0x08048406	0205849a0408	add al, byte [0x8049a84]
0x0804840c	83f800	cmp eax, 0
0x0804840f	7405	je 0x8048416

B		
0x080483ff	e832000000	0x8048436 ()
0x08048404	31c0	eax = 0
0x08048406	0205849a0408	al += byte [0x8049a84]
0x0804840c	83f800	var = eax - 0
0x0804840f	7405	if (!var) goto 0x8048416

Переменная полезна при дизассемблировании малоизвестных архитектур.

Переменная `asm.os`

Задает целевую операционную систему загруженного в данный момент двоичного файла. Обычно ОС автоматически определяется `rabin -rI`. Изменение значение `asm.os` используется для переключения на таблицу системных вызовов конкретной операционной системы.

Переменная `asm.flags`

Если задано значение ```true`'', представление дизассемблера будет включать столбец флагов.

Переменная `asm.lines.call`

Если задано значение ```true`'', рисует линии слева от вывода дизассемблирования (`pd`, `pD`), чтобы графически представить поток управления (переходы и вызовы), которые входят внутрь текущего блока. Кроме того, смотрите инструкцию в `asm.lines.out`.

Переменная `asm.lines.out`

Если она равна «`true`», представление дизассемблера также будет рисовать потоки, выходящие за пределы блока.

Переменная `asm.linestyle`

Логическое значение, изменяющее направление анализа потока управления. Если задано значение ```false`'', производится сверху вниз блока; в противном случае он идет снизу вверх. Настройка «`false`», по-видимому, является лучшим выбором для улучшения читаемости и является настройкой по умолчанию.

Переменная `asm.offset`

Логическое значение, которое управляет видимостью адресов смещений для отдельно дизассемблированных инструкций.

Переменная `asm.trace`

Логическое значение, управляющее отображением информации трассировки (порядковый номер и счетчик) слева от каждого оп-кода. Используется в анализе трассировки программ.

Переменная asm.bytes

Логическое значение, используемое для отображения или скрытия raw-байтов инструкций.

Переменная asm.sub.reg

Логическое значение, используемое для замены имен регистров аргументами или связанной с его псевдонимом роли.

Например, если у вас есть что-то вроде этого:

0x080483ea	83c404	add esp, 4
0x080483ed	68989a0408	push 0x8049a98
0x080483f7	e870060000	call sym.imp.printf
0x080483fc	83c408	add esp, 8
0x08048404	31c0	xor eax, eax

Переменная изменяет представление на:

0x080483ea	83c404	add SP, 4
0x080483ed	68989a0408	push 0x8049a98
0x080483f7	e870060000	call sym.imp.printf
0x080483fc	83c408	add SP, 8
0x08048404	31c0	xor A0, A0

Переменная asm.sub.jmp

Логическое значение, используемое для замены целевых объектов перехода, вызова и ветвления при дизассемблировании. Например, во включенном режиме он будет отображать `jal 0x80001a40` как `jal fcn.80001a40` в дизассемблировании.

Переменная asm.sub.rel

Логическое значение, которое заменяет относительные выражения `pc` при дизассемблировании. Если этот параметр включен, ссылки отображаются как строковые ссылки, например:

`0x5563844a0181 488d3d7c0e00. lea rdi, [rip + 0xe7c] ; str.argv__2d__:__s`

Если эта переменная включена, она позволяет отобразить приведенную выше инструкцию следующим образом:

`0x5563844a0181 488d3d7c0e00. lea rdi, str.argv__2d__:__s ; 0x5563844a1004 ; "argv[%2d]: %s\n"`

Переменная asm.sub.section

Логическое значение, показывающее смещение в дизассемблировании с префиксом названия раздела или карты. Это означает что-то вроде:

`0x000067ea 488d0def0c01. lea rcx, [0x000174e0]`

то же самое, но со включенным режимом.

`0x000067ea 488d0def0c01. lea rcx, [fmap.LOAD1.0x000174e0]`

Переменная asm.sub.varonly

Заменять ли выражение переменной именем локальной переменной. Например, `var_14h` как `rbp - var_14h` в дизассемблировании.

Переменная cfg.bigendian

Изменение порядка байтов: «`true`» означает биг-эндиан, «`false`» - литл-эндиан, причем «`file.id`» и «`file.flag`» должны быть «`true`».

Переменная cfg.newtab

Если эта переменная включена, сообщения справки будут отображаться вместе с именами команд в автодополнении.

Переменная scr.color

Эта переменная задает режим вывода «в цвете» или нет: «`false`» (или 0) означает отсутствие цветов, «`true`» (или 1) означает 16-цветной режим, 2 означает 256-цветовой режим, 3 означает режим «о 16-ти миллионов цветов». Если вами предпочтаемая тема выглядит странно, попробуйте увеличить число.

Переменная scr.seek

Переменная принимает выражение или указатель/флаг (например, `eip`). Если установлено, radare установит начальное смещение в это значение при запуске.

Переменная scr.scrollbar

Если настроены какие-либо зоны флагов (`f z ?`), эта переменная позволит вам отобразить полосу прокрутки с флаг-зонами в визуальном режиме. Установите значение в **1**, чтобы отобразить полосу прокрутки справа, **2** для верхней части экрана и **3** - снизу.

Переменная scr.utf8

Логическая переменная для отображения символов UTF-8 вместо ANSI.

Переменная cfg.fortunes

Включает или отключает сообщения «высказываний Фортуны», отображаемые при каждом запуске радара.

Переменная cfg.fortunes.type

Высказывания Фортуны классифицируются по типу. Эта переменная определяет, какие типы разрешены для отображения, когда `cfg.fortunes true`, их можно настроить на целевую аудиторию. Текущие типы - это советы (`tips`), веселье (`fun`), нсфв (`nsfw`), жуткие (`creepy`).

Переменная stack.size

Эта переменная позволяет задать размер стека в байтах.

Файлы radare2

Используйте `r2 -H` для перечисления всех переменных среды, управляющих поиском файлов на диске. Путь поиска зависит от настроек вашей сборки r2 и операционной системы.

```
R2_PREFIX=/usr
MAGICPATH=/usr/share/radare2/2.8.0-git/magic
PREFIX=/usr
INCDIR=/usr/include/libr
LIBDIR=/usr/lib64
LIBEXT=so
RCONFIGHOME=/home/user/.config/radare2
RDATAHOME=/home/user/.local/share/radare2
RCACHEHOME=/home/user/.cache/radare2
LIBR_PLUGINS=/usr/lib/radare2/2.8.0-git
USER_PLUGINS=/home/user/.local/share/radare2/plugins
USER_ZIGNS=/home/user/.local/share/radare2/zigns
```

Файлы RC

RC-файлы --- это сценарии r2, которые загружаются во время запуска. Эти файлы должны находиться в трех разных местах:

- Системная, radare2 сначала попытается загрузить `/usr/share/radare2/radare2rc`,
- В домашнем директории.

Каждый пользователь в системе может настраивать свои собственные сценарии r2 для запуска, например, выбор цветовой схемы, а также другие параметры, задаваемые там командами r2.

- `~/.radare2rc`,
- `~/.config/radare2/radare2rc`,
- `~/.config/radare2/radare2rc.d/`.

Целевой файл

Если надо запускать сценарий каждый раз при открытии файла, просто создайте файл с тем же именем, что и бинарный файл, но добавив к нему `.r2`.

Основные команды

Большинство названий команд в radare являются производными от названий действий. Их легко запомнить, так как они короткие. На самом деле, все команды являются одиночными буквами. Подкоманды или связанные команды задаются с помощью второго символа в названии. Например, / foo --- это команда для поиска строки, а /x 90 90 используется для поиска шестнадцатеричных пар.

Общий формат команды как описано в главе Формат команды выглядит следующим образом:

```
[.][times][cmd][~grep][@[@iter]addr!size][|>pipe] ; ...
```

Например,

```
> 3s +1024 ; ищет три раза 1024 из текущего смещения
```

Если команда начинается с !=, остальная часть строки передается загруженному в данный момент подключаемому модулю (например, отладчику). Большинство плагинов предоставляют инструкции при помощи !=? или !=!help.

```
$ r2 -d /bin/ls  
> !=!help ; обрабатывается плагином ввода-вывода
```

Если команда начинается с !, вызывается posix_system() для передачи команды в оболочку. Для получения дополнительных вариантов и примеров использования есть команда !?.

```
> !ls ; запустить `ls` в операционной системе
```

Значение аргументов iter, addr, size зависит от конкретной команды. Как правило, большинство команд принимают число в качестве аргумента, чтобы указать количество байтов для работы, а не текущий размер блока. Некоторые команды принимают математические выражения или строки.

```
> px 0x17 ; показать 0x17 байт в шестнадцатеричном виде по текущему адресу,  
> s base+0x33 ; установить смещение на флаг 'base' плюс 0x33 байта,  
> / lib ; искать строку 'lib'.
```

Знак @ используется для указания временного адреса или смещения, в котором выполняется команда, вместо текущего смещения. Очень удобно, так как вам не нужно всякий раз переустанавливать смещение.

```
> p8 10 @ 0x4010 ; показать 10 байт по смещению 0x4010  
> f patata @ 0x10 ; ассоциировать флаг 'patata' со смещением 0x10
```

Используя @@, можно выполнить одну команду над каждым элементом списка флагов, подобно glob. Можете воспринимать это как foreach:

```
> s 0  
> / lib ; поиск строки 'lib'  
> p8 20 @@ hit@_* ; показать 20 байт в шестнадцатеричном виде по каждому найденному адресу
```

Операция > используется для перенаправления выходных данных команды в файл (его перезаписи, если он уже существует).

```
> pr > dump.bin ; сделать дамп 'raw'-байтов текущего блока в файл с именем 'dump.bin'  
> f > flags.txt ; сделать дамп списка флагов в 'flags.txt'
```

Операция | (канал) аналогична тому, что в операционной системе *NIX: перенаправить вывод одной команды на вход для другой.

```
[0x4A13B8C0]> f | grep section | grep text  
0x0805f3b0 512 section._text  
0x080d24b0 512 section._text_end
```

Можно выполнить несколько команд в одной строке, разделив их точкой с запятой ;:

```
> px ; dr
```

При помощи _ можно распечатать результат, полученный последней командой.

```
[0x000001060]> axt 0x000002004  
main 0x1181 [DATA] lea rdi, str.argv__2d:_  
[0x000001060]> _  
main 0x1181 [DATA] lea rdi, str.argv__2d:_  
_s
```

Установка базового смещения

Чтобы перемещаться по проверяемому файлу, т.е. менять адрес смещения, требуется изменять смещение при помощи команды S. Аргумент представляет собой математическое выражение, которое может содержать имена флагов, скобки, сложение, вычитание, умножение непосредственных значений содержимого памяти с помощью скобок.

Несколько примеров использования команд:

```
[0x00000000]> s 0x10
[0x00000010]> s+4
[0x00000014]> s-
[0x00000010]> s+
[0x00000014]>
```

Посмотрите, как изменяется смещение (`seek`) в левой части командной строке. Первая строка устанавливает текущее смещение по адресу `0x10`. Вторая смещает адрес на четыре байта вперед (переместиться относительно предыдущего смещения). И, наконец, последние две команды отменяют (`undo`) и заново выполняют (`redo`) последние операции установки адреса (`seek`). Вместо использования исключительно числовых значений, можно использовать сложные выражения и обычные арифметические операции для вычисления адреса. Ознакомьтесь с инструкцией, предоставляемой командой `?$?`, там описаны переменные, которые можно использовать в адресных выражениях. Например, следующие выражение делает то же самое что и `s +4`.

```
[0x00000000]> s $$+4
```

В отладчике (или в режиме эмуляции ESIL) можно использовать имена регистров в качестве адресов. Имена регистров --- это флаги, их перечень доступен при помощи `.dr*`.

```
[0x00000000]> s rsp+0x40
```

Приведем справку по командам группы `s`. Более детально они будут рассмотрены на примерах далее.

```
[0x00000000]> s?
Usage: s      # Инструкция по командам смены смещения. Смотрите ?$? - перечень всех переменных
| s          Показать текущее смещение (адрес)
| s.hexoff   Задать смещение, используя систему счисления из core->offset
| s:pad      Показать текущее смещение в поле N символов (по умолчанию 8), заполняя пустое место нулями
| s addr     Установить смещение по заданному адресу
| s-         Отменить предыдущую операцию установки смещения
| s-*        Вернуть ранее отмененную операцию смещения
| s- n       Переместиться на n в обратном направлении
| s--[n]     Переместиться на blocksize байт в обратном направлении (/=n)
| s+         Вернуть ранее отмененную операцию смещения
| s+ n      Переместиться на n дальше
| s++[n]    Переместиться на blocksize байт вперед (/=n)
| s[j*=!]   Показать историю отмен операций смещения (JSON, =list, *r2, !=names, s==)
| s/ DATA    Поиск следующего вхождения 'DATA'
| s/x 9091   Поиск следующего вхождения \x90\x91
| sa [[+-]a] [asz] Установить смещение asz (или bsize), выровненный на адрес
| sb          Установить смещение выровненным на начало базового блока
| sc[?] string Установить смещение на комментарий, соответствующий заданной строке
| sf          Установить смещение на следующую функцию (f->addr+f->size)
| sf function Установить смещение на адрес заданной функции
| sf.         Установить адрес смещения на начало текущей функции
| sg/sG      Установить смещение на начало (sg) или конец (sG) секции или файла
| sl[?] [+/-]line Установить смещение на строку
| sn/sp ([nkey]) Установить смещение следующее/предыдущее место, в соответствии с scr.nkey
| so [N]     Установить смещение на N-ый следующий по-код
| sr pc     Установить смещение на адрес в регистре
| ss          Установить смещение без добавления записи в историю адресов смещений

> 3s++      ; выполнить три раза операцию установки смещения к следующему блоку
> s 10+0x80  ; установить смещение на 0x80+10
```

Результат математического выражения можно посмотреть при помощи команды `?`. Выражение передается в качестве аргумента. Результат отображается в шестнадцатеричном, десятичном, восьмеричном и двоичном форматах.

```
> ? 0x100+200
0x1C8 ; 456d ; 710o ; 1100 1000
```

Существуют также подкоманды `?`, отображающие результаты в одном конкретном формате (по базе 10, 16, ...). Смотрите `?v` и `?vi`.

В визуальном режиме можно нажать `u` (`undo`) или `U` (`redo`) внутри истории адресов смещений --- перемещение в обоих направлениях по адресам смещений.

Открыть файл

Рассмотрим работу команд на простом примере `hello_world.c`, скомпилированном в Linux ELF. После сборки компьютером, откроем его radare2-ом:

```
$ r2 hello_world
```

Теперь у нас есть командная строка:

```
[0x00400410]>
```

Перемещение смещения на любые адреса

Все команды перемещения (seek), принимающие адрес в качестве параметра, могут использовать любую систему счисления, например, шестнадцатеричную, восьмеричную, двоичную или десятичную.

Установка адреса 0x0. Альтернативный вариант --- просто 0x0:

```
[0x00400410]> s 0x0
[0x00000000]>
```

Показать текущий адрес:

```
[0x00000000]> s
0x0
[0x00000000]>
```

Существует альтернативный способ печати текущей позиции - ?v \$\$.

Перемещение на N позиций вперед, пробел необязателен:

```
[0x00000000]> s+ 128
[0x00000080]>
```

Отмена последних двух смещений, возврат к исходному адресу:

```
[0x00000080]> s-
[0x00000000]> s-
[0x00400410]>
```

Мы вернулись к 0x00400410.

Теперь можно показать историю адресов смещений:

```
[0x00400410]> s*
f undo_3 @ 0x400410
f undo_2 @ 0x40041a
f undo_1 @ 0x400410
f undo_0 @ 0x400411
# Current undo/redo position. # Текущее положение
f redo_0 @ 0x4005b4
```

Размер блока

Размер блока определяет, сколько байт будут обрабатывать команды radare2, если не задан явный аргумент размера. Можно временно изменять размер блока, указав числовой аргумент для команд печати. Например, px 20.

```
[0x00000000]> b?
Usage: b[f] [arg] # Узнать/установить размер блока
| b 33      установить размер блока 33
| b eip+4   числовой аргумент может быть выражением
| b        показать текущий размер блока
| b+3     увеличить размер блока на 3
| b-16    уменьшить размер блока на 16
| b*      показать текущий размер блока в r2
| bf foo   установить размер блока равным размеру флага
| bj      представить информацию о размере блока в виде JSON
| bm 1M    установить максимальный размер блока
```

Команда b используется для изменения размера блока:

```
[0x00000000]> b 0x100  # размер блока = 0x100
[0x00000000]> b+16   # ... = 0x110
[0x00000000]> b-32   # ... = 0xf0
```

Команда bf используется для изменения размера блока на определенное значение, связанное с флагом. Например, в символах размер блока флаг представляет размер функции. Чтобы это работало, нужно запустить анализ функций af, входящий в aa, или вручную найти и определить функции при помощи Vd.

```
[0x00000000]> bf sym.main  # размер блока = sizeof(sym.main)
[0x00000000]> pD @ sym.main  # дизассемблировать sym.main
```

Совместить две операции в одной --- команда pdf. Кроме pdf никакие другие команды не влияют на размер глобального блока.

```
[0x00000000]> pdf @ sym.main  # дизассемблировать sym.main
```

Другой способ --- использовать специальные переменные \$FB и \$FS, обозначающие начало и размер функции при выполнении seek. Подробнее ищите в Используемые переменные.

```
[0x00000000]> s sym.main + 0x04
[0x00001ec9]> pD @ $FB !$FS # дизассемблировать текущую функцию
211: int main (int argc, char **argv, char **envp);
    0x00001ec5      55          push rbp
    0x00001ec6      4889e5      mov rbp, rsp
    0x00001ec9      4881ecc0000000 sub rsp, 0xc0
...
    0x00001f97      c3          ret
```

Примечание: не ставьте пробел после !. Смотрите также Формат команды.

Секции

Понятие секций (разделов) привязано к информации, извлекаемой из двоичного файла. Наиболее часто --- это исполняемый файл. Двоичными файлами в этой книге также обозначаются скомпилированные объектные файлы и им подобные. Информация о секциях отображается с помощью команды `i`.

Отображение информации о секциях:

```
[0x00005310]> iS
[Sections]
00 0x00000000    0 0x00000000    0 ----
01 0x00000238    28 0x00000238    28 -r-- .interp
02 0x00000254    32 0x00000254    32 -r-- .note.ABI_tag
03 0x00000278    176 0x00000278   176 -r-- .gnu.hash
04 0x00000328    3000 0x00000328   3000 -r-- .dynsym
05 0x00000ee0    1412 0x00000ee0   1412 -r-- .dynstr
06 0x00001464    250 0x00001464   250 -r-- .gnu.version
07 0x00001560    112 0x00001560   112 -r-- .gnu.version_r
08 0x000015d0    4944 0x000015d0   4944 -r-- .rela.dyn
09 0x00002920    2448 0x00002920   2448 -r-- .rela.plt
10 0x000032b0    23 0x000032b0    23 -r-x .init
...
...
```

Как вы, возможно, знаете, двоичные файлы состоят из секций и карт памяти. Секции определяют содержимое части файла, отображаемое в память (или нет). То, что отображается, определяется сегментами.

До рефакторинга ввода-вывода, выполненного `condret`, команда `S` использовалась для управления так называемыми картами. В настоящее время команда `S` устарела, поскольку `iS` и `om` теперь достаточно.

Загрузчики прошивки микроконтроллеров, загрузчики двоичных файлов операционной системы обычно размещают секции по разным адресам в памяти. Radare интерпретирует работу загрузчика --- команды группы `iS`. Инструкции представлены в `iS?`. Для перечисления всех созданных секций используйте `iS` (или `iSj` для получения результата в формате json). Команда `iS=` покажет карту разделов в ascii-art.

С помощью подкоманды `om` можно создать новое отображение следующим образом:

```
om fd vaddr [size] [paddr] [rwx] [name]
```

Пример:

```
[0x0040100]> om 4 0x00000100 0x00400000 0x0001ae08 rwx test
```

Также можно использовать команду `om?` для просмотра сведений об отображаемых разделах:

```
[0x00401000]> om?
6 fd: 4 +0x0001ae08 0x00000100 - 0x004000ff rwx test
5 fd: 3 +0x00000000 0x00000000 - 0x0000055f r-- fmap.LOAD0
4 fd: 3 +0x00001000 0x00001000 - 0x000011e4 r-x fmap.LOAD1
3 fd: 3 +0x00002000 0x00002000 - 0x0000211f r-- fmap.LOAD2
2 fd: 3 +0x00002de8 0x00003de8 - 0x0000402f r-- fmap.LOAD3
1 fd: 4 +0x00000000 0x00004030 - 0x00004037 rw- mmap.LOAD3
```

В результате выполнения `om?` будет выведен перечень всех подкоманд. Чтобы перечислить все ранее определенные карты памяти, используйте `om` (или `omj` для получения результата в формате json, `om*` --- формат команд `r2`). Чтобы получить представление в ascii-art, используйте `om=`. Также можно удалить сопоставленный участок с помощью команды `om-mapid`.

Пример:

```
[0x00401000]> om-6
```

Отображение файлов

Подсистема ввода-вывода Radare позволяет сопоставлять содержимое файлов в пространство ввода-вывода, используемое для хранения загруженного двоичного файла. Программа radare2 способна открывать файлы и отображать их части в

любые места памяти, указывая такие атрибуты, как разрешения и имя. Это идеальный подход к воспроизведению среды, такой как core dump, сеанс отладки, а также загрузки и отображения в память всех библиотек, от которых зависит двоичный файл.

Команда O позволяет пользователю открыть файл, он отображается на смещение 0, если у него нет стандартного двоичного заголовка, карты памяти создаются в виртуальных адресах. Иногда надо перебазировать двоичный файл, или, загрузить и отобразить файл по другому адресу. При запуске r2 базовый адрес можно изменить флагом -B. Есть разница при открытии файлов с неизвестными заголовками, такими как bootloader-ы, их нужно отображать при помощи флага -m, или указав его в качестве аргумента команде O. Посмотрим инструкцию для команды:

```
[0x00000000]> o?
|Usage: o [com-] [file] ([offset])
| o                                     перечень открытых файлов
| o-1                                    закрыть файловый дескриптор 1
| o-*                                    закрыть все открытые файлы
| o--                                    закрыть все файлы, анализы, двоичные файлы, флаги, тоже, что и !r2 --
| o [file]                                открыть файл [file] в режиме "только для чтения"
| o+ [file]                               открыть файл в режиме "чтение-запись"
| o [file] 0x4000 rwx                     отобразить файл по адресу 0x4000
| oa[-] [A] [B] [filename]                указать архитектуру и размер регистров для заданного файла
| oq                                     перечислить все открытые файлы
| o*                                     перечислить открытые файлы в виде команд r2
| o. [len]                                открыть malloc://[len], копируя байты из текущего смещения
| o=                                     перечислить открытые файлы (ascii-art bars)
| ob[?] [lbdos] [...]                    перечислить открытые файлы "backed by" fd
| oc [file]                                открыть файл core-дампа, аналогично перезапуску r2
| of [file]                                открыть файл и отобразить его по адресу 0 в режиме "только для чтения"
| oi[-|idx]                               синоним для o, но используя индекс вместо fd
| oj[?]                                   перечислить открытые файлы в формате JSON
| oL                                     перечислить все зарегистрированные плагины ввода-вывода
| om[?]                                   создать, перечислить, удалить карты ввода-вывода
| on [file] 0x4000                      отобразить raw-файл по смещению 0x4000 (при этом r_bin не используется)
| oo[?]                                   переоткрыть текущий файл (kill+fork в отладчике)
| oo+                                    переоткрыть текущий файл в режиме "только для чтения"
| ood[r] [args]                           переоткрыть в режиме отладки (с аргументами)
| oo[bnm] [...]                          смотри инструкцию при помощи oo?
| op [fd]                                 приоритизировать заданный fd (смотри также ob)
| ox fd fdx                             поменять местами fd-descs и fdx и сохранить отображение
```

Покажем простой пример:

```
$ rabin2 -l /bin/ls
[Linked libraries]
libselinux.so.1
librt.so.1
libacl.so.1
libc.so.6
```

4 libraries

Сопоставление файла:

```
[0x000001190]> o /bin/zsh 0x499999
```

Список отображенных файлов:

```
[0x00000000]> o
- 6 /bin/ls @ 0x0 ; r
- 10 /lib/ld-linux.so.2 @ 0x100000000 ; r
- 14 /bin/zsh @ 0x499999 ; r
```

Показать шестнадцатеричные значения из /bin/zsh:

```
[0x00000000]> px @ 0x499999
```

Отмена отображения файлов с помощью команды O-. Укажите дескриптор файла в качестве аргумента:

```
[0x00000000]> o-14
```

Также можно просмотреть таблицу ascii со списком открытых файлов:

```
[0x00000000]> ob=
```

Режимы отображения данных

Одной из ключевых особенностей radare2 является отображение информации, представляя ее в разных форматах. Цель состоит в том, чтобы предложить выбор вариантов отображения для наилучшей интерпретации двоичных данных.

Двоичные данные представляют целые числа (int), shorts, longs, floats, timestamps, шестнадцатеричных строк или более сложных форматов, таких как структуры С, дизассемблированием, декомпиляцией, результатами внешних обработок и др.

Ниже приведен список доступных режимов отображения, перечисленных в p? (print):

```
[0x000005310]> p?
[Usage: p[=68abcdDfiImrstuxz] [arg|len] [@addr]
| p[b|B|xb] [len] ([S])    вывести двоичный дамп N-бит, пропустив S байт
| p[iI][df] [len]          вывести N оп-кодов/байт (f=func) (смотрите ri? и pdi)
| p[kK] [len]              распечатать ключ в виде random-art (K - использовать мозаику)
| p-[?][jh] [mode]         bar|json|гистограммные-блоки (режим: e?search.in)
| p2 [len]                 представить плитками 8x8, 2bpp (2 бита на пиксель)
| p3 [file]                Напечатать стереограмму (3D)
| p6[de] [len]             кодирование/декодирование base64
| p8[?][j] [len]           список 8bit-овых шестнадцатеричных байтов
| p=[?][бер] [N] [L] [b]   построить диаграмму энтропии/печатных символов/символов
| pa[edD] [arg]            ра:ассоциирование ра[dD]:дизассемблирование или рае: ESIL-инструкции из шестнадцатеричных
| pA[n_ops]                показать n_ops-адрес и тип
| pb[?] [n]                вывести в битовом представлении N бит
| pB[?] [n]                вывести в битовом представлении N байт
| pc[?][p] [len]           представить в формате С (или python)
| pC[aAcdDxw] [rows]      показать дизассемблирование в виде колонок (смотрите hex.cols и pdi)
| pd[?] [sz] [a] [b]       дизассемблировать N оп-кодов (pd) или N байт (pD)
| pf[?][.nam] [fmt]        распечатать отформатированные данные (pf.name, pf.name $<expr>)
| pF[?][apx]               распечатать asn1, pkcs7 или x509
| pg[?][x y w h] [cmd]    создать новый визуальный гаджет или распечатать его (смотри инструкцию в pg?)
| ph[?][=hash] ([len])   вычислить хэш для блока
| pj[?] [len]              распечатать в виде JSON с отступами
| pm[?] [magic]           распечатать данные libmagic (смотрите rm? и /m?)
| po[?] hex                распечатать операцию, применяемую к блоку (смотрите ro?)
| pp[?][sz] [len]          распечатать шаблоны, инструкция в pr?
| pq[?][is] [len]          распечатать QR-код для первых N байт
| pr[?][glx] [len]         распечатать N байт в raw-формате (в виде строк или шестнадцатеричных кодов, 'g'unzip')
| ps[?][pwz] [len]          распечатать строки в форматах pascal/wide/zero-terminated
| pt[?][dn] [len]           распечатать различные timestampы
| pu[?][w] [len]            распечатать N байт в кодировке url (w=wide)
| pv[?][jh] [mode]          показать переменную/указатель/значение в памяти
| pwd                      показать текущий рабочий директорий
| px[?][owq] [len]          вывести дамп N шестнадцатеричных байт (o=octal, w=32bit, q=64bit)
| pz[?] [len]               распечатать вид zoom (инструкция в pz?)
[0x000005310]>
```

Совет: при использовании вывода json можно добавлять ~{ } к команде, которая оформит красивые отступы:

```
[0x00000000]> oj
[{"raised":false,"fd":563280,"uri":"malloc://512","from":0,"writable":true,"size":512,"overlaps":false}]
[0x00000000]> oj~{{
  [
    {
      "raised": false,
      "fd": 563280,
      "uri": "malloc://512",
      "from": 0,
      "writable": true,
      "size": 512,
      "overlaps": false
    }
  ]
}}
```

Для получения дополнительной информации о функциональных возможностях ~ смотрите инструкцию в ?@?, а также главу Формат команды в этой книге.

Шестнадцатеричный вид

px дает удобный вывод, показывающий 16 пар чисел в строке со смещениями и raw-представлении:

```
[0x00404888]> px
- offset - 0 1 2 3 4 5 6 7 8 9 A B C D E F 0 1 2 3 4 5 6 7 8 9 ABCDEF
0x00404888 31ed 4989 d15e 4889 e248 83e4 f050 5449 1.I..^H..H...PTI
0x00404898 c7c0 4024 4100 48c7 c1b0 2341 0048 c7c7 ..@$A.H...#A.H...
0x004048a8 d028 4000 e83f dcff fff4 6690 662e 0f1f .(@...?....f.f...
```

Рис. 5: hexprint

Шестнадцатеричный дамп слов (32 бита)

```
[0x00404888]> pxw
0x00404888 0x8949ed31 0x89485ed1 0xe48348e2 0x495450f0 1.I..^H..H...PTI
0x00404898 0x2440c0c7 0xc7480041 0x4123b0c1 0xc7c74800 ..@$.H...#A.H..
0x004048a8 0x004028d0 0xfffdc3fe8 0x9066f4ff 0x1f0f2e66 .(@...?....f.f...
[0x00404888]> e cfg.bigendian
false
[0x00404888]> e cfg.bigendian = true
[0x00404888]> pxw
0x00404888 0x31ed4989 0xd15e4889 0xe24883e4 0xf0505449 1.I..^H..H...PTI
0x00404898 0xc7c04024 0x410048c7 0xc1b02341 0x0048c7c7 ..@$.H...#A.H..
0x004048a8 0xd0284000 0xe83fdcff 0xffff46690 0x662e0f1f .(@...?....f.f...

```

Рис. 6: wordprint

8-битовый шестнадцатеричный набор байтов

```
[0x00404888]> p8 16
31ed4989d15e4889e24883e4f0505449
```

```
[0x08049A80]> pxq
0x00001390 0x65625f6b63617473 0x646e6962006e6967 stack_begin.bind
0x000013a0 0x616d6f6474786574 0x7469727766006e69 textdomain.fwrite
0x000013b0 0x6b636f6c6e755f65 0x6d63727473006465 e_unlocked.strcm
...

```

Рис. 7: pxq

Шестнадцатеричный дамп из Quad-слов (64 бита)

Форматы даты/времени

В настоящее время поддерживаются следующие режимы вывода timestamp-ов:

```
[0x00404888]> pt?
|Usage: pt [dn] print timestamps
| pt. распечатать текущее время
| pt распечатать время UNIX (32 бит `cfg.bigendian`) с 1 января 1970
| ptd распечатать время DOS (32 бит `cfg.bigendian`) с 1 января 1980
| pth распечатать время HFS (32 бит `cfg.bigendian`) с 1 января 1904
| ptn распечатать время NTFS (64 бит `cfg.bigendian`) с 1 января 1601
```

Например, можно представить текущий буфер в виде временных меток во времени NTFS:

```
[0x08048000]> e cfg.bigendian = false
[0x08048000]> pt 4
29:04:32948 23:12:36 +0000
[0x08048000]> e cfg.bigendian = true
[0x08048000]> pt 4
20:05:13001 09:29:21 +0000
```

Как видите, порядок байтов влияет на результат. После того, timestamp напечатан, можно профильтировать grep-ом выходные данные, например, по номеру года:

```
[0x08048000]> pt ~1974 | wc -l
15
[0x08048000]> pt ~2022
27:04:2022 16:15:43 +0000
```

Формат даты по умолчанию можно настроить с помощью переменной `cfg.datefmt`. Правила форматирования следуют хорошо известному формату strftime(3). Покажем справочную страницу, вот наиболее важное:

%a	Сокращенное название дня недели в соответствии с текущим языковым стандартом.
%A	Полное название дня недели в соответствии с текущим языковым стандартом.
%d	День месяца в виде десятичного числа (диапазон от 01 до 31).
%D	Эквивалентно %m/%d/%y. (Yecch – только для американцев).
%H	Час в виде десятичного числа с использованием 24-часового формата (диапазон от 00 до 23).
%I	Час в виде десятичного числа с использованием 12-часового формата (диапазон от 01 до 12).
%m	Месяц в виде десятичного числа (диапазон от 01 до 12).
%M	Минута в виде десятичного числа (диапазон от 00 до 59).
%p	Либо "AM", либо "PM" в соответствии с заданным значением времени.

```
%s Количество секунд со временем Эпохи, 1970-01-01 00:00:00 +0000 (UTC). (T3)
%S Секунда в виде десятичного числа (диапазон от 00 до 60). (Диапазон составляет до 60 секунд, что позволяет время...
%T Время в 24-часовой нотации (%H:%M:%S). (SU)
%y Год в виде десятичного числа без века (диапазон от 00 до 99).
%Y Год в виде десятичного числа, включая век.
%z Числовой часовой пояс +hhmm или -hhmm (то есть часовое и минутное смещение от UTC). (SU)
%Z Название или аббревиатура часового пояса.
```

Основные типы

Существуют режимы печати, доступные для всех базовых типов. Если вас интересует более сложная структура, введите `pf???` для символов формата и `pf???` для примеров:

```
[0x00499999]> pf??
|pf: pf[.k[=v]][[v]]|[n]|[@|cnt][fmt] [a0 a1 ...]
| Format:
| b      байт (unsigned)
| B      интерпретировать битовое enum-поле (смотрите t?)
| c      char (байт со знаком)
| C      байт в десятичном виде
| d      значение 0xHEX (4 байта) (смотрите 'i' и 'x')
| D      дезассемблирование одного оп-кода
| e      временно сменить порядок байтов
| E      представить в виде имени из enum (смотрите t?)
| f      значение float (4 байта)
| F      значение double (8 байт)
| i      целое значение со знаком (4 байта) (смотрите 'd' и 'x')
| n      следующий символ задает размер значения со знаком (1, 2, 4 или 8 байт)
| N      следующий символ задает размер значения без знака (1, 2, 4 или 8 байт)
| o      восьмеричное значение (4 байта)
| p      указатель (2, 4 или 8 байт)
| q      quadword (8 байт)
| r      регистр CPU `pf r (eax)plop`
| s      32-битовый указатель на строку (4 байта)
| S      64-битовый указатель на строку (8 байт)
| t      UNIX timestamp (4 байта)
| T      показать первых десять (Ten) байт буфера
| u      uleb128 (variable length)
| w      слово (2 байта, unsigned short в шестнадцатеричном виде)
| x      значение в 0xHEX и флаг (fd @ addr) (смотрите 'd' и 'i')
| X      отформатированный шестнадцатеричный дамп
| z      строка, заканчивающаяся 0
| Z      wide-строка, заканчивающаяся 0
| ?      структура данных 'pf ? (имя_структурьи)иня_экземпляра'
| *      следующий символ - указатель (учитывает asm.bits)
| +      включить/выключить вывод флага для каждого смещения
| :      пропустить 4 байта
| .      пропустить 1 байт
| ;      отступить назад на 4 байт
| ,      отступить назад на 1 байт
```

Используйте тройной вопросительный знак `pf???` для просмотра примеров с использованием формата печати строк (`printf`).

```
[0x00499999]> pf???
|pf: pf[.k[=v]][[v]]|[n]|[@|cnt][fmt] [a0 a1 ...]
| Примеры:
| pf 3xi foo bar                         массив из трех структур, в каждой три поля: 'foo' в шестнадцатеричном виде
| pf B (BitFldType)arg_name`              тип битовых полей
| pf E (EnumType)arg_name`                тип enum
| pf.obj xxzd prev next size name        определить формат obj как xxzd
| pf obj=xxzd prev next size name        то же, что и выше
| pf *z*i*w nb name blob                 распечатать указатели с заданными именами
| pf iwq foo bar troll                   распечатать в формате iwq, foo, bar, troll - соответствующие поля строки
| pf 0iwq foo bar troll                  то же, что выше, но представляющих юнион (все поля на смещении 0)
| pf.plop ? (troll)mystruct             использовать структуру troll, определенную ранее
| pfj.plop @ 0x14                        применить формат к заданному смещению
| pf 10xiz pointer length string        распечатать 10 ячеек массива структур формата xiz с соответствующими полями
| pf 5sqw string quad word              распечатать массив элементов формата sqw с соответствующими полями
| pf {integer}? (bifc)                  распечатать integer раз формат (bifc)
| pf [4]w[7]i                            распечатать массив из четырех слов и за ним массив из семи целых чисел
| pf ic...?i foo bar "(pf xw yo foo)troll" yo  распечатать вложенные непоименнованные структуры
| pf ;..x                               распечатать значение, расположенные через шесть байт от текущего смещения
| pf [10]z[3]i[10]Zb                     распечатать строку фиксированного размера, widechar и переменную
| pfj +F @ 0x14                          распечатать содержимое по заданному смещению с флагом
| pf n2                                распечатать знаковое короткое целое (2 байта) значение. Используйте M
| pf [2]? (plop)structname @ 0          Песатает массив структур
| pf eqew bigWord beef                  переключить порядок байтов, печать с использованием полей
| pf.foo rr (eax)reg1 (eip)reg2         создать объект ссылающийся на значение в регистре
```

| pf tt troll plop распечатать timestamp-ы, используя поля troll и plop

Ниже приведены примеры:

```
[0x4A13B8C0]> pf i  
0x00404888 = 837634441
```

```
[0x4A13B8C0]> pf  
0x00404888 = 837634432.000000
```

Высокоуровневые представления в синтаксисе языков программирования

Допустимые форматы печати, использующие высокоуровневый синтаксис:

- pc C
 - pc* печать команд `wx` r2
 - pch C half-words (2 байта)
 - pcw слова C (4 байта)
 - pcd C dwords (8 байт)
 - pci массив байт C с инструкциями
 - pca GAS .byte blob
 - pcA .bytes с инструкциями в комментариях
 - pcs строка
 - pcS скрипт оболочки операционной системы, воссоздающий бинарные данные
 - pcj json
 - pcJ javascript
 - pco Objective-C
 - pcp python
 - pck kotlin
 - pcr rust
 - pcv JaVa
 - pcV V (vlang.io)
 - pcy yara
 - pcz Swift

Если нужно создать файл .c, содержащий BLOB, используйте команду `rc`. Размер по умолчанию такой же, как и в многих других командах: размер блока можно изменить с помощью команды `b`. Также можно просто временно переопределить этот размер блока, выразив его в качестве аргумента.

```
[0xB7F8E810]> pc 32
#define _BUFFER_SIZE 32
unsigned char buffer[_BUFFER_SIZE] = {
0x89, 0xe0, 0xe8, 0x49, 0x02, 0x00, 0x00, 0x89, 0xc7, 0xe8, 0xe2, 0xff, 0xff, 0xff, 0x81, 0xc3, 0xd6, 0xa7, 0x01, 0x
```

Структура `cstring` может использоваться во многих языках программирования, а не только в C.

```
[0x7fcfd6a891630]> pcs  
"\x48\x89\xe7\xe8\x68\x39\x00\x00\x49\x89\xc4\x8b\x05\xef\x16\x22\x00\x5a\x48\x8d\x24\xc4\x29\xc2\x52\x48\x89\xd6\x4
```

Строки

С анализа строк, вероятно, начинается каждый реверс-инженеринг программы, поскольку на них обычно ссылается код тела функции: конкретные операторы, отладочная информация и другие сообщения. Поэтому radare поддерживает различные строковые форматы:

```
[0x00000000] > ps?
| Usage: ps[bijqpsuwWxz+] [N]  Печать строки
| ps      печать строки
| ps+[j]  печать строки libc++ std::string (same-endian, ascii, zero-terminated)
| psb    печать строк, найденных в текущем блоке
| psi    печать строки внутри текущего смещения
| psj    печать строки в формате JSON
| psp[j]  печать pascal-строки
| psq    синоним для rqs
| pss    печать строки на экран, вставляя переносы
| psu[zj]  печать utf16-юникода (json)
| psw[j]  печать wide-строку 16-бит
| psW[j]  печать wide-строку 32-бит
| psx    показать строку при помощи escape-последовательностей
| psz[j]  печать строки, заканчивающихся нулем
```

Большинство строк заканчиваются нулем. Ниже приведен пример использования отладчика для продолжения выполнения программы до тех пор, пока не выполнится системный вызов `open'. Как только процессом приостановлен, получаем

аргументы, переданные в системный вызов, на нужный нам указывает %ebx. В случае вызова 'open' это строка с нулевым окончанием, проверим с помощью `psz`.

```
[0x4A13B8C0]> dcs open
0x4a14fc24 syscall(5) open ( 0x4a151c91 0x00000000 0x00000000 ) = 0xffffffffda
[0x4A13B8C0]> dr
  eax 0xfffffffda    esi 0xfffffffff    eip    0x4a14fc24
  ebx 0x4a151c91    edi 0x4a151be1    oeax   0x00000005
  ecx 0x00000000    esp 0xbfbedb1c    eflags 0x200246
  edx 0x00000000    ebp 0xbfbbedbb0    cPaZstIdor0 (PZI)
[0x4A13B8C0]>
[0x4A13B8C0]> psz @ 0x4a151c91
/etc/ld.so.cache
```

Печать содержимого памяти

Печать различных упакованных типов данных делается с помощью команды `pf`:

```
[0xB7F08810]> pf xxS @ rsp
0x7fff0d29da30 = 0x00000001
0x7fff0d29da34 = 0x00000000
0x7fff0d29da38 = 0x7fff0d29da38 -> 0x0d29f7ee /bin/ls
```

Форматы можно использовать для просмотра аргументов, передаваемых функции. Передайте строку, описывающую формат в аргумент `pf` и задайте смещение при помощи `@`. Также можно определять массивы структур. Для этого добавьте к строке форматирования префикс с числовым значением. Можно также определять имена для каждого поля структуры, добавив их в виде списка аргументов, разделенных пробелами.

```
[0x4A13B8C0]> pf 2*xw pointer type @ esp
0x00404888 [0] {
    pointer :
(*0xffffffff8949ed31)      type : 0x00404888 = 0x8949ed31
    0x00404890 = 0x48e2
}
0x00404892 [1] {
(*0x50f0e483)      pointer : 0x00404892 = 0x50f0e483
    type : 0x0040489a = 0x2440
}
```

Практический пример использования `pf` в двоичном файле плагина GStreamer:

```
$ radare2 /usr/lib/gstreamer-1.0/libgstflv.so
[0x00006020]> aa; pdf @ sym.gst_plugin_flv_get_desc
[x] Analyze all flags starting with sym. and entry0 (aa)
sym.gst_plugin_flv_get_desc ();
[...]
 0x00013830      488d0549db0000  lea rax, section..data.rel.ro ; 0x21380
 0x00013837      c3            ret
[0x00006020]> s section..data.rel.ro
[0x00021380]> pf ii*z*zp*z*z*z*z*zz major minor name desc init version license source package origin release_datet
major : 0x00021380 = 1
minor : 0x00021384 = 18
name : (*0x19cf2)0x00021388 = "flv"
desc : (*0x1b358)0x00021390 = "FLV muxing and demuxing plugin"
init : 0x00021398 = (qword)0x0000000000013460
version : (*0x19cae)0x000213a0 = "1.18.2"
license : (*0x19ce1)0x000213a8 = "LGPL"
source : (*0x19cd0)0x000213b0 = "gst-plugins-good"
package : (*0x1b378)0x000213b8 = "GStreamer Good Plugins (Arch Linux)"
origin : (*0x19cb5)0x000213c0 = "https://www.archlinux.org/"
release_datetime : (*0x19cf6)0x000213c8 = "2020-12-06"
```

Дизассемблирование

Команда `pd` используется для дизассемблирования кода. Она принимает числовое значение, указывающее, сколько инструкций должно быть разобрано. Команда `PD` аналогична, но вместо нескольких инструкций она декомпилирует заданное количество байтов.

- `d` : дизассемблирование N оп-кодов, количество оп-кодов,
- `D` : дизассемблировать N байт, учитывая asm.arch

```
[0x00404888]> pd 1
;-- entry0:
0x00404888 31ed      xor ebp, ebp
```

Выбор целевой архитектуры

Целевая архитектура дизассемблера определяется переменной `asm.arch`. Можно использовать `e asm.arch=??` для перечисления всех доступных архитектур.

```
[0x000005310]> e asm.arch=??  
_dAe _8_16      6502      LGPL3  6502/NES/C64/Tamagotchi/T-1000 CPU  
_dAe _8          8051      PD    8051 Intel CPU  
_dA_  _16_32     arc       GPL3  Argonaut RISC Core  
a___ _16_32_64   arm.as    LGPL3  as ARM Assembler (use ARM_AS environment)  
adAe _16_32_64   arm       BSD   Capstone ARM disassembler  
_dA_  _16_32_64   arm.gnu   GPL3  Acorn RISC Machine CPU  
_d__  _16_32     arm.winedbg LGPL2  WineDBG's ARM disassembler  
adAe _8_16       avr      GPL   AVR Atmel  
adAe _16_32_64   bf        LGPL3 Brainfuck  
_dA_  _32       chip8     LGPL3 Chip8 disassembler  
_dA_  _16       cr16      LGPL3 cr16 disassembly plugin  
_dA_  _32       cris      GPL3  Axis Communications 32-bit embedded processor  
adA_  _32_64     dalvik    LGPL3 AndroidVM Dalvik  
ad__  _16       dcpu16    PD    Mojang's DCPU-16  
_dA_  _32_64     ebc       LGPL3 EFI Bytecode  
adAe _16       gb        LGPL3 GameBoy(TM) (z80-like)  
_dAe _16       h8300     LGPL3 H8/300 disassembly plugin  
_dAe _32       hexagon   LGPL3 Qualcomm Hexagon (QDSP6) V6  
_d__  _32       hppa      GPL3  HP PA-RISC  
_dAe _0         i4004     LGPL3 Intel 4004 microprocessor  
_dA_  _8         i8080     BSD   Intel 8080 CPU  
adA_  _32       java      Apache Java bytecode  
_d__  _32       lanai     GPL3  LANAI  
...
```

Настройка дизассемблера

Существует несколько вариантов настроек вывода дизассемблера. Все настройки описаны в `e? asm.`

```
[0x000005310]> e? asm.  
asm.anal:      Анализировать код и ссылки при дизассемблировании (смотрите anal.strings)  
asm.arch:      Задать архитектуру, используемую при ассемблировании  
asm.assembler: Установить имя плагина для использования во время ассемблирования  
asm.bblines:   Добавить пустую строку между базовыми блоками  
asm.bits:      Размер слова в битах при ассемблировании  
asm.bytes:    Показать байтовое представление для каждой инструкции  
asm.bytespace: Разделять шестнадцатеричное представление пробелами  
asm.calls:    Показать информацию, релевантную вызываемой функции в виде комментариев в дизассемблировании  
asm.capitalize: Использовать CamelCase при дизассемблировании  
asm.cmt.col:  Колонка выравнивания коментариев  
asm.cmt.flgrefs: Показать флаг-комментарий, ассоциированный с ветвлением  
asm.cmt.fold: Скрыть комментарии, переключается при помощи Vz  
...
```

В настоящее время существует 136 переменных конфигурации `asm.`, не будем перечислять их все.

Синтаксис дизассемблера

Переменная `asm.syntax` используется для изменения диалекта ассемблера, используемого механизмом дизассемблера. Чтобы переключиться между представлениями Intel и AT&T:

```
e asm.syntax = intel  
e asm.syntax = att
```

Также можно попробовать `asm.pseudo`, являющийся экспериментальным представлением псевдокода, и `asm.esil`, выводящий ESIL ('Evaluable Strings Intermediate Language'). Цель ESIL состоит в том, чтобы иметь удобное представление семантики оп-кодов. Такие варианты представления данных полезны для интерпретации отдельных инструкций.

Флаги

Флаги концептуально похожи на закладки. Они связывают имя с заданным смещением в файле. Флаги группируются в «пространства флагов». Пространство флагов --- это пространство имен для флагов, объединяющее флаги со схожими характеристиками или типом. Примеры флаговых пространств: секции (sections), регистры (registers), символы (symbols).

Создание флага:

```
[0x4A13B8C0]> f flag_name @ offset
```

Удаление флага --- добавление символа - к команде. Большинство команд принимают префиксный аргумент - как указание на удаление чего-либо.

```
[0x4A13B8C0]> f-flag_name
```

Для переключения между пространствами флагов или созданием новых используйте команду `fs`:

```
[0x00005310]> fs?
[Usage: fs [*] [+][flagspace|addr] # Управление пространствами флагов
| fs      перечислить пространства флагов
| fs*     перечислить пространства флагов в виде команд r2
| fsj    перечислить пространства флагов в виде JSON-а
| fs *   выбрать все пространства флагов
| fs flagspace  выбрать пространство флагов или создать его, если не существует
| fs-flagspace удалить пространство флагов
| fs-*   удалить все пространства флагов
| fs+foo запомнить в стек предыдущее пространство флагов и установить новое
| fs-   установить предыдущее пространство флагов из стека
| fs-.  удаление текущего пространства флагов
| fsq   перечислить пространства флагов в "тихом" режиме
| fsm [addr] переместить флаги на заданном адресе в текущее пространство флагов
| fss   показать стек пространств флагов
| fss*  показать стек пространств флагов в виде команд r2
| fssj   показать стек пространств флагов в виде JSON-а
| fsr newname удалить выбранное пространство флагов
[0x00005310]> fs
0 439 * strings
1 17 * symbols
2 54 * sections
3 20 * segments
4 115 * relocs
5 109 * imports
[0x00005310]>
```

Вот несколько примеров команд:

```
[0x4A13B8C0]> fs symbols ; выбрать флагов в пространстве флагов "symbols"
[0x4A13B8C0]> f           ; перечислить только флаги в пространстве "symbols"
[0x4A13B8C0]> fs *        ; выбрать все пространства имен
[0x4A13B8C0]> f myflag   ; создать новый флаг с именем 'myflag'
[0x4A13B8C0]> f-myflag   ; удалить флаг, называющийся 'myflag'
```

Переименовывание флагов с помощью `fr`.

Локальные флаги

Каждое имя флага должно быть уникальным по понятным причинам. Часто надо задавать флаги, например внутри функций, с простыми и обычными именами, такими как `loop` или `return`. Здесь можно использовать так называемые «локальные» флаги, которые привязаны к конкретным функциям. Их можно добавить с помощью команды `f .`:

```
[0x00003a04]> pd 10
| 0x00003a04 48c705c9cc21. mov qword [0x002206d8], 0xffffffffffffffff ;
[0x2206d8:8]=0
| 0x00003a0f c60522cc2100. mov byte [0x00220638], 0 ; [0x220638:1]=0
| 0x00003a16 83f802 cmp eax, 2
.-< 0x00003a19 0f84880d0000 je 0x47a7
| 0x00003a1f 83f803 cmp eax, 3
.—< 0x00003a22 740e je 0x3a32
|| 0x00003a24 83e801 sub eax, 1
.—< 0x00003a27 0f84ed080000 je 0x431a
||| 0x00003a2d e8fef8ffff call sym.imp.abort ; void abort(void)
||| ; CODE XREF from main (0x3a22)
||| |> 0x00003a32 be07000000 mov esi, 7
[0x00003a04]> f. localflag @ 0x3a32
[0x00003a04]> f.
0x00003a32 localflag [main + 210]
[0x00003a04]> pd 10
| 0x00003a04 48c705c9cc21. mov qword [0x002206d8], 0xffffffffffffffff ;
[0x2206d8:8]=0
| 0x00003a0f c60522cc2100. mov byte [0x00220638], 0 ; [0x220638:1]=0
| 0x00003a16 83f802 cmp eax, 2
.-< 0x00003a19 0f84880d0000 je 0x47a7
| 0x00003a1f 83f803 cmp eax, 3
.—< 0x00003a22 740e je 0x3a32 ; main.localflag
|| 0x00003a24 83e801 sub eax, 1
.—< 0x00003a27 0f84ed080000 je 0x431a
```

```

|||| 0x00003a2d      e8fef8ffff    call sym.imp.abort          ; void abort(void)
|||| ; CODE XREF from main (0x3a22)
||`-> .localflag:
|||| ; CODE XREF from main (0x3a22)
||`-> 0x00003a32      be07000000    mov esi, 7
[0x00003a04]>

```

Зоны флагов

Radare2 реализует механизм зоны флагов (flagzone), позволяющий помечать различные смещения на полосе прокрутки, чтобы упростить навигацию по большим двоичным файлам. Установка зона флага по текущему смещению:

```
[0x00003a04]> fz flag-zone-name
```

Установите `scr.scrollbar=1` и перейдите в визуальный режим, зона флага появится на полосе прокрутки с правой стороны окна.

Посмотрите инструкцию - `fz?`.

Сохранение данных

Radare может манипулировать загруженным двоичным файлом. Можно изменять размер файла, перемещать и копировать/вставлять байты, вставлять новые байты (перемещая данные в конец блока или файла) или просто перезаписывать байты. Новые данные могут быть предоставлены в виде строки специального вида, инструкций ассемблера, или они могут быть считаны из другого файла.

Изменения размера файла --- команда `r`. Она принимает числовой аргумент. Положительное значение задает новый размер файла. Отрицательное значение приведет к усечению файла до текущей позиции поиска за вычетом N байтов.

```
r 1024      ; установить размер файла равным 1024 байт
r -10 @ 33   ; вырезать десять байт по смещению 33
```

Сохранение (запись в файл) байтов --- команда `w`. Она поддерживает несколько форматов входных данных, таких как результаты ассемблирования, дружественные к порядку байтов DWORD, файлы, шестнадцатеричные файлы, широкие строки:

```
[0x00404888]> w?
Usage: w[x] [str] [<file>] [<<EOF>>] [@addr]
| w[1248][+-][n]      увеличить/уменьшить на 1 байт, слово..
| w foobar           запись строки 'foobar'
| w0 [len]            запись 'len' байт значениями 0x00
| w6[de] base64/hex  запись base64-декодированной или -кодированной строки
| wa[?] push ebp     запись оп-кодов, разделенных символом ';' (окружите команду '''-ками)
| waf f.asm           ассоциировать файл и результат записать
| waF f.asm           ассоциировать файл и результат записать, затем показать ('wx') оп-коды в шестнадцатеричном виду
| wao[?] op           изменить оп-код (изменить условия переходов (jmp), вставить пор и др.)
| wA[?] r 0            изменить/заменить оп-код по текущему смещению (смотрите wA?)
| wb 010203           заполнить текущий блок шестнадцатеричными данными циклически
| wB[-]0xVALUE        установить или сбросить биты заданным значением
| wc                 перечислить все изменения, относящиеся к записи
| wc[?][jir+-*?]      показать кэш команд undo/commit/reset/list (плагин io.cache)
| wd [off] [n]         сдублировать N байт (memcp), расположенных в текущем смещении, также смотрите y?
| we[?] [nNsX] [arg]  выполнять операции "запись" в режиме вставки (extend write)
| wf[fs] -|file       записать содержимое файла по текущему смещению
| wh r2               команды операционной системы whereis/which
| wm f0ff             установить двоичную маску в виде шестнадцатеричного кода для использования в последующих командах
| wo[?] hex           запись в блок при помощи операций. Инструкции - 'wo?' fmi
| wp[?] -|file        применить заплатку radare из файла. Инструкции - wp? fmi
| wr 10               запись десяти случайных байтов
| ws pstring          запись одного байта длины и затем строки
| wt[f][?] file [sz]  запись в файл (с текущего смещения, размером blocksize или sz байт)
| wts host:port [sz]  отправить данные на удаленную рабочую станцию host:port по tcp://
| ww foobar           запись wide-строки 'f\x00o\x00b\x00a\x00r\x00'
| wx[?][fs] 9090       запись двух intel-овских оп-кодов (с wxfile или wxseek)
| wv[?] eip+34         запись 32-64-тowych значений с учетом cfg.big endian
| wz string           запись строки, заканчивающейся нулем (аналогично w + \x00)
```

Примеры:

```
[0x00000000]> wx 123456 @ 0x8048300
[0x00000000]> wv 0x8048123 @ 0x8049100
[0x00000000]> wa jmp 0x8048320
```

Сохранение поверх существующих данных

Команда **wo** (write over) имеет множество подкоманд, каждая комбинирует существующие данные с новыми данными. Команда применяется к текущему блоку. Поддерживаемые операторы включают XOR, ADD, SUB, ...

```
[0x4A13B8C0]> wo?
|Usage: wo[asmdxoArl24] [hexpairs] @ addr[:bsize]
|Примеры:
|  wox 0x90 ; выполнить операцию xor 0x90 с текущим блоком
|  wox 90   ; выполнить операцию xor 0x90 с текущим блоком
|  wox 0x0203 ; выполнить операцию xor 0203 с текущим блоком
|  woa 02 03 ; добавить [0203][0203][...] к текущему блоку
|  woe 02 03 ; создать последовательность от 2 до 255 с шагом 3
|Поддерживаемые операции:
|  wow == запись циклического значения (псевдоним для 'wb')
|  woa += сложение
|  wos -= вычитание
|  wom *= умножение
|  wod /= деление
|  wox ^= xor (исключающее или)
|  woo |= or (или)
|  woA &= and (и)
|  woR сгенерировать случайные байты (псевдоним для 'wr $b')
|  wor >>= сдвиг вправо
|  wol <<= сдвиг влево
|  wo2 2= 2-байтовый обмен
|  wo4 4= 4-байтовый обмен
```

Возможна реализация алгоритмов шифрования с использованием команд ядра radare и **wo**. Пример сеанса, выполняющего **xor(90) + add(01, 02)**:

```
[0x7fc6a891630]> px
- offset -      0 1 2 3 4 5 6 7 8 9 A B C D E F
0x7fc6a891630 4889 e7e8 6839 0000 4989 c48b 05ef 1622
0x7fc6a891640 005a 488d 24c4 29c2 5248 89d6 4989 e548
0x7fc6a891650 83e4 f048 8b3d 061a 2200 498d 4cd5 1049
0x7fc6a891660 8d55 0831 ede8 06e2 0000 488d 15cf e600
[0x7fc6a891630]> wox 90
[0x7fc6a891630]> px
- offset -      0 1 2 3 4 5 6 7 8 9 A B C D E F
0x7fc6a891630 d819 7778 d919 541b 90ca d81d c2d8 1946
0x7fc6a891640 1374 60d8 b290 d91d 1dc5 98a1 9090 d81d
0x7fc6a891650 90dc 197c 9f8f 1490 d81d 95d9 9f8f 1490
0x7fc6a891660 13d7 9491 9f8f 1490 13ff 9491 9f8f 1490
[0x7fc6a891630]> woa 01 02
[0x7fc6a891630]> px
- offset -      0 1 2 3 4 5 6 7 8 9 A B C D E F
0x7fc6a891630 d91b 787a 91cc d91f 1476 61da 1ec7 99a3
0x7fc6a891640 91de 1a7e d91f 96db 14d9 9593 1401 9593
0x7fc6a891650 c4da 1a6d e89a d959 9192 9159 1cb1 d959
0x7fc6a891660 9192 79cb 81da 1652 81da 1456 a252 7c77
```

Масштабирование

Масштабирование --- это режим отображения информации, позволяющий получить глобальное представление всего файла или карты памяти на одном экране. В этом режиме каждый байт картинки представляет блок **file_size/block_size** байт файла. Используйте команду **pz** или просто **Z** в визуальном режиме, чтобы переключить режим масштабирования.

Для быстрой прокрутки в режиме масштабирования можно использовать курсор. Повторное нажатие **Z** приведет к увеличению масштаба в текущем положении курсора.

```
[0x004048c5]> pz?
|Usage: pz [len] печать масштабированных блоков размером filesize/N
| e zoom.maxsz  максимальный размер блока
| e zoom.from  начальный адрес
| e zoom.to    конечный адрес
| e zoom.byte  указать, как вычислять каждый байт
| pzp          количество печатных символов
| pzf          подсчитать количество флагов в блоке
| pzs          строки в заданном диапазоне
| pz0          число байт равных '0'
| pzF          число байт равных 0xFF
| pze          вычислить энтропию и расширить на диапазон 0-255
| pzh          вывести начало (значение первых байтов); это режим по умолчанию
```

Рассмотрим несколько примеров:

```
[0x08049790]> e zoom.byte=h
[0x08049790]> pz // или по умолчанию pzh
```

```

0x0000000000 7f00 0000 e200 0000 146e 6f74 0300 0000
0x000000010 0000 0000 0068 2102 00ff 2024 e8f0 007a
0x000000020 8c00 18c2 ffff 0080 4421 41c4 1500 5dff
0x000000030 ff10 0018 0fc8 031a 000c 8484 e970 8648
0x000000040 d68b 3148 348b 03a0 8b0f c200 5d25 7074
0x000000050 7500 00e1 ffe8 58fe 4dc4 00e0 dbc8 b885

[0x08049790]> e zoom.byte=p
[0x08049790]> pz // или pzp
0x00000000 2f47 0609 070a 0917 1e9e a4bd 2a1b 2c27
0x00000010 322d 5671 8788 8182 5679 7568 82a2 7d89
0x00000020 8173 7f7b 727a 9588 a07b 5c7d 8daf 836d
0x00000030 b167 6192 a67d 8aa2 6246 856e 8c9b 999f
0x00000040 a774 96c3 b1a4 6c8e a07c 6a8f 8983 6a62
0x00000050 7d66 625f 7ea4 7ea6 b4b6 8b57 a19f 71a2

[0x08049790]> eval zoom.byte = flags
[0x08049790]> pz // или pzf
0x00406e65 48d0 80f9 360f 8745 ffff ffeb ae66 0f1f
0x00406e75 4400 0083 f801 0f85 3fff ffff 410f b600
0x00406e85 3c78 0f87 6301 0000 0fb6 c8ff 24cd 0026
0x00406e95 4100 660f 1f84 0000 0000 0084 c074 043c
0x00406ea5 3a75 18b8 0500 0000 83f8 060f 95c0 e9cd
0x00406eb5 feff ff0f 1f84 0000 0000 0041 8801 4983
0x00406ec5 c001 4983 c201 4983 c101 e9ec feff ff0f

[0x08049790]> e zoom.byte=F
[0x08049790]> p0 // или pzf
0x00000000 0000 0000 0000 0000 0000 0000 0000 0000
0x00000010 0000 2b5c 5757 3a14 331f 1b23 0315 1d18
0x00000020 222a 2330 2b31 2e2a 1714 200d 1512 383d
0x00000030 1e1a 181b 0a10 1a21 2a36 281e 1d1c 0e11
0x00000040 1b2a 2f22 2229 181e 231e 181c 1913 262b
0x00000050 2b30 4741 422f 382a 1e22 0f17 0f10 3913

```

Масштабирование можно ограничить диапазоном байтов, и не показывать всё пространство, --- переменные среды `zoom.from` и `zoom.to`:

```

[0x00003a04]> e? zoom.
zoom.byte: Функция обратного вызова для вычисления каждого байта (смотрите инструкцию в pz?)
zoom.from: Начальный адрес блока для масштабирования
zoom.in: Границы для масштабирования
zoom.maxsz: Максимальный размер блока для масштабирования
zoom.to: Конечный адрес для масштабирования
[0x00003a04]> e zoom.
zoom.byte = h
zoom.from = 0
zoom.in = io.map
zoom.maxsz = 512
zoom.to = 0

```

Сохранение/Вставка

Radare2 имеет свой буфер обмена для сохранения и загрузки оттуда частей памяти, взятых из текущего объекта ввода-вывода. Содержимым буфера обмена можно манипулировать при помощи команды `y` (`yank`).

Двумя основными операциями являются:

- Копирование (`yank`),
- Вставка (`paste`).

Операция `yank` загружает (вытаскивает) N байт, заданных аргументом, в буфер обмена. Затем используя команду `yy`, делаем вставку сохраненного ранее блока в файл. Можно загружать/вставлять (`yank/paste`) байты в визуальном режиме, выбирая их в режиме курсора (`Vc`), а затем используя привязки клавиш `u` и `Y`, являющиеся псевдонимами для команд `y` и `yy` интерфейса командной строки.

```

[0x00000000]> y?
Usage: y[ptxy] [len] [[@]addr] # Посмотрите wd? темсру то же самое, что и 'uf'.
| y!          открытие cfg.editor для редактирования содержания буфера обмена
| y 16 0x200   копировать 16 байт в буфер обмена с адреса 0x200
| y 16 @ 0x200 копировать 16 байт в буфер обмена с адреса 0x200
| y 16          копировать 16 байт в буфер обмена
| y          показать информацию о yank-буфере (srcoff len байт)
| y*          печатать в виде команд r2 того, что будет загружено из буфера
| yf 64 0x200 скопировать 64 байта из файла по смещению 0x200
| yfa file copy скопировать все байты из файла (открывает ввод-вывод на запись, w/)
| yfx 10203040 загрузить из шестнадцатеричного кода (то же, что и уwx)
| yj          показать в виде формата JSON содержимого из буфера обмена
| yr          печатать содержания буфера обмена

```

```

| yq      печать содержания буфера обмена в шестнадцатеричном виде
| ys      печать содержания буфера обмена в виде строки
| yt 64 0x200  копировать 64 байта с текущего смещения по адресу 0x200
| ytf file  выдать дамп содержания буфера обмена в заданный файл
| yw hello world загрузить из строки
| ywx 10203040 загрузить из шестнадцатеричного кода (то же, что и yfx)
| yx      выдать содержимого буфера обмена в шестнадцатеричном виде
| yy 0x3344  вставить в буфер обмена
| yz [len]  копировать строку, завершающуюся нулем, в буфер обмена, учитывая ограничения blocksize

```

Пример сессии:

```
[0x00000000]> s 0x100 ; установить смещение 0x100
[0x00000100]> u 100 ; вытащить в буфер обмена 100 байт с текущего смещения
[0x00000200]> s 0x200 ; установить смещение 0x200
[0x00000200]> uy ; вставить 100 байт из буфера обмена
```

Можно выполнить сохранение и вставку в одной строке, используя команду `yt` (yank-to). Вот её синтаксис:

```
[0x4A13B8C0]> x
  offset 0 1 2 3 4 5 6 7 8 9 A B 0123456789AB
0x4A13B8C0, 89e0 e839 0700 0089 c7e8 e2ff ...9.....
0x4A13B8CC, ffff 81c3 eea6 0100 8b83 08ff .....
0x4A13B8D8, ffff 5a8d 2484 29c2 ..Z.$.).
```



```
[0x4A13B8C0]> yt 8 0x4A13B8CC @ 0x4A13B8C0
```



```
[0x4A13B8C0]> x
  offset 0 1 2 3 4 5 6 7 8 9 A B 0123456789AB
0x4A13B8C0, 89e0 e839 0700 0089 c7e8 e2ff ...9.....
0x4A13B8CC, 89e0 e839 0700 0089 8b83 08ff ...9.....
0x4A13B8D8, ffff 5a8d 2484 29c2 ..Z.$.).
```

Сравнение байтов

В реверс-инжениринге часто решаются задачи поиска различий между двумя двоичными файлами, выявления измененных байтов, сравнения графов, представляющих результаты анализа кода, а также выполнение других операций сравнения. Сравнение реализуется при помощи программы `radiff2`:

```
$ radiff2 -h
```

Внутри `r2` функциональные возможности, предоставляемые `radiff2`, доступны при помощи набора команд `C`.

Команда `C` (сокращение от «compare») позволяет сравнивать массивы байтов из разных источников. Команда принимает ввод в нескольких форматах, а затем сравнивает его со значениями, найденными в текущем смещении.

```
[0x00404888]> c?
Usage: c[?dfx] [argument] # Сравнение
| c [string]    Сравнение обычной строки со строкой с escape-последовательностями
| c* [string]   То же, что выше, но печатая результат в виде команд r2
| c1 [addr]     Сравнить 8 бит в текущем смещении
| c2 [value]    Сравнить со словом, полученным в результате вычисления выражения
| c4 [value]    Сравнить с двойным словом, полученным в результате вычисления выражения
| c8 [value]    Сравнить с quadword, полученным в результате вычисления выражения
| cat [file]    Показать содержимое файла (смотрите pwd, ls)
| cc [at]       Сравнить две колонки шестнадцатеричных дампов, ограниченных размером блока
| ccc [at]      То же, что выше, но показывая измененные строки
| ccd [at]      Сравнить два столбца дизассемблирования, ограниченных размером блока
| ccdd [at]     Сравнить вывод декомпилятора (е cmd.pdc=pdg|pdd)
| cf [file]    Сравнить с содержимым файлов по текущему смещению
| cg[?] [o] [file] Сравнение графов текущего файла и файла [file]
| cu[?] [addr] @at Сравнить дампы памяти по адресу $$ и dst, используя формат unified diff
| cud [addr] @at Unified diff disasm from $$ and given address
| cv[1248] [hexpairs] @at Сравнить 1,2,4,8-байтовые последовательности (silent return in $?)
| cV[1248] [addr] @at Сравнить 1,2,4,8-байтовые последовательности, задаваемые адресом (silent, return in $?)
| cw[?] [us?] [...] Сравнить watcher-ов памяти
| cx [hexpair]  Сравнить строки шестнадцатеричных кодов (используйте '.' для обозначения позиции в шаблоне)
| cx* [hexpair] Сравнить строки шестнадцатеричных кодов (вывод в виде команд r2)
| cX [addr]     Аналогично 'cc', но используя hexdiff
| cd [dir]      Перейти в директорию
| cl|cls|clear Стереть все с экрана, (clear0 - перейти на 0, и только 0)
```

Чтобы сравнить содержимое памяти по текущему смещению с заданной строкой шестнадцатеричных значений используйте `x`:

```
[0x08048000]> p8 4
7f 45 90 46
```

```
[0x08048000]> cx 7f 45 90 46
Compare 3/4 equal bytes
```

```
0x00000002 (byte=03) 90 ' ' -> 4c 'L'  
[0x08048000]>
```

Еще одна команда группы C - это CC (compare code). Сравнение последовательности байтов с последовательностью в памяти по заданному адресу:

```
[0x4A13B8C0]> cc 0x39e8e089 @ 0x4A13B8C0
```

Сравнение содержимого двух функций, указанных по именам:

```
[0x08049A80]> cc sym.main2 @ sym.main
```

Команда C8 сравнивает quadword по текущему смещению (в приведенном ниже примере 0x00000000) с математическим выражением:

```
[0x00000000]> c8 4
```

```
Compare 1/8 equal bytes (0%)  
0x00000000 (byte=01) 7f ' ' -> 04 ' '  
0x00000001 (byte=02) 45 'E' -> 00 ' '  
0x00000002 (byte=03) 4c 'L' -> 00 ' '
```

Параметр может быть математическим выражением, в котором используются имена флагов и все, что разрешено в выражении:

```
[0x00000000]> cx 7f469046
```

```
Compare 2/4 equal bytes  
0x00000001 (byte=02) 45 'E' -> 46 'F'  
0x00000002 (byte=03) 4c 'L' -> 90 ' '
```

Можно использовать команду сравнения, чтобы найти различия между текущим блоком и файлом, ранее сброшенным на диск:

```
r2 /bin/true  
[0x08049A80]> s 0  
[0x08048000]> cf /bin/true  
Compare 512/512 equal bytes
```

База данных строк

Аббревиатура SDB расшифровывается как String DataBase. SDB - это база данных "ключ-значение", работающая только со строками (автор - pancake). Она используется во многих частях r2, база данных хранится на диске и в памяти, является небольшой и быстрой для управления --- хеш-таблица на стероидах. Дисковое представление реализовано на основе djb cdb, поддерживающей JSON и интроспекцию массивов. Также есть библиотека sdbtypes для vala, реализующая несколько структур данных поверх SDB или memcache.

SDB поддерживает

- пространства имен (несколько путей sdb),
- атомарную синхронизацию базы данных,
- привязку для vala, luvit, newlisp и nodejs,
- интерфейс командной строки для баз данных sdb,
- клиент и сервер memcache с бэкэндом sdb,
- массивы (синтаксический сахар),
- интеграцию с json.

Пример использования

Создание базы данных!

```
$ sdb d hello(world  
$ sdb d hello  
world
```

Использование массивов:

```
$ sdb - '[]list=1,2' '[0]list' '[0]list=foo' '[]list' '[+1]list=bar'  
1  
foo  
2  
foo  
bar  
2
```

Проверим работоспособность json:

```
$ sdb d g='{"foo":1,"bar":{"cow":3}}'  
$ sdb d g?bar.cow  
3  
$ sdb - user='{"id":123}' user?id=99 user?id  
99
```

Используя командную строку без дисковой базы данных:

```
$ sdb - foo=bar foo a=3 +a -a  
bar  
4  
3  
  
$ sdb -  
foo=bar  
foo  
bar  
a=3  
+a  
4  
-a  
3
```

Удаление базы данных

```
$ rm -f d
```

Применения базы данных

SDB позволяет хранить данные внутри сеансов radare2! Рассмотрим простой двоичный файл и проверим, что уже SDB-изировано.

```
$ cat test.c  
int main(){  
    puts("Hello world\n");  
}  
$ gcc test.c -o test  
  
$ r2 -A ./test  
[0x08048320]> k **  
bin  
anal  
syscall  
debug  
  
[0x08048320]> k bin/**  
fd.6  
[0x08048320]> k bin/fd.6/*  
archs=0:0:x86:32
```

Файл, соответствующий шестому дескриптору файла, является двоичным файлом x86_32.

```
[0x08048320]> k anal/meta/*  
meta.s.0x080484d0=12,SGVsbG8gd29ybGQ=  
[...]  
[0x08048320]> ?b64- SGVsbG8gd29ybGQ=  
Hello world
```

Строки хранятся в кодировке base64.

Еще примеры

Пространства имен

```
k **
```

Подпространства имен

```
k anal/**
```

Ключи

```
k *  
k anal/*
```

Задание ключа

```
k foo=bar
```

Получение значения ключа

```
k foo
Список всех системных вызовов
k syscall/*~^0x
Список всех комментариев
k anal/meta/*~.c.

Показать комментарий, ассоциированный с заданным смещением:
k %anal/meta/[1]meta.C.0x100005000
```

Dietline

Radare2 поставляется с readline-подобной командной строкой ввода, реализованная на основе ``бережливого'' принципа навигации по истории команд и редактирования содержимого. Она позволяет пользователям передвигать курсор, перебирать предыдущие команды, обеспечивает автодополнение. Благодаря портативности Radare2, Dietline обеспечивает единый пользовательский интерфейс для всех поддерживаемых платформ. Интерфейс используется во всех подоболочках Rarade2: в основной командной строке, оболочке SDB, визуальных подсказках и подсказках адреса смещения. Также реализованы наиболее распространенные функции и сочетания клавиш, совместимые с GNU Readline.

Dietline поддерживает два основных режима конфигурации: режимы Emacs и Vi. Он также поддерживает знаменитый **Ctrl-R**, обратный поиск по истории. Клавиша TAB позволяет прокручивать варианты автодополнения.

Автодополнение

В каждой оболочке и Radare2 поддерживается автодополнение команд. Есть несколько режимов --- файлы, флаги и ключи/пространства имён SDB. Простой способ выбора возможных вариантов завершения --- прокручиваемый всплывающий виджет. Он включается с помощью **scr.prompt.popup**, просто установите его в **true**.

Режим Emacs (по умолчанию)

По умолчанию режим dietline совместим с привязками клавиш режима readline, подобного Emacs. Таким образом, активны:

Перемещение

- **Ctrl-a** - перейти в начало строки,
- **Ctrl-e** - перейти в конец строки,
- **Ctrl-b** - переместиться на один символ назад,
- **Ctrl-f** - переместиться на один символ вперед.

Удаление

- **Ctrl-w** - удалить предыдущее слово,
- **Ctrl-u** - удалить всю строку,
- **Ctrl-h** - удалить символ слева,
- **Ctrl-d** - удалить символ справа,
- **Alt-d** - удалить символ после курсора.

Killing and Yanking

- **Ctrl-k** - удалить текст с текущей точки до конца строки,
- **Ctrl-x** - удалить текст с начала строки до текущей позиции,
- **Ctrl-t** - удалить от текущей точки до конца текущего слова, а если позиция находится между словами, то до конца следующего слова.
- **Ctrl-w** - удалить часть слова левее от текущей позиции, используя пробел в качестве ограничителя. Удаленная часть сохраняется в специальный буфер,
- **Ctrl-y** - вытащить из буфера удаленных строк и поместить в текущую позицию,
- **Ctrl-]** - сдвинуть буфер на одну позицию, и вытащить оттуда новую строку. Вращение можно делать только если предыдущая команда была yank или yank-pop.

История

- **Ctrl-r** - поиск в обратном направлении в истории.

Режим Vi

Radare2 также настраивается на режим vi, подключаемый при помощи `scr.prompt.vi`. В этом режиме доступны сочетания клавиш:

Вход в командные режимы

- `ESC` - вход в режим управления,
- `i` - вход в режим вставки.

Перемещение

- `j` - выполняет функцию стрелки вверх,
- `k` - выполняет функцию стрелки вниз,
- `a` - переместить курсор вперед и перейти в режим вставки,
- `I` - перейти в начало строки и перейти в режим вставки,
- `A` - перейти в конец строки и перейти в режим вставки,
- `^` - перейти в начало строки,
- `0` - перейти в начало строки,
- `$` - перейти в конец строки,
- `h` - передвинуться на одну позицию назад,
- `l` - передвинуться на одну позицию вперед.

Удаление и вставка

- `x` - вырезает символ,
- `dw` - удалить текущее слово,
- `diw` - удалить текущее слово,
- `db` - удалить предыдущее слово,
- `D` - удалить строку целиком,
- `dh` - удалить символ слева,
- `dl` - удалить символ справа,
- `d$` - удалить текст от текущей позиции до конца строки,
- `d^` - удалить текст от текущей позиции до начала строки,
- `de` - удалить текст от текущей позиции до конца слова, а если позиция находится между словами, то до конца следующего слова.
- `p` - вытащить из буфера удаленных строк и поместить в текущую позицию,
- `c` - работают аналогично командам группы d, но переходит в режим вставки в конце операции, если перед командой вставить число, то команда будет выполнена несколько раз.

Если непонятно, в каком режиме vi вы находитесь, установите `scr.prompt.mode=true`, при этом будет происходить обновление цвета командной строки в зависимости от режима vi.

Визуальный режим

Визуальный режим является более удобной альтернативой интерфейсу командной строки radare2. Он обеспечивает удобную навигацию, поддерживает режим курсора для выбора байтов и предлагает множество привязок клавиш для упрощения использования отладчика. Чтобы войти в визуальный режим, используйте команду V. Чтобы выйти из него обратно в командную строку, нажмите клавишу Q.

Навигация

Навигация может осуществляться с помощью HJKL, клавиш со стрелками и PgUp/PgDown. Также работают клавиши Home и End. Как в Vim, действия можно повторять, предваряя навигационную клавишу номером, например, `5j` будет двигаться вниз на пять строк, а `2l` будет перемещаться на два символа вправо.

режимы отображения (панели)

Визуальный режим использует «режимы отображения», представляемые панелями, содержимое которых настраивается. По умолчанию это:

- Панель дампа → Панель дизассемблирования → Панель отладки → Панель дампа Word → Панель дампа без Нех → Панель цветовой разметки Ор-анализа → Панель аннотированного дампа (Панели дампов отображают байты в шестнадцатеричном виде) □.

```
[0x00404890 16% 120 /bin/ls]> pd $r @ entry0
[0x00404890 16% 120 /bin/ls]> pc @ entry0
#define _BUFSIZE 120
unsigned char buffer[120] = {
0x31, 0xed, 0x49, 0xb9, 0xd1, 0x5e, 0x48, 0xb9, 0xe2, 0x48, 0xb3,
0xe4, 0xf0, 0x50, 0x54, 0x49, 0xc7, 0xc0, 0xd0, 0x1e, 0x41, 0x00,
0x48, 0xc7, 0xc1, 0x60, 0x1e, 0x41, 0x00, 0x48, 0xc7, 0xc7, 0xc0,
0x28, 0x40, 0x00, 0xe8, 0x37, 0xdc, 0xff, 0xf4, 0x66, 0x0f,
0x1f, 0x44, 0x00, 0x00, 0xb8, 0xff, 0xa5, 0x61, 0x00, 0x55, 0x48,
0x2d, 0xf8, 0xa5, 0x61, 0x00, 0x48, 0xb3, 0xf8, 0x0e, 0x48, 0xb9,
0x5, 0x77, 0x02, 0x5d, 0xc3, 0xb8, 0x00, 0x00, 0x00, 0x00, 0x48,
0x85, 0xc0, 0x74, 0xf4, 0x5d, 0xbf, 0xf8, 0xa5, 0x61, 0x00, 0xff,
0xe0, 0x0f, 0x1f, 0x80, 0x00, 0x00, 0x00, 0x00, 0xb8, 0xf8, 0xa5,
0x61, 0x00, 0x55, 0x48, 0x2d, 0xf8, 0xa5, 0x61, 0x00, 0x48, 0xc1,
0xf8, 0x03, 0x48, 0xb9, 0xe5, 0x48, 0xb9, 0xc2, 0x48, 0xc1, };

[0x00404890 16% 120 /bin/ls]> pd $r @ entry0
/ (fcn) entry0 42
| -- entry0:
| 0x00404890 31ed xor ebp, ebp
| 0x00404892 4989d1 mov r9, rdx
| 0x00404895 5e pop rsi
| 0x00404896 4889e2 mov rdx, rsp
| 0x00404899 4883e4f0 and rsp, 0xfffffffffffffff0
| 0x0040489d 50 push rax
| 0x0040489e 54 push rsp
| 0x0040489f 49c7c0d01e41. mov r8, 0x411ed0
| 0x004048a0 48c7c1601e41. mov rcx, 0x411e60
| 0x004048a1 48c7c7c02840. mov rdi, main ; "AWAVAUATUH..S..H..
| 0x004048b4 e837dcffff call sym.imp._libc_start_main ;[1]
|     sym.imp._libc_start_main(unk, unk, unk)
\ 0x004048b9 f4 hlt
0x004048ba 660f1f440000 nop word [rax + rax]

/ (fcn) fcn.004048c0 41
| ; CALL XREF from 0x0040493d (fcn.00404930)
| 0x004048c0 b8ff a56100 mov eax, 0x61a5ff ; "hstrtab" @ 0x61a5ff
| 0x004048c5 55 push rbp
| 0x004048c6 4820df8a56100 sub rax, 0x61a5f8
| 0x004048cc 4883f80e cmp rax, 0xe
| 0x004048d0 4889e5 mov rbp, rsp

[0x00404890 16% 368 /bin/ls]> x @ entry0
offset - 0 1 2 3 4 5 6 7 8 9 A B C D E F 0123456789ABCDEF
0x00404890 31ed 4989 d15e 4889 e248 83e4 f050 5449 1.I..^H..H...PTI
0x004048a0 c7c0 d01e 4100 48c7 c160 1e41 0048 c7c7 ....A.H..`A.H..
0x004048b0 c028 4000 e837 dcff ffd4 660f 1f44 0000 .@..7...f..D..
0x004048c0 b8ff a561 0055 482d f8a5 6100 4883 f80e ..a.UH-..a.H...
0x004048d0 4889 e577 025d c3b8 0000 0000 4885 c074 H.w.]....H.t
0x004048e0 f45d bff8 a561 00ff e00f 1f80 0000 0000 .]...a.....
0x004048f0 bbf8 a561 0055 482d f8a5 6100 48c1 f803 ..a.UH-..a.H...
0x00404900 4889 e548 89c2 48c1 ea3f 4801 d048 d1f8 H..H..H..?H..H..
0x00404910 7502 5dc3 ba00 0000 0048 85d2 74f4 5d48 u.].....H.t]H
0x00404920 89c6 bff8 a561 00ff e20f 1f80 0000 0000 .....a.....
0x00404930 803d 215d 2100 0075 1155 4889 e5e8 7e[.=!]!..u.UH-~.
0x00404940 ffcc 5dc6 050e 5d21 0001 f3c3 0f1f 4000 .]...].]....@.
0x00404950 4883 3da8 5421 0000 741e b800 0000 0048 H.=.T!.t.....H
0x00404960 85c0 7414 55bf 009e 6100 4889 e5ff d05d ..t.U...a.H...]
0x00404970 e97b ffff f0f 1f00 e973 ffff f0f 1f00 .{.....s .....
0x00404980 488b 0731 d248 f7f6 4889 d0c3 0f1f 4000 H..1.H..H..@.
0x00404990 31c0 488b 1648 3917 7406 f3c3 0f1f 4000 1.H..H9.t....@.
0x004049a0 488b 4608 4839 4708 0f94 c0c3 0f1f 4000 H.F..H9G.....@.
0x004049b0 8b05 8266 2100 85c0 7506 893d 7866 2100 ...f!...u.=xf!.
0x004049c0 f3c3 6666 6666 662e 0f1f 8400 0000 0000 ...fffff.....
0x004049d0 e91b d8ff ff66 662e 0f1f 8400 0000 0000 ....ff.....
/ (fcn) entry0 42
| -- entry0:
| 0x00404890 31ed xor ebp, ebp
| 0x00404892 4989d1 mov r9, rdx
| 0x00404895 5e pop rsi
| 0x00404896 4889e2 mov rdx, rsp
| 0x00404899 4883e4f0 and rsp, 0xfffffffffffffff0
| 0x0040489d 50 push rax
| 0x0040489e 54 push rsp
| 0x0040489f 49c7c0d01e41. mov r8, 0x411ed0
```

Рис. 8: Визуальный режим

Обратите внимание, что верхняя часть панели содержит используемую сейчас команду, например, для режима дизассемблирования:

```
[0x00404890 16% 120 /bin/ls]> pd $r @ entry0
```

Получение инструкций

Чтобы просмотреть справку по всем ключевым функциям, определенным для визуального режима, нажмите ?:

Справка по визуальному режиму:

```
?      показать эту справку
??     показать удобный для пользователя hud
%      в режиме курсора находит подходящую пару или переключает автоблокировку
@      пересыпывать экран каждые 1 с (многопользовательский вид)
^      установить смещение на начало функции
!      войти в режим визуальных панелей
-      введите флаг/комментарий/функции/.. hud (то же, что и VF_)
=      задать cmd.vprompt (верхняя строка)
|      задать cmd.cprompt (правый столбец)
.      установить смещение на значения программного счетчика
\      переключить режим визуального разделения
"      переключите режим столбца (использует pC..)
/      в режиме курсора поиск в текущем блоке
:cmd   выполнить команду радара
;[-]cmt добавить/удалить комментарий
0      установить смещение на начало текущей функции
[1-9]   следовать jmp/call, обозначенному ярлыком (например ;[1])
,file   add a link to the text file
/*--[] изменить размер блока, [] = изменить размер hex.cols
</>   поиск в соответствии с размером блока (поиск курсора в режиме курсора)
a/A   ассемблировать код, в визуальном режиме - (A)ссемблер
b      просматривать символы, флаги, конфигурации, классы, ...
B      включить/выключить точку останова
c/C   переключатель (c)ursor и (C)olors
d[f?] определить функцию, данные, код, ..
D      войти в режим визуального сравнения (установить diff.from/to)
e      редактирование конфигурационных переменных среды
f/F   установить/удалить или просматривать флаги. f - удаление, F - просмотр, ...
gG   установить смещение на начало и конец файла (0-$s)
hjkl  перемещение (или HJKL) (влево-вниз-вверх-вправо)
i      вставка шеснадцатеричного кода или строки (in hexdump), используйте tab для переключения
mK/'K пометить/перейти к ключу (любой ключ)
M      обзор подмонтированной файловой системы
n/N   установить смещение на следующую/предыдущую функцию/флаг/хит (scr.nkey)
g      установить смещение на заданное смещение
o      переключить режимы отображения asm.pseudo и asm.esil
p/P   переключение режимов отображения (hex, disasm, debug, words, buf)
q      возврат к командной строке радара
r      обновить экран / в режиме курсора - просмотр комментариев
R      случайным образом перенастроить палитру цветов (ecr)
sS   step / step over
t      просмотр типов
T      вход в текстовый журнал чата консоли (TT)
uU   undo/redo установки смещения
v      меню анализа кода функций/переменных (визуальный режим)
V      (V)просмотр графа управления с использованием cmd.graph (agv?)
wW   установить курсор на следующее/предыдущее слово
xX   показывает xrefs/refs текущей функции из/в данные/код
yY   копирование и вставка выбранного региона
z      fold/unfold комментариев в дизассемблировании
Z      включение/выключение масштабирования
Enter установить адрес на цель jump/call
```

Функциональные клавиши: (см. 'e key.'), по умолчанию:

```
F2      включить/выключить точку останова,
F4      выполнить до курсора,
F7      single step (шаг отладки),
F8      step over (шаг, не входя в вызов),
F9      continue (продолжить).
```

Визуальный дизассемблер

Навигация

Перемещение внутри дизассемблования осуществляется с помощью клавиш со стрелками или hjkl. Используйте g для смещения непосредственно к флагу или адресу, введите его в командной строки: [offset]>. Переходы по jump и

call - клавиши цифр на клавиатуре [0 - 9], а также номер справа в окне дизассемблирования. В этом примере ввод 1 на клавиатуре будет следовать за вызовом `sym.imp._libc_start_main` и, следовательно, устанавливать смещение на адрес этого символа.

```
0x00404894      e857dcffff    call sym.imp._libc_start_main ;[1]
```

Вернуться на предыдущее смещение - клавиша U, U - повторение поиска.

Команда d - определить

Клавиша d используется для изменения типа данных текущего блока, доступно несколько основных типов/структур, а также более продвинутый с использованием шаблона pf :

```
d → ...
0x004048f7      48c1e83f      shr rax, 0x3f
d → b
0x004048f7 .byte 0x48
d → B
0x004048f7 .word 0xc148
d → d
0x004048f7 hex length=165 delta=0
0x004048f7 48c1 e83f 4801 c648 d1fe 7415 b800 0000
...
```

Для улучшения удобочитаемости кода можно изменить способ представления числовых значений radare2 при дизассемблировании, по умолчанию большинство вариантов отображают числовое значение как шестнадцатеричное. Иногда надо числа представлять в десятичной, двоичной системах счисления и даже печатать пользовательские константы. Чтобы изменить формат значения, используется d, за которым следует i, затем надо выбрать, в какой системе счисления работать, это эквивалентно ah:i:

```
d → i → ...
0x004048f7      48c1e83f      shr rax, 0x3f
d → i → 10
0x004048f7      48c1e83f      shr rax, 63
d → i → 2
0x004048f7      48c1e83f      shr rax, '?'
```

Использование курсора для вставки/исправления...

Помните, что возможность редактировать файла доступна только, если файл загружен в radare2 с использованием флага -w. В противном случае файл открывается в режиме "только для чтения".

При нажатии строчной буквы C radare переключается режим курсора. Когда этот режим активен, выделенный в данный момент байт (или диапазон байтов) подсвечивается.

```
[0x00404890 16% 330 (0x6:-1=1)]> pd $r @ entry0+6 # 0x404896
/ (fcn) entry0 42
|-- entry0:
|   0x00404890 31ed      xor ebp, ebp
|   0x00404892 4989d1    mov r9, rdx
|   0x00404895 5e        pop rsi
|   0x00404896 * 4889e2    mov rdx, rsp
|   0x00404899 4883e4f0    and rsp, 0xfffffffffffffff0
|   0x0040489d 50        push rax
|   0x0040489e 54        push rsp
|   0x0040489f 49c7c0d01e41. mov r8, 0x411ed0
|   0x004048a6 48c7c1601e41. mov rcx, 0x411e60
|   0x004048ad 48c7c7c02840. mov rdi, main           ; "AWAVAUATUH..S..H...." @ 0x4028c0
|   0x004048b4 e837dcffff    call sym.imp._libc_start_main ;[1]
|       sym.imp._libc_start_main(unk, unk)
|   0x004048b9 f4        hlt
|   0x004048ba 660f1f440000    nop word [rax + rax]
```

Рис. 9: Курсор по адресу 0x00404896

Курсор используется для выбора диапазона байтов или просто для указания конкретного байта. Его можно использовать для создания именованного флага по указанному смещению. Для этого установите смещение, затем нажмите клавишу f и введите имя флага. Если файл был открыт в режиме записи (флаг -w или команды o+), то курсор может перезаписывать выбранный диапазон новыми значениями. Для этого выберите диапазон байтов (при нажатии клавиш HJKL и SHIFT), затем нажмите клавишу i и введите шестнадцатеричные значения для новых данных. Данные будут повторяться по мере необходимости для заполнения выбранного диапазона. Например:

```
<выбрать 10 байт в визуальном режиме, используя SHIFT+HJKL>
<нажать 'i' и затем ввести '12 34'>
```

Выбранные десять байтов будут изменены на ``12 34 12 34 12 12...''.

Визуальный ассемблер --- это функция, которая обеспечивает предварительный просмотр в реальном времени при вводе новых инструкций исправления в ассемблерном коде. Чтобы использовать эту функцию, найдите или поместите курсор в нужное место и нажмите клавишу «A». Чтобы ввести несколько инструкций подряд, разделите их точкой с запятой - ;.

Перекрасные ссылки

Во время анализа radare2 обнаруживает перекрасные ссылки (XREF), эта информация в визуальном дизассемблировании показывается с помощью тега XREF :

; DATA XREF from 0x00402e0e (unk)
str.David_MacKenzie:

Чтобы увидеть, где вызывается этот флаг, нажмите x, если вы хотите перейти к месту, где используются данные, затем нажмите соответствующую цифру [0-9] на клавиатуре. (Эта функциональность аналогична `axt`)

X соответствует обратной операции $\alpha x f$.

Отображение аргументов функции

Чтобы включить это представление, используйте переменную конфигурации `e_dbg.funcarg = true`

```
[0x5621e0acd83 190 /bin/ls]>=?0;f tmp;s..
- offset - 0 1 2 3 4 5 6 7 8 9 A B C D E F 0123456789ABCDEF
0x7ffcf13a0e0 20a1 13fd fc7f 0000 0240 264a ba7f 0000 .....@&J...
0x7ffcf13a0f0 90e4 20e1 2156 0000 60e2 20e1 2156 0000 .. .!V.. .!V..
0x7ffcf13a100 0000 0000 0000 0000 10e0 20e1 2156 0000 ..... .!V..
0x7ffcf13a110 0000 0000 0000 0000 0000 0000 0000 0000 ..... .
rax 0x5621e120f850 = r_ar rbx 0x7ffcf13a248 e->ana rcx 0x7ffcf139f64
rdx 0x00000000 t (fcn) { r8 0x00000081 r9 0x7fba4a5c9a20
r10 0xfffffffffffffec8 f r11 0x00000000name; r12 0x5621e0ace600
r13 0x7ffcf13a240 lse { r14 0x00000000 r15 0x00000000
rsi 0x5621e0ae0261 t rdi 0x5621e0ae0247 rsp 0x7ffcf13a0e0
rbp 0x00000001 rip 0x5621e0acd8at en = rflags 1I_l (core->flags, pc)
orax 0xfffffffffffffff
arg [0] -domainname: 0x5621e0ae0247 --> "coreutils" tem->name;
arg [1] -dirname: 0x5621e0ae0261 --> "/usr/share/locale"
0x5621e0acd83 488d3dbd3401. lea rdi, [0x5621e0ae0247]
;-- rip:
0x5621e0acd8a t = r_e881fafffffunc_a call sym.imp.bindtextdomain ;[1]
0x5621e0acd8f g_str 488d3db13401. or lea rdi, [0x5621e0ae0247] ;[2]
0x5621e0acd96 { e835faffff call sym.imp.textdomain ;[2]
0x5621e0acd9b cons_p488d3dae8100. lea rdi, [0x5621e0ad4f50]
0x5621e0accda2 c7054cb42100. mov dword obj.exit_failure, 2
0x5621e0accdac cons_pe89f1a0100 call 0x5621e0ade850 ;[3]
0x5621e0accdb1 48b800000000. movabs rax, 0x80000000000000000000
0x5621e0accdbb t_empt c7054bc32100. mov dword [0x5621e0ce9110], 0
0x5621e0accdc5 t_argcc605ecc32100. mov byte [0x5621e0ce91b8], 1
0x5621e0accdcc list_f4889059dc421. mov qword [0x5621e0ce9270], rax
0x5621e0accdd3 8b0507b42100. mov eax, dword obj.ls_modesource
0x5621e0accdd9 48c7059cc421. mov qword [0x5621e0ce9280], 0
0x5621e0accde4 48c70589c421. mov qword [0x5621e0ce9278], 0xffff
0x5621e0accdef c605e2c32100. mov byte [0x5621e0ce91d8], 10 _RE_
0x5621e0accdf6 83f802 format_val cmp eax, 2. arg->fmt, on;t2ck, ai
,=< 0x5621e0accdf9 0f844f0c0000 je 0x5621e0acda4e ;[4]
| 0x5621e0accdff 83f803 cmp eax, 3 ; 3
,==< 0x5621e0acce02 740e je 0x5621e0acce12 ;[5]
|| 0x5621e0acce04 (fcn) 83e801 sub eax, 1
---- 0x5621e0acce07 0f8487080000 je 0x5621e0acd694 ;[6]
```

Рис. 10: funcarg

Добавление комментария

Чтобы добавить комментарий, нажмите ;.

Введите другие команды

Чтобы вводить команды, используйте :.

Поиск

/: позволяет подсвечивать строки на текущем дисплее. :cmd позволяет использовать одну из команд ``?'' , выполняющих специализированный поиск.

HUD-ы

UserFriendly HUD

Доступ к ``UserFriendly HUD'' можно получить с помощью комбинации клавиш ?? . HUD действует как интерактивная шпаргалка, которую можно использовать для более легкого поиска и выполнения команд. Он особенно полезен для новичков. Для опытных пользователей есть другие HUD-ы, более специфичные для конкретной деятельности.

HUD для флагов/комментариев/функций/...

HUD отображается с помощью клавиши _, он показывает список всех определенных флагов и позволяет переходить к ним. С помощью клавиатуры вы можете быстро отфильтровать список до флага, который содержит определенный шаблон.

Режим ввода HUD можно закрыть с помощью ^C. Он также завершится при нажатии backspace, когда строка ввода пользователя пуста.

Настройка дизассемблирования

Внешний вид дизассемблирования контролируется с помощью конфигурационных ключей ``asm.*'', изменяемых с помощью команды e. Все ключи конфигурации также можно редактировать с помощью редактора визуальной конфигурации.

Визуальный редактор конфигурации

Доступ к этому HUD можно получить с помощью клавиши e в визуальном режиме. Редактор позволяет легко просматривать и изменять конфигурацию radare2. Например, если вы хотите что-то изменить в дисплее дизассемблирования, выберите asm из списка, перейдите к элементу, который вы хотите изменить, а затем выберите его, нажав Enter. Если элемент является логической переменной, он будет переключаться, в противном случае будет предложено ввести новое значение.

Пример переключения на псевдодизассемблирование:

Ниже приведены некоторые примеры настроек переменных среды, связанных с дизассемблированием.

Примеры

asm.arch: Изменение архитектуры && asm.bits: Размер слова в битах на ассемблере Просмотр списка всех дуг с помощью e asm.arch=?

```
e asm.arch = dalvik
0x00404870      31ed4989      cmp-long v237, v73, v137
0x00404874      d15e4889      rsub-int v14, v5, 0x8948
0x00404878      e24883e4      ushr-int/lit8 v72, v131, 0xe4
0x0040487c      f0505449c7c0  +invoke-object-init-range {}, method+18772 ;[0]
0x00404882      90244100      add-int v36, v65, v0

e asm.bits = 16
0000:4870      31ed          xor bp, bp
0000:4872      49             dec cx
0000:4873      89d1          mov cx, dx
0000:4875      5e             pop si
0000:4876      48             dec ax
0000:4877      89e2          mov dx, sp
```

Последняя операция также может быть выполнена с помощью & в визуальном режиме.

```
[EvalSpace]
```

```
anal
> asm
bin
cfg
cmd
dbg
diff
dir
esil
file
fs
graph
hex
http
hud
io
key
magic
pdb
rap
rop
scr
search
stack
time
zoom
```

```
Sel:asm.arch
```

```
/ (fcn) entry0 42
|     ;-- entry0:
|     0x00404890  31ed          xor    ebp, ebp
|     0x00404892  4989d1        mov    r9, rdx
|     0x00404895  5e             pop    rsi
|     0x00404896  4889e2        mov    rdx, rsp
|     0x00404899  4883e4f0      and    rsp, 0xfffffffffffffff0
```

Рис. 11: Выберите сначала asm

```
[EvalSpace < Variables: asm.arch]
```

```
asm.functions = true
asm.indent = false
asm.lbytes = true
asm.lines = true
asm.linescall = false
asm.linesout = true
asm.linesright = false
asm.linesstyle = false
asm.lineswide = false
asm.linewidth = 7
asm.maxrefs = 5
asm.middle = false
asm.nbytes = 6
asm.offset = true
asm.os = linux
asm.parser = x86.pseudo
> asm.pseudo = false
asm.reloff = false
asm.section = false
asm.segoff = false
asm.size = false
asm.stackptr = false
asm.syntax = intel
asm.tabs = 0
asm.trace = false
asm.tracespace = false
asm.ucase = false
asm.vars = true
asm.varsub = true
asm.varxs = false
asm.xrefs = true
```

```
Selected: asm.pseudo (Enable pseudo syntax)
```

```
/ (fcn) entry0 42
|   ;-- entry0:
|   0x00404890  31ed          xor  ebp, ebp
|   0x00404892  4989d1        mov  r9, rdx
|   0x00404895  5e             pop  rsi
|   0x00404896  4889e2        mov  rdx, rsp
|   0x00404899  4883e4f0      and  rsp, 0xfffffffffffffff0
```

Рис. 12: Псевдодизассемблирование запрещено

```
[EvalSpace < Variables: asm.arch]
```

```
asm.functions = true
asm.indent = false
asm.lbytes = true
asm.lines = true
asm.linescall = false
asm.linesout = true
asm.linesright = false
asm.linesstyle = false
asm.lineswide = false
asm.linewidth = 7
asm.maxrefs = 5
asm.middle = false
asm.nbytes = 6
asm.offset = true
asm.os = linux
asm.parser = x86.pseudo
> asm.pseudo = true
asm.reloff = false
asm.section = false
asm.segoff = false
asm.size = false
asm.stackptr = false
asm.syntax = intel
asm.tabs = 0
asm.trace = false
asm.tracespace = false
asm.ucase = false
asm.vars = true
asm.varsub = true
asm.varxs = false
asm.xrefs = true
```

```
Selected: asm.pseudo (Enable pseudo syntax)
```

```
/ (fcn) entry0 42
    ;-- entry0:
    0x00404890  31ed          ebp = 0
    0x00404892  4989d1        r9 = rdx
    0x00404895  5e             pop rsi
    0x00404896  4889e2        rdx = rsp
    0x00404899  4883e4f0      rsp &= 0xfffffffffffff0
```

Рис. 13: Псевдодизассемблирование разрешено

asm.pseudo: Включить псевдосинтаксис

```
e asm.pseudo = true
0x00404870    31ed        ebp = 0
0x00404872    4989d1      r9 = rdx
0x00404875    5e          pop rsi
0x00404876    4889e2      rdx = rsp
0x00404879    4883e4f0    rsp &= 0xfffffffffffffff0
```

asm.syntax: Выбор синтаксиса дизассемблирования (intel, att, masm...)

```
e asm.syntax = att
0x00404870    31ed        xor %ebp, %ebp
0x00404872    4989d1      mov %rdx, %r9
0x00404875    5e          pop %rsi
0x00404876    4889e2      mov %rsp, %rdx
0x00404879    4883e4f0    and $0xfffffffffffffff0, %rsp
```

asm.describe: Показать описание оп-кода

```
e asm.describe = true
0x00404870 xor ebp, ebp ; logical exclusive or
0x00404872 mov r9, rdx ; moves data from src to dst
0x00404875 pop rsi ; pops last element of stack and stores the result in argument
0x00404876 mov rdx, rsp ; moves data from src to dst
0x00404879 and rsp, -0xf ; binary and operation between src and dst, stores result on dst
```

Визуальный ассемблер

Визуальный режим можно использовать для ассемблирования кода - клавиша A. Например, давайте заменим push на jmp:

Обратите внимание на новый вариант дизассемблирования и новые стрелки, отражающие переходы:

Нужно открыть файл в режиме записи (r2 -w или oo+), только так можно вносить исправления в файл. Также можно использовать режим кэширования: e io.cache = true и wc?.

Помните, что исправление файлов в режиме отладки только исправляет память, а не файл.

Визуальный редактор конфигурации

Команда Ve или e в визуальном режиме позволяют редактировать конфигурацию radare2 визуально. Например, если вы хотите изменить режим отображения дизассемблирования, выберите asm в списке и выберите вариант отображения.

Пример переключения на псевдодизассемблирование:

Визуальные панели

Концепция

Визуальные панели характеризуются следующими основными функциональными возможностями:

1. Разделение экрана,
2. Отображение нескольких экранов, таких как символы, регистры, стек, а также пользовательские панели,
3. Меню охватывает все эти часто используемые команды, так что вам не нужно запоминать ни одну из них.

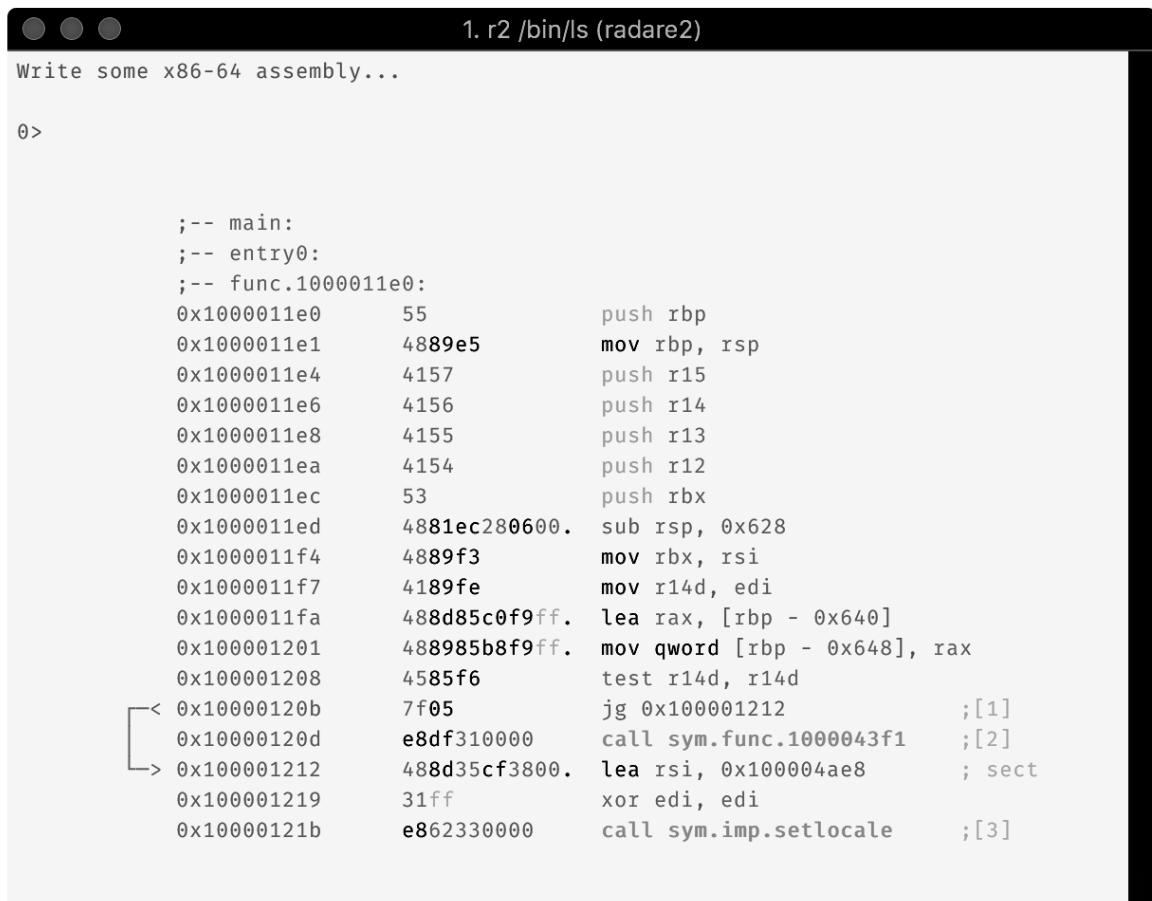
Для CUI разработаны полезные графические интерфейсы в качестве меню, то есть визуальные панели.

Доступ к панелям можно получить с помощью команды V или с помощью ! из визуального режима.

Обзор

Команды

```
|Visual Ascii Art Panels:
| |    разделить панель вертикально
| -    разделить панель горизонтально
| :    выполнить команду при помощи командной строки r2
| ;    добавить/удалить комментарий
| \    запуск hud-режима (input)
| ?    показать user-friendly hud
| ?    показать справку
```



1. r2 /bin/ls (radare2)

Write some x86-64 assembly...

0>

```
;-- main:  
;-- entry0:  
;-- func.1000011e0:  
0x1000011e0      55          push rbp  
0x1000011e1      4889e5      mov rbp, rsp  
0x1000011e4      4157        push r15  
0x1000011e6      4156        push r14  
0x1000011e8      4155        push r13  
0x1000011ea      4154        push r12  
0x1000011ec      53          push rbx  
0x1000011ed      4881ec280600. sub rsp, 0x628  
0x1000011f4      4889f3      mov rbx, rsi  
0x1000011f7      4189fe      mov r14d, edi  
0x1000011fa      488d85c0f9ff. lea rax, [rbp - 0x640]  
0x100001201      488985b8f9ff. mov qword [rbp - 0x648], rax  
0x100001208      4585f6      test r14d, r14d  
0x10000120b      7f05        jg 0x100001212 ;[1]  
0x10000120d      e8df310000  call sym.func.1000043f1 ;[2]  
0x100001212      488d35cf3800. lea rsi, 0x100004ae8 ; sect  
0x100001219      31ff        xor edi, edi  
0x10000121b      e862330000  call sym.imp.setlocale ;[3]
```

Рис. 14: Было

1. r2 /bin/ls (radare2)

Write some x86-64 assembly...

```
2> jmp 0x1000011ec
* eb0a

    <-- main:
    <-- entry0:
    <-- func.1000011e0:
    < 0x1000011e0      eb0a          jmp 0x1000011ec           ;[1]
    0x1000011e2      89e5          mov ebp, esp
    0x1000011e4      4157          push r15
    0x1000011e6      4156          push r14
    0x1000011e8      4155          push r13
    0x1000011ea      4154          push r12
    > 0x1000011ec      53            push rbx
    0x1000011ed      4881ec280600. sub rsp, 0x628
    0x1000011f4      4889f3          mov rbx, rsi
    0x1000011f7      4189fe          mov r14d, edi
    0x1000011fa      488d85c0f9ff. lea rax, [rbp - 0x640]
    0x100001201      488985b8f9ff. mov qword [rbp - 0x648], rax
    0x100001208      4585f6          test r14d, r14d
    < 0x10000120b      7f05          jg 0x100001212        ;[2]
    0x10000120d      e8df310000  call sym.func.1000043f1   ;[3]
    > 0x100001212      488d35cf3800. lea rsi, 0x100004ae8  ; sect
    0x100001219      31ff          xor edi, edi
    0x10000121b      e862330000  call sym.imp.setlocale ;[4]
```

Рис. 15: Стало

```
[EvalSpace]
```

```
anal
> asm
bin
cfg
cmd
dbg
diff
dir
esil
file
fs
graph
hex
http
hud
io
key
magic
pdb
rap
rop
scr
search
stack
time
zoom
```

```
Sel:asm.arch
```

```
/ (fcn) entry0 42
|     ;-- entry0:
|     0x00404890  31ed          xor    ebp, ebp
|     0x00404892  4989d1        mov    r9, rdx
|     0x00404895  5e             pop    rsi
|     0x00404896  4889e2        mov    rdx, rsp
|     0x00404899  4883e4f0      and    rsp, 0xfffffffffffffff0
```

Рис. 16: Выберите сначала asm

```
[EvalSpace < Variables: asm.arch]
```

```
asm.functions = true
asm.indent = false
asm.lbytes = true
asm.lines = true
asm.linescall = false
asm.linesout = true
asm.linesright = false
asm.linesstyle = false
asm.lineswide = false
asm.linewidth = 7
asm.maxrefs = 5
asm.middle = false
asm.nbytes = 6
asm.offset = true
asm.os = linux
asm.parser = x86.pseudo
> asm.pseudo = false
asm.reloff = false
asm.section = false
asm.segoff = false
asm.size = false
asm.stackptr = false
asm.syntax = intel
asm.tabs = 0
asm.trace = false
asm.tracespace = false
asm.ucase = false
asm.vars = true
asm.varsub = true
asm.varxs = false
asm.xrefs = true
```

```
Selected: asm.pseudo (Enable pseudo syntax)
```

```
/ (fcn) entry0 42
|   ;-- entry0:
|   0x00404890  31ed          xor  ebp, ebp
|   0x00404892  4989d1        mov  r9, rdx
|   0x00404895  5e             pop  rsi
|   0x00404896  4889e2        mov  rdx, rsp
|   0x00404899  4883e4f0      and  rsp, 0xfffffffffffffff0
```

Рис. 17: Псевдодизассемблирование запрещено

```
[EvalSpace < Variables: asm.arch]
```

```
asm.functions = true
asm.indent = false
asm.lbytes = true
asm.lines = true
asm.linescall = false
asm.linesout = true
asm.linesright = false
asm.linesstyle = false
asm.lineswide = false
asm.linewidth = 7
asm.maxrefs = 5
asm.middle = false
asm.nbytes = 6
asm.offset = true
asm.os = linux
asm.parser = x86.pseudo
> asm.pseudo = true
asm.reloff = false
asm.section = false
asm.segoff = false
asm.size = false
asm.stackptr = false
asm.syntax = intel
asm.tabs = 0
asm.trace = false
asm.tracespace = false
asm.ucase = false
asm.vars = true
asm.varsub = true
asm.varxs = false
asm.xrefs = true
```

```
Selected: asm.pseudo (Enable pseudo syntax)
```

```
/ (fcn) entry0 42
    ;-- entry0:
    0x00404890  31ed          ebp = 0
    0x00404892  4989d1        r9 = rdx
    0x00404895  5e             pop rsi
    0x00404896  4889e2        rdx = rsp
    0x00404899  4883e4f0      rsp &= 0xfffffffffffff0
```

Рис. 18: Псевдодизассемблирование разрешено

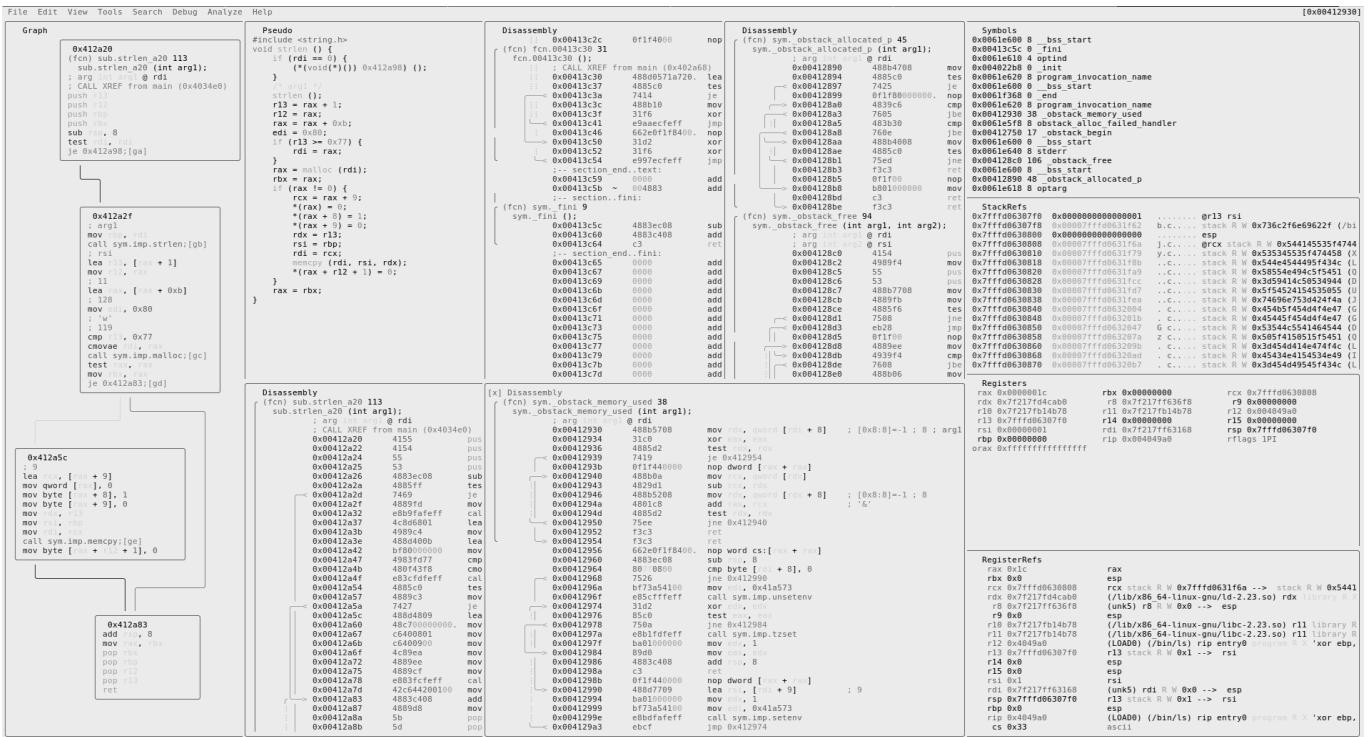


Рис. 19: Обзор панелей

```
!
 запустить игру r2048
 .
 установить смещение равное PC или на точку входа
 *
 показать декомпилятор в текущей панели
 "
 создать панель из списка и заменить текущее
 подсветить ключевые слова
 (
 включить/выключить "снег"
 & включить/выключить кэш
 [1-9] следовать на целевой адрес jmp/call, задаваемый ярлыком (например ;[1])
 ' (пробел) включить/выключить граф управление / режим панелей
 tab перейти на следующую панель
 Enter запуск режима Zoom
 a включить/выключить автообновление декомпилиатора
 b просмотр символов, флагов, конфигурации, классов, ...
 c включить/выключить курсор
 C включить/выключить цветную тему
 d задать сущность по текущему адресу, аналогично Vd
 D показать дизассемблирование в текущей панели
 e сменить заголовок и "команду" текущей панели
 f задать/добавить ключевые слова фильтра
 F удалить все фильтры
 g перейти/установить на смещение на заданный адрес
 G перейти/установить на подсвеченное
 i вставить шеснадцатеричный код
 h jkl перемещение изображения в панели (навигация, влево-вниз-вверх-вправо)
 HJKL перемещение изображения в панели (навигация, влево-вниз-вверх-вправо) постранично
 m выбрать панель меню
 M открыть новую панель
 n/N установить смещение следующей/предыдущей функции/флаг/хит (scr.nkey)
 p/P повернуть расположение элементов на панели
 q выход или закрыть вкладку
 Q закрыть все вкладки и выйти
 r включить/выключить хинты вызов/переход/загрузка адреса
 R сгенерировать палитру из случайных цветов (ecr)
 s/S step in / step over (отладка)
 t/T открыть комендную строку / закрыть ее
 u/U undo / redo установки смещения
 w запуск оконного режима
 V перейти в режим графа
 x/X показать xrefs/refs текущей функции от/в данные/лшв
 z сменить текущую панель с первой
```

Базовые функции

Используйте клавишу TAB для перемещения по панелям, пока не дойдете до целевой. Клавиши hjkl - прокручивание панели, на которой вы сейчас находитесь. Используйте S и S для step over/in, и все панели будут динамически обновляться во время отладки. На панелях «Регистры» или «Стек» значения можно редактировать, вставив шестнадцатеричный формат. Об этом будет рассказано позже.

Использование tab позволяет вам перемещаться между панелями, настоятельно рекомендуется использовать m для открытия меню. Как обычно, вы можете использовать hjkl для перемещения по меню и найдете там тонны полезных вещей. Также можно нажать " для быстрого просмотра различных настроек просмотра предложений и изменения содержимого выбранной панели.

Разделение экрана

Символы | - вертикальное и - - горизонтальное разделение. Можно удалить любую панель, нажав клавишу X.

Размер разделенных панелей может быть изменен из оконного режима, доступ к которому осуществляется с помощью W.

Команды оконного режима

```
| Справка по панели оконный режима:  
| ?      показать эту справку  
| ??     показать user-friendly hud  
| Enter   запуск режима Zoom  
| c      включить/выключить курсор  
| hjkl   перемещение изображения в панели (навигация, влево-вниз-вверх-вправо)  
| JK     изменить размер вертикально  
| HL     изменить размер горизонтально  
| q      выйти из оконного режима
```

Изменение значений

На панели «Регистр» или «Стек» можно редактировать значения. Используйте C для активации режима курсора, перемещать курсор - hjkl, как обычно. Затем нажмите i, как и в режиме вставки vim, чтобы вставить значение.

Вкладки

Визуальные панели также предлагают вкладки для быстрого доступа к нескольким формам информации. Нажмите клавишу t, чтобы войти в режим вкладок. Номера вкладок будут видны в правом верхнем углу.

По умолчанию показывается одна вкладка, можно нажать t для создания новой вкладки с теми же панелями, и T, чтобы создать новую чистую панель.

Для обхода вкладок можно ввести номер вкладки.

И нажатие - удаляет вкладку, в которой вы находитесь.

Сохранение макетов

Можно сохранить пользовательский макет визуальных панелей, выбрав опцию «Сохранить макет» в меню «Файл», либо выполнив:

```
v= test
```

Где test --- это имя, под которым вы хотите сохранить настройки.

Сохраненный макет можно открыть, передав имя в качестве параметра v:

```
v test
```

Дополнительные инструкции можно узнать в разделе v?.

Поиск байтов

Поисковая система radare2 поддерживает поиск по нескольким ключевым словам, двоичным маскам и шестнадцатеричным значениям. Автором подсистемы является esteve, также несколько функций реализована поверх нее. Подсистема автоматически создает флаги для найденных вхождений (hits), позволяя ссылаться на них далее.

Поиск инициируется командой /.

```
[0x00000000]> /?
|Usage: /![bf] [arg] # Инструменты поиска (Инструкция по настройке - 'e??search')
|Используйте io.va для поиска в невиртуализированных адресных пространствах
| / foo\x00           поиск строки 'foo\0'
| /j foo\x00          поиск строки 'foo\0' с вводом в json
| /! ff              поиск первого вхождения, не соответствующего критерию поиска, модификатор команды
| /!x 00             найти первый байт != 0x00
| + /bin/sh          конструировать строку из кусков
| //                повторить предыдущую операцию поиска
| /a jmp eax         ассемблировать оп-код и попытаться найти его байты
| /A jmp             найти проанализированные инструкции данного типа (инструкция - /A?)
| /b                поиск в обратном направлении, модификатор команды, за которым следует команда
| /B                поиск в распознанных заголовках RBin
| /c jmp [esp]       поиск инструкций, соответствующих заданной строке
| /ce rsp,rbp        поиск соответствующих критерию esil-выражений
| /C[ar]            поиск криптографических материалов
| /d 101112          поиск дельтифицированной последовательности из N байт
| /e /E.F/i          проверка соответствия регулярному выражению
| /E esil-expr      смещение подходящего критерия esil-выражения %%= here
| /f                поиск вперед, модификатор команды, за которым идет команда
| /F file [off] [sz] поиск в содержимом файла, ограниченный смещением и размером
| /g[g] [from]       найти все пути в графе из A в B (/gg следует по jmp-ам, смотрите search.count и anal.dept
| /h[t] [hash] [len] найти блок, для которого подходит заданных хэш, подробности в ph
| /i foo            поиск строки 'foo', игнорируя размер букв
| /m magicfile     поиск известных файловых систем и монтировать их автоматически
| /M                показать смещение в п инструкций в обратном направлении
| /o [n]            аналогично /o, но используется другая функция обработки ошибки, если стандартная не может
| /O [n]            искать шаблон заданного размера
| /p patternsize   искать подобные блоки
| /P patternsize   анализировать оп-код, ссылающийся на смещение (/re для esil)
| /r[erwx][?] sym.printf  поиск соответствий для ROP gadgets, разделяются точкой с запятой
| /R [grepopcode]  поиск всех системных вызовов в регионе (ЭКСПЕРИМЕНТАЛЬНАЯ функция)
| /s                поиск 32-битового значения с учетом `cfg.big endian`
| /v[1248] value   поиск 32-битового значения с учетом `cfg.big endian`, ограничен диапазоном
| /V[1248] min max поиск wide-строки 'f\00\00\00'
| /w foo           поиск wide-строки, игнорируя размер букв 'f\00\00\00'
| /wi foo          поиск шестнадцатеричных данных, игнорируя некоторые позиции символов
| /x ff..33         поиск шестнадцатеричных данных
| /x ff0033        поиск шестнадцатеричных данных
| /x ff43:ffd0    поиск шестнадцатеричных данных с маской
| /z min max      поиск шестнадцатеричных данных, ограничен размером
```

Поскольку в radare2 все рассматривается как файл, то не имеет значения, выполняете ли вы поиск в сокете, на удаленном устройстве, в памяти процесса или в файле. Следующая синтаксическая структура '' запускает многострочный комментарий, а не команду поиска. Введите '/', чтобы закончить комментарий.

Обычный поиск

Поиск обычной текстовой строки в файле будет выглядеть примерно так:

```
$ r2 -q -c "/ lib" /bin/ls
Searching 3 bytes from 0x00400000 to 0x0041ae08: 6c 69 62
hits: 9
0x00400239 hit0_0 "lib64/ld-linux-x86-64.so.2"
0x00400f19 hit0_1 "libselinux.so.1"
0x00400fae hit0_2 "librt.so.1"
0x00400fc7 hit0_3 "libacl.so.1"
0x00401004 hit0_4 "libc.so.6"
0x004013ce hit0_5 "libc_start_main"
0x00416542 hit0_6 "libs/"
0x00417160 hit0_7 "lib/xstrtol.c"
0x00417578 hit0_8 "lib"
```

Как видно из вывода выше, radare2 генерирует флаг ``hit'' (вхождение) для каждой найденной записи. Затем можно использовать команду ps, чтобы увидеть строки, хранящиеся в смещениях, отмеченных флагами, имена будут выведены в форме hit0_<номер_вхождения>:

```
[0x00404888]> / ls
...
[0x00404888]> ps @ hit0_0
lseek
```

Можно искать строки с wide-символами (например, буквы юникода) с помощью команды /w:

```
[0x00000000]> /w Hello
0 results found.
```

Для выполнения поиска строк без учета регистра используйте параметр /i:

```
[0x0040488f]> /i Stallman
Searching 8 bytes from 0x00400238 to 0x0040488f: 53 74 61 6c 6c 6d 61 6e
[# ]hits: 004138 < 0x0040488f hits = 0
```

В строке поиска можно указывать шестнадцатеричные escape-последовательности, предварив их \x:

```
[0x00000000]> / \x7FELF
```

Если вместо этого ищете строку шестнадцатеричных значений, вероятно лучше использовать команду /x:

```
[0x00000000]> /x 7F454C46
```

Если надо замаскировать некоторый кусок во время поиска, можно использовать символ . (точка):

```
[0x00407354]> /x 80..80
0x0040d4b6 hit3_0 800080
0x0040d4c8 hit3_1 808080
0x004058a6 hit3_2 80fb80
```

Если вам не известны некоторые значения битов вашего шестнадцатеричного паттерна, можно использовать битовую маску в вашем шаблоне. Каждый бит, равный единице в маске, указывает на поиск значения бита в шаблоне. Бит, равный нулю в маске, указывает, что значение совпадающего значения может быть равно 0 или 1:

```
[0x00407354]> /x 808080:ff80ff
0x0040d4c8 hit4_0 808080
0x0040d7b0 hit4_1 808080
0x004058a6 hit4_2 80fb80
```

Можно заметить, что команда /x 808080:ff00ff эквивалентна команде /x 80..80.

После завершения поиска результаты сохраняются в пространстве флага searches.

```
[0x00000000]> fs
0    0 . strings
1    0 . symbols
2    6 . searches
```

```
[0x00000000]> f
0x00000135 512 hit0_0
0x00000b71 512 hit0_1
0x00000bad 512 hit0_2
0x00000bdd 512 hit0_3
0x00000bfb 512 hit0_4
0x00000f2a 512 hit0_5
```

Чтобы удалить флаги ``hit'', используйте команду f-hit*.

Часто во время длительных сессий поиска нужно запускать последний поиск более одного раза. Вы можете использовать команду // для повторения последнего поиска.

```
[0x00000f2a]> //      ; повторить последний поиск
```

Настройка параметров поиска

Поисковая система radare2 настраивается с помощью нескольких конфигурационных переменных, изменяемых с помощью команды e.

```
e cmd.hit = x      ; команда radare2, которую надо запускать для каждого вхождения (hit)
e search.distance = 0 ; ограничение расстояния при поиске строк
e search.in = [foo]  ; указать границы поиска. Поддерживаемые значения перечислены в разделе e search.in=??
e search.align = 4   ; показывать только результаты поиска, выровненные на заданные границы
e search.from = 0    ; начальный адрес
e search.to = 0      ; конечный адрес
e search.asmstr = 0   ; поиск строки, а не результата ассемблирования
e search.flags = true ; если разрешен, то создать флаг для каждого вхождения
```

Переменная search.align используется для ограничения допустимых поисковых запросов определенным выравниванием. Например, при помощи e search.align=4 будут выведены только результаты, найденные на 4-байтовых смещениях. Логическая переменная search.flags инструктирует поисковую систему помечать вхождения флагами,ими потом можно ссылаться на найденные смещения. Если текущий поиск прерывается с помощью последовательности клавиш Ctrl-C, текущая позиция поиска помечается search_stop.

Поиск сопоставлением с шаблоном

Команда /р позволяет делать повторный поиск шаблонов к серверному хранилищу ввода-вывода. Можно идентифицировать повторяющиеся последовательности байтов без явного их указания. Единственный параметр команды задает минимальную обнаруживаемую длину шаблона.

Пример:

```
[0x00000000]> /p 10
```

Вывод команды покажет различные найденные шаблоны и сколько раз каждый из них встречается.

Возможен поиск шаблонов с известной разницей между последовательными байтами с помощью команды `/d`. Например, команда для поиска всех шаблонов, где у первого и второго байта первые биты различны, а у второго и третьего байтов отличаются вторые биты:

```
[0x00000000]> /d 0102
Searching 2 bytes in [0x0-0x400]
hits: 2
0x00000118 hit2_0 9a9b9d
0x00000202 hit2_1 a4a5a7
```

Автоматизация обработки найденных вхождений

Переменная конфигурации `cmd.hit` используется для определения команды `radare2`, запускающейся в момент, когда найдено соответствие поисковому запросу (*вхождение, hit*). Если вы хотите выполнить несколько команд, разделяйте их `;`. Можно задать команды в отдельном сценарии, а затем выполнить его целиком с помощью команды `. script-file-name`.

Пример использования:

```
[0x00404888]> e cmd.hit = p8 8
[0x00404888]> / lib
Поиск трех байтов с 0x00400000 до 0x0041ae08: 6c 69 62
hits: 9
0x00400239 hit4_0 "lib64/ld-linux-x86-64.so.2"
31ed4989d15e4889
0x00400f19 hit4_1 "libselinux.so.1"
31ed4989d15e4889
0x00400fae hit4_2 "librt.so.1"
31ed4989d15e4889
0x00400fc7 hit4_3 "libacl.so.1"
31ed4989d15e4889
0x00401004 hit4_4 "libc.so.6"
31ed4989d15e4889
0x004013ce hit4_5 "libc_start_main"
31ed4989d15e4889
0x00416542 hit4_6 "libs/"
31ed4989d15e4889
0x00417160 hit4_7 "lib/xstrtol.c"
31ed4989d15e4889
0x00417578 hit4_8 "lib"
31ed4989d15e4889
```

Поиск в обратном направлении

Иногда надо найти что-то, что находится ``выше по тексту''. Для реализованы команды поиска в обратном направлении с заданным критерием поиска --- команда `/b`.

```
# Поиск вперед
[0x100001200]> / nop
0x100004b15 hit0_0 .STUwbcdefghiklmnopqrstuvwxyzbin/ls.
0x100004f50 hit0_1 .STUwbcdefghiklmnopqrstuvwxyz1] [file .
# Поиск назад
[0x100001200]> /b nop
[0x100001200]> s 0x100004f50p
[0x100004f50]> /b nop
0x100004b15 hit2_0 .STUwbcdefghiklmnopqrstuvwxyzbin/ls.
[0x100004f50]>
```

Обратите внимание, что `/b` делает то же самое, что и `/`, но в обратном направлении, так что, если надо использовать `/x` в обратном направлении? Можно использовать `/bx`, аналогичной работают и другие поисковые подкоманды:

```
[0x100001200]> /x 90
0x100001a23 hit1_0 90
0x10000248f hit1_1 90
0x1000027b2 hit1_2 90
0x100002b2e hit1_3 90
0x1000032b8 hit1_4 90
0x100003454 hit1_5 90
0x100003468 hit1_6 90
0x10000355b hit1_7 90
0x100003647 hit1_8 90
0x1000037ac hit1_9 90
0x10000389c hit1_10 90
0x100003c5c hit1_11 90
```

```
[0x100001200]> /bx 90
[0x100001200]> s 0x10000355b
[0x10000355b]> /bx 90
0x100003468 hit3_0 90
0x100003454 hit3_1 90
0x1000032b8 hit3_2 90
0x100002b2e hit3_3 90
0x1000027b2 hit3_4 90
0x10000248f hit3_5 90
0x100001a23 hit3_6 90
[0x10000355b]>
```

Поиск в результатах ассемблирования

Если надо найти определенные оп-коды ассемблера, можно использовать команды `/a`. Команда `/ad/ jmp [esp]` ищет указанную категорию мнемоники ассемблера:

```
[0x00404888]> /ad/ jmp qword [rdx]
f hit_0 @ 0x0040e50d    # 2: jmp qword [rdx]
f hit_1 @ 0x00418dbb    # 2: jmp qword [rdx]
f hit_2 @ 0x00418fcf    # 3: jmp qword [rdx]
f hit_3 @ 0x004196ab    # 6: jmp qword [rdx]
f hit_4 @ 0x00419bf3    # 3: jmp qword [rdx]
f hit_5 @ 0x00419c1b    # 3: jmp qword [rdx]
f hit_6 @ 0x00419c43    # 3: jmp qword [rdx]
```

Команда `/a jmp eax` собирает строку для машинного кода, затем выполняет поиск байтов по-кода:

```
[0x00404888]> /a jmp eax
hits: 1
0x004048e7 hit3_0 ffe00f1f8000000000b8
```

Поиск криптографических данных

Поиск ключей AES

Radare2 способен находить **расширенные ключи AES** с помощью команды `/ca`. Он выполняет поиск от текущей позиции поиска до предела `Search.distance` или до конца файла. Можно прервать текущий поиск, нажав `Ctrl-C`. Например, чтобы найти ключи AES в дампе памяти:

```
0x00000000]> /ca
Searching 40 bytes in [0x0-0x1ab]
hits: 1
0x000000fb hit0_0 6920e299a5202a6d656e636869746f2a
```

Полученная длина соответствует размеру используемого ключа AES: 128, 192 или 256 бит. Если вы просто ищете ключи AES в виде открытого текста в двоичном файле, `/ca` не найдет их, они должны быть расшифрованы специальным алгоритмом.

Поиск закрытых ключей и сертификатов

Команда `/cr` реализует поиск закрытых ключей (RSA и ECC). Команда `/cd` реализует аналогичную функцию поиска сертификатов.

```
[0x00000000]> /cr
Searching 11 bytes in [0x0-0x15a]
hits: 2
0x000000fa hit1_0 302e020100300506032b657004220420fb3d588296fed5694ff7049eafb74490bf4bc6467ee11a08...
```

Энтропийный анализ

р=е дают подсказки, указывая участки с высокой энтропией, пытающиеся скрыть жестко закодированный секрет. Есть возможность разграничить участки энтропии для последующего использования с помощью команды `\s`:

```
[0x00000000]> b
0x100
[0x00000000]> b 4096
[0x00000000]> /s
0x00100000 - 0x00101000 ~ 5.556094
0x014e2c88 - 0x014e3c88 ~ 0.000000
0x01434374 - 0x01435374 ~ 6.332087
0x01435374 - 0x0144c374 ~ 3.664636
0x0144c374 - 0x0144d374 ~ 1.664368
0x0144d374 - 0x0144f374 ~ 4.229199
0x0144f374 - 0x01451374 ~ 2.000000
```

```
(...)
[0x00000000]> /s*
f entropy_section_0 0x00001000 0x00100000
f entropy_section_1 0x00001000 0x014e2c88
f entropy_section_2 0x00001000 0x01434374
f entropy_section_3 0x00017000 0x01435374
f entropy_section_4 0x00001000 0x0144c374
f entropy_section_5 0x00002000 0x0144d374
f entropy_section_6 0x00002000 0x0144f374
```

Размер блока увеличивается до 4096 байт со 100 байт по умолчанию, чтобы энтропийный поиск `/s` мог работать на блоках разумного размера. Флаги разделов могут быть переданы в оператор ``точка'' `./*`, и затем перебраны при помощи `rx 32 @@ entropy*`.

Дизассемблирование

Дизассемблирование в радаре --- это всего лишь способ представления массива байтов. Он обрабатывается как специальный режим печати внутри команды `p`.

В прежние времена, когда ядро радара было меньше, дизассемблер управлялся внешним файлом `rsc`. То есть радар сначала сбрасывал текущий блок в файл, а потом просто вызывал `objdump`, сконфигурированный на дизассемблирование для Intel, ARM и другие поддерживаемые архитектуры.

Это было работающее и дружественное к Unix решение, но оно было неэффективным, так как повторяло одни и те же дорогостоящие действия снова и снова, так же не было кеша. В результате прокрутка была ужасно медленной.

Поэтому возникла необходимость создать универсальную библиотеку дизассемблера для поддержки нескольких плагинов для разных архитектур. Перечислим текущие используемые плагины с помощью

```
$ rasm2 -L
```

или в radare2:

```
> e asm.arch=??
```

За много лет до появления capstone r2 использовал дизассемблеры `udis86` и `olly`, разные от gnu (`binutils`).

В настоящее время поддержка дизассемблера является одной из основных особенностей радара. Теперь у него есть много параметров - порядок следования байтов, включая вариант целевой архитектуры и варианты дизассемблера и др.

Чтобы получить дизассемблированный код, используйте команду `pd`. Он принимает числовой аргумент, указывающий сколько оп-кодов текущего блока вы хотите видеть. Большинство команд в radare считают текущий размер блока ограничением по умолчанию для ввода данных. Если вы хотите дизассемблировать больше байтов, установите новый размер блока с помощью команды `b`.

```
[0x00000000]> b 100 ; установить размер блока равным 100
[0x00000000]> pd ; дизассемблировать сто байт
[0x00000000]> pd 3 ; дизассемблировать три оп-кода
[0x00000000]> pd 30 ; дизассемблировать 30 байт
```

Команда `pd` работает как `pd`, но принимает в качестве аргумента количество байтов вместо количества оп-кодов.

«Псевдо»-синтаксис может быть несколько проще для понимания человеком, чем нотации ассемблера по умолчанию. Но иногда становится раздражающим чтение большого объема кода. Попробуйте использовать разные варианты:

```
[0x00405e1c]> e asm.pseudo = true
[0x00405e1c]> pd 3
; JMP XREF from 0x00405dfa (fcn.00404531)
0x00405e1c    488b9424a80. rdx = [rsp+0x2a8]
0x00405e24    64483314252. rdx ^= [fs:0x28]
0x00405e2d    4889d8      rax = rbx

[0x00405e1c]> e asm.syntax = intel
[0x00405e1c]> pd 3
; JMP XREF from 0x00405dfa (fcn.00404531)
0x00405e1c    488b9424a80. mov rdx, [rsp+0x2a8]
0x00405e24    64483314252. xor rdx, [fs:0x28]
0x00405e2d    4889d8      mov rax, rbx

[0x00405e1c]> e asm.syntax=att
[0x00405e1c]> pd 3
; JMP XREF from 0x00405dfa (fcn.00404531)
0x00405e1c    488b9424a80. mov 0x2a8(%rsp), %rdx
0x00405e24    64483314252. xor %fs:0x28, %rdx
0x00405e2d    4889d8      mov %rbx, %rax
```

Добавление метаданных к тексту дизассемблера

Типичная работа, связанная с взломом двоичных файлов, --- аннотация, она значительно увеличивает производительность процесса. Radare предлагает несколько способов хранения и извлечения таких метаданных.

Следуя общим основным принципам UNIX, легко написать небольшую утилиту на языке сценариев, которая использует `objdump`, `otool` или любую другую существующую утилиту для получения информации из двоичного файла, и импортировать ее результаты в radare. Например, взгляните на `idc2r.py`, которая поставляется с `radare2ida`. Ее использование - запуск `idc2r.py file.idc > file.r2`. Она считывает файл IDC, экспортенный из базы данных IDA Pro, и создает скрипт `r2`, содержащий те же комментарии, имена функций и другие данные. Полученный файл `'file.r2'` можно импортировать с помощью команды ``точка'' (.) `radar-a`:

```
[0x00000000]> . file.r2
```

Команда `.` используется для интерпретации команд Radare из внешних источников, включая файлы и выходные данные программы. Чтобы пропустить генерацию промежуточного файла и импортировать скрипт напрямую, можно использовать такую комбинацию:

```
[0x00000000]> .!idc2r.py < file.idc
```

Импорт метаданных IDA Pro из дампа IDC является устаревшим механизмом и может перестать работать в будущем. Рекомендуемый способ - использовать скрипт `ida2r2.py`, основанный на `python-idb`, он открывает IDB-файлы напрямую без установки IDA Pro.

Команда `C` используется для управления комментариями и отображением данных. Можно снабдить некоторый диапазон байтов метаданными, указывающими как его следует интерпретировать: код, блок двоичных данных или строкой. Также можно запустить внешнюю программу, передав ей часть данных флага. Программа должна провести анализ и передать обратно метаданные, например, комментарии. Данные для анализа никто не запрещает получать из внешнего файла или базы данных, в том числе.

Существует множество различных команд задания метаданных, вот краткий обзор некоторых из них:

```
[0x00404cc0]> C?  
| Usage: C[-LCvsdfm*?][*?][...] # Управление метаданными  
| C                                         перечислить метаданные в виде текста  
| C*                                        перечислить метаданные в виде команд r2  
| C*.                                       перечень метаданных, привязанных к текущему смещению, формат команды  
| C-[len] [[@]addr]                         удалить метаданные, привязанные к заданному диапазону адресов  
| C.                                         перечислить метаданные, привязанные к текущему смещению в виде текста  
| CC![@addr]                                редактировать комментарий при помощи $EDITOR-а  
| CC[?][-] [comment-text] [@addr]           добавить/удалить комментарий  
| CC.[addr]                                 показать комментарий, привязанный к текущему адресу  
| CCa[-at][at] [text] [@addr]                добавить/удалить комментарий, привязанный к текущему адресу  
| CCu [comment-text] [@addr]                добавить уникальный комментарий  
| CF[sz] [fcn-sign..] [@addr]               сигнатура функции  
| CL[-][*] [file:line] [addr]              показать или добавить 'code line'-информацию (bininfo)  
| CS[-][space]                             управление мета-пространствами для фильтрации комментариев и т.п.  
| C[Cthsdmf]                               перечислить комментарии/типы/скрытое/строки/данные/magic-  
и в виде текста                            перечислить комментарии/типы/скрытое/строки/данные/magic-  
| C[Cthsdmf]*  
и в виде команд r2                          сделай шестнадцатеричный дамп массива данных (Cd 4 10 == dword [10]  
| Cd[-] [size] [repeat] [@addr]            показать размер данных по текущему адресу  
| Cd. [@addr]                            задать формат области памяти (смотрите pf?)  
| Cf[?][-] [sz] [0|cnt][fmt] [a0 a1... ] [@addr] скрыть данные  
| Ch[-] [size] [@addr]                      распознать magic (смотрите rm?)  
| Cm[-] [sz] [fmt..] [@addr]                добавить строку  
| Cs[?][-] [size] [@addr]                  добавить/удалить комментарий об анализе типа  
| Ct[?][-] [comment-text] [@addr]          показать содержимое по текущему смещению или заданному адресу  
| Ct. [@addr]                            добавить комментарий к аргументу функции  
| Cv[bsr]?                                добавить строку (смотрите Cs?)  
| Cz[@addr]
```

Добавление комментария к определенной строке/адресу - команда `Ca` :

```
[0x00000000]> CCa 0x00000002 this guy seems legit  
[0x00000000]> pd 2  
0x00000000 0000 add [rax], al  
; this guy seems legit  
0x00000002 0000 add [rax], al
```

Семейство команд `C?` позволяет пометить некоторый диапазон адресов заданным типом. Три основных типа метаданных - это код (дизассемблирование выполняется с помощью `asm.arch`), блок данных (массив элементов данных) или строка. Команда `CS` - определение метаданных для строки, команда `Cd` - для массива элементов данных, команда `Cf` - определение сложных структур данных.

Аннотация типов данных легче всего выполнять в визуальном режиме, используя клавишу `<d>`, сокращение от ``data type change''. Используйте курсор для выбора диапазона байтов (нажмите клавишу `C`, чтобы переключить режим курсора,

используйте клавиши HJKL для определения области выделения), затем нажмите «d», чтобы получить меню возможных действий/типов. Например, чтобы пометить диапазон как строку, используйте опцию 's' из меню. В командной строке такой же результат получается при помощи команды Cs:

```
[0x00000000]> f string_foo @ 0x800
[0x00000000]> Cs 10 @ string_foo
```

Команда Cf используется для определения структуры для участка памяти (используется тот же синтаксис, что в команде pf). Вот пример:

```
[0x7fd9f13ae630]> Cf 16 2xi foo bar
[0x7fd9f13ae630]> pd
;-- rip:
0x7fd9f13ae630 format 2xi foo bar {
0x7fd9f13ae630 [0] {
    foo : 0x7fd9f13ae630 = 0xe8e78948
    bar : 0x7fd9f13ae634 = 14696
}
0x7fd9f13ae638 [1] {
    foo : 0x7fd9f13ae638 = 0x8bc48949
    bar : 0x7fd9f13ae63c = 571928325
}
} 16
0x7fd9f13ae633    e868390000    call 0x7fd9f13b1fa0
0x7fd9f13ae638    4989c4        mov r12, rax
```

Аргумент [sz] у Cf используется для задания количества байтов, занимаемых структурой при дизассемблировании, она полностью независима от размера структуры данных, определяемой строкой задания формата. Может это кажется странным, но этому есть несколько применений. Например, можно задать и форматирование структуры, отображаемой при дизассемблировании, и по-прежнему отображать эти адреса в виде смещений и байтов. Для больших структур удобно идентифицировать только несколько полей, если нужны только они. Затем можно указать r2 отображать только эти поля, используя строку формата и специальные 'skip'-поля, а также продолжить дизассемблирование после отображения всей структуры, задав ей полный размер с помощью аргумента Sz. Использование Cf позволяет описывать сложные структуры при помощи простых односторонних командных выражений. Посмотрите инструкцию pf?.

Помните, что все команды группы C также доступны в визуальном режиме, нажав кнопку d (data conversion). В отличие от команды t команда Cf не изменяет результаты анализа. Это только инструмент визуализации.

Иногда добавление одной строки комментария бывает недостаточно, radare2 позволяет создавать ссылки на текстовые файлы. Функция реализуется командой CC, или нажатием клавиши , в визуальном режиме. В результате откроется \$EDITOR для создания нового файла, если файл существует, просто создаст ссылку. Файл будет показан в комментариях к коду:

```
[0x00003af7 11% 290 /bin/ls]> pd $r @ main+55 # 0x3af7
[0x00003af7  call sym.imp.setlocale      ;[1] ; ,(locale-help.txt) ; char *setlocale(int category, const char *loco
[0x00003afc  lea rsi, str usr share locale ; 0x179cc ; "/usr/share/locale"
[0x00003b03  lea rdi, [0x000179b2]       ; "coreutils"
[0x00003b0a  call sym.imp.bindtextdomain ;[2] ; char *bindtextdomain(char *domainname, char *dirname)
```

Найдите ,(locale-help.txt) в комментариях выше, если теперь снова нажать , в визуальном режиме, файл откроется. Используя этот механизм, создаются длинные описания конкретных мест в дизассемблированном коде, устанавливаются ссылки на документацию, статьи по теме.

ESIL

ESIL расшифровывается как ``Evaluable Strings Intermediate Language''. Технология использует описание на языке, подобном Forth, семантики оп-кодов операций целевого процессора. Представления ESIL интерпретируются, имитируя отдельные инструкции. Команды выражения ESIL разделяются запятыми. Его виртуальная машина описывается так:

```
while ((word=haveCommand())) {
    if (word.isOperator()) {
        esilOperators[word](esil);
    } else {
        esil.push (word);
    }
    nextCommand();
}
```

Как мы видим, ESIL использует интерпретатор на основе стека, аналогичный тому, который обычно используется для калькуляторов. Есть две категории входных данных: значения и операторы. Значение просто помещается в стек, затем оператор извлекает значения (если хотите, его аргументы) из стека, выполняет операцию и помещает результаты (если они есть) обратно в стек. ESIL использует постфиксную нотацию операций, которые мы хотим выполнить.

Посмотрим пример:

```
4,esp,-=,ebp,esp,=[4]
```

Догадывается, что это такое? Если возьмем запись и преобразуем ее обратно в инфиксную, получим

```
esp -= 4  
4bytes(dword) [esp] = ebp
```

Теперь видно, что это выражение соответствует инструкции x86. `push ebp!` Круто, да? Цель проекта состоит в том, чтобы выразить большинство операций, выполняемых процессорами: двоичные арифметические операции, загрузка и сохранение ячеек памяти, обработка системных вызовов. Преобразовав инструкции в ESIL, будет видно, что делает программа во время исполнения. Технология позволяет моделировать самые загадочные архитектуры, или, если у вас нет устройства для отладки.

Использование ESIL

Визуальный режим r2 отлично подходит для использования ESIL.

Есть три переменные среды r2, управляющие формой представления процесса исполнения программы:

```
[0x00000000]> e emu.str = true
```

Переменная `asm.emu` настраивает r2, таким образом, чтобы отображалась информация ESIL. Если установлено значение `true`, справа от вашего дизассемблированного кода добавятся комментарии, показывающие изменения содержимого регистров и адресов памяти в результате исполнения соответствующей инструкции. Например, для инструкции, вычитающей значение из регистра, ESIL сообщит, каким было значение до и каким оно становится после исполнения инструкции. Теперь не нужно отслеживать, какое значение куда идет. Правда при этом приходится воспринимать сразу много информации, что иногда просто не нужно. В r2 есть хороший компромисс. Для этого есть `emu.str (asm.emu.str для версии r2 <= 2.2)`. Вместо сверхподробного вывода с каждым значением регистра, настройка добавляет к коду только действительно полезную информацию, например, строки, найденные по адресам, или вероятность того, что переход будет выполнен или нет.

Третья важная переменная - `asm.esil`. Она переключает дизассемблирование таким образом, что оно больше не показывает фактически дизассемблированные инструкции, а вместо этого печатает соответствующие выражения ESIL, описывающие, что делает инструкция. Если хотите посмотреть, как инструкции выражаются в ESIL, просто установите значение переменной `<asm.esil>` в `true`.

```
[0x00000000]> e asm.esil = true
```

В визуальном режиме также можно переключить эти варианты представления кода, просто нажав 0 .

ESIL-команды

- ``ae" : Вычислить выражение ESIL.

```
[0x00000000]> "ae 1,1,+"  
0x2  
[0x00000000]>
```

- ``aes" : Шаг ESIL.

```
[0x00000000]> aes  
[0x00000000]>10aes
```

- ``aeso" : ESIL Step Over.

```
[0x00000000]> aeso  
[0x00000000]>10aeso
```

- ``aesu" : ESIL Step Until.

```
[0x00001000]> aesu 0x1035  
ADDR BREAK  
[0x00001019]>
```

- ``ar" : Показать/задать регистр ESIL.

```
[0x00001ec7]> ar r_00 = 0x1035  
[0x00001ec7]> ar r_00  
0x00001035  
[0x00001019]>
```

Набор инструкций ESIL

Вот полный набор инструкций, используемый виртуальной машиной ESIL:

Код опе- ра- ции ESIL	Опе- ран- ды	На- звा- ние	Опе- ра- ция	Пример
TRAP	src	Trap	Trap- сигнал	*** * src interrupt 0x80, 0 src Си- стем- ный вы- зов
\$\$	src	Ад- рес ин- струк- ции	Полу- чить адрес теку- щей ин- струк- ции стек=ад- рес ин- струк- ции	
==	src,dst	Срав- не- ние	stack =(dst == src); update_eflags(dst - src)	
<	src,dst	Мень- ше (срав- нен- ие с уче- том зна- ка)	stack [0x0000000]> ``ae 1,5,<" 0x0> ``ae 5,5"0x0" =(dst < src) ;	
<=	src,dst	Мень- ше или рав- но (срав- нен- ие с уче- том зна- ка)	stack [0x0000000]> ``ae 1,5,<" 0x0> ``ae 5,5"0x1" =(dst <= src); update_eflags(dst - src)	
>	src,dst	Боль- ше (срав- нен- ие с уче- том зна- ка)	stack > ``ae 1,5,>"0x1> ``ae 5,5,>"0x0 =(dst > src) ;	

Код опе- ра- ции ESIL	Опе- ран- ды	На- звा- ние	Опе- ра- ция	Пример
>=	src,dst	Боль- ше или рав- но (срав- нение с уче- том зна- ка)	stack = (dst > src); update_eflags(dst - src) не- ниe с уче- том зна- ка)	> ``ae 1,5,>="0x1> ``ae 5,5,>="0x1
<<	src,dst	Сдвиг влево	stack = dst << src	> ``ae 1,1,<<"0x2> ``ae 2,1,<<"0x4
>>	src,dst	Сдвиг впра- во	stack = dst >> src	> ``ae 1,4,>>"0x2> ``ae 2,4,>>"0x1
<<<	src,dst	Цик- личе- ский сдвиг влево	stack=dst> ``ae 31,1,<<<"0x80000000> ``ae 32,1,<<<"0x1 ROL src	
>>>	src,dst	Цик- личе- ский сдвиг впра- во	stack=dst> ``ae 1,1,>>>"0x80000000> ``ae 32,1,>>>"0x1 ROR src	
&	src,dst	AND	stack = dst & src	> ``ae 1,1,&"0x1> ``ae 1,0,&"0x0> ``ae 0,1,&"0x0> ``ae 0,0,&"0x0
	src,dst	OR	stack = dst src	> ``ae 1,1,"0x1> ``ae 1,0,"0x1> ``ae 0,1,"0x1> ``ae 0,0,"0x0
^	src,dst	XOR	stack = dst ^src	> ``ae 1,1,^"0x0> ``ae 1,0,^"0x1> ``ae 0,1,^"0x1> ``ae 0,0,^"0x0
+	src,dst	ADD	stack = dst + src	> ``ae 3,4,+"0x7> ``ae 5,5,+"0xa
-	src,dst	SUB	stack = dst - src	> ``ae 3,4,-"0x1> ``ae 5,5,-"0x0> ``ae 4,3,-"0xffffffffffffffff
*	src,dst	MUL	stack = dst * src	> ``ae 3,4,*"0xc> ``ae 5,5,*"0x19
/	src,dst	DIV	stack = dst / src	> ``ae 2,4,"0x2> ``ae 5,5,"0x1> ``ae 5,9,"0x1
%	src,dst	MOD	stack = dst % src	> ``ae 2,4,"%0x0> ``ae 5,5,"%0x0> ``ae 5,9,"%0x4
~	bits,src	SIGNEXTACK	= src sign extended	> ``ae 8,0x80,~"0xfffffffffffff80

Код опе- ра- ции ESIL	Опе- ран- ды	На- звा- ние	Опе- ра- ция	Пример
~/	src,dst	SIGNEDstack	> ``ae 2,-4,~/"0xfffffffffffffe	
		DIV = dst /		
		src		
		(signed)		
~%	src,dst	SIGNEDstack	> ``ae 2,-5,~%"0xfffffffffffffe	
		MOD = dst		
		% src		
		(signed)		
!	src	NEG	stack > ``ae 1,!"0x0> ``ae 4,!"0x0> ``ae 0,!"0x1	
		=		
		!!!src		
++	src	INC	stack > ar r_00=0;ar r_000x00000000> ``ae r_00,++"0x1> ar r_000x00000000> ``ae 1,++"0x2	
		=		
--	src	DEC	stack > ar r_00=5;ar r_000x00000005> ``ae r_00,--"0x4> ar r_000x00000005> ``ae 5,--"0x4	
		=		
=	src,reg	EQU	reg = > ``ae 3,r_00,=!"> aer r_000x00000003> ``ae r_00,r_01,=!"> aer r_010x00000003	
		src		
:=	src,reg	сль- бое EQU	reg = > ``ae 3,r_00,:=!"> aer r_000x00000003> ``ae r_00,r_01,:=!"> aer r_010x00000003	
		without side effects		
+=	src,reg	ADD	reg = > ar r_01=5;ar r_00=0;ar r_000x00000000> ``ae r_01,r_00,+="> ar r_000x00000005> ``ae	
		eq	reg + 5,r_00,+="> ar r_000x0000000a	
		src		
-=	src,reg	SUB	reg = > ``ae r_01,r_00,-!"> ar r_000x00000004> ``ae 3,r_00,-!"> ar r_000x00000001	
		eq		
		reg -		
		src		
=	src,reg	MUL	reg = > ar r_01=3;ar r_00=5;ar r_000x00000005> ``ae r_01,r_00,!"> ar r_000x0000000f> ``ae	
		eq	reg * 2,r_00,*!"> ar r_000x0000001e	
		src		
/=	src,reg	DIV	reg = > ar r_01=3;ar r_00=6;ar r_000x00000006> ``ae r_01,r_00,/!"> ar r_000x00000002> ``ae	
		eq	reg / 1,r_00,/!"> ar r_000x00000002	
		src		
%=	src,reg	MOD	reg = > ar r_01=3;ar r_00=7;ar r_00 0x00000007 > ``ae r_01,r_00,%!"> ar r_00 0x00000001 > ar	
		eq	reg % r_00=9;ar r_00 0x00000009 > ``ae 5,r_00,%!"> ar r_00 0x00000004	
		src		
<<=	src,reg	Shift Left	reg = > ar r_00=1;ar r_01=1;ar r_010x00000001> ``ae r_00,r_01,<<!"> ar r_010x00000002> ``ae	
		eq	reg << src	
>>=	src,reg	Shift Right	reg = > ar r_00=1;ar r_01=8;ar r_010x00000008> ``ae r_00,r_01,>>!"> ar r_010x00000004> ``ae	
		eq	reg >> src	
&=	src,reg	AND	reg = > ar r_00=2;ar r_01=6;ar r_010x00000006> ``ae r_00,r_01,&"> ar r_010x00000002> ``ae	
		eq	reg & 2,r_01,&"> ar r_010x00000002	
		src		
=	src,reg	OR	reg = > ar r_00=2;ar r_01=1;ar r_010x00000001> ``ae r_00,r_01, =!"> ar r_010x00000003> ``ae	
		eq	reg 4,r_01, =!"> ar r_010x00000007	
		src		
^=	src,reg	XOR	reg = > ar r_00=2;ar r_01=0xab;ar r_010x000000ab> ``ae r_00,r_01,^!"> ar r_010x000000a9> ``ae	
		eq	reg ^ 2,r_01,^!"> ar r_010x000000ab	
		src		
++=	reg	INC	reg = > ar r_00=4;ar r_000x00000004> ``ae r_00,++!"> ar r_000x00000005	
		eq		
		reg + 1		

Код опе- ра- ции ESIL	Опе- ран- ды	На- звा- ние	Опе- ра- ция	Пример
--=	reg	DEC	reg =	> ar r_00=4;ar r_000x00000004> ``ae r_00,--=> ar r_000x00000003
		eq	reg -	
			1	
!=	reg	NOT	reg =	> ar r_00=4;ar r_000x00000004> ``ae r_00,!=""> ar r_000x00000000> ``ae r_00,!=""> ar
		eq	!reg	r_000x00000001
---	---	---	---	-----
=[][*]=[s1],[13]=[4]bk[8]			*dst=src	> ``ae 0xdeadbeef,0x10000,=[4],"> pxw 4@0x10000x00010000 0xdeadbeef> ``ae
				0x0,0x10000,=[4],"> pxw 4@0x10000x00010000 0x00000000
[][*][1][2]#4[8]	peek		stack=*size w test@0x10000>	``ae 0x10000,[4],"0x74736574> ar r_00=0x10000> ``ae
=[] =[1] #g =[4] #8]bre	SWAP	Swap	code	r_00,[4],"0x74736574
				>>
			Pоме-	SWAP
			нять	
			ме-	
			ста-	
			ми	
			два	
			верх-	
			них	
			эле-	
			мен-	
			та	
DUP	Duplicate	Дуб- лиро- вать	DUP	
		верх- ний		
		эле- мент		
		в		
		стеке		

Код	опе-	Опе-	На-	Опе-	Пример
	ра-	ран-	зви-	ра-	
	ции	ды	тие	ция	
NUM		Numeric	Если верх- ний эле- мент явля- ется ссыл- кой (реги- стро- вое имя, метка и т.д.), разы- мено- вать его и поло- жить на стек его ре- аль- ное значе- ние	NUM	
CLEAR		Clear	Очи- стить стек	CLEAR	
BREAK		Break	Оста- нав- лива- ет эму- ля- цио ESIL	BREAK	
GOTO n		Goto	Пере- ход к N-му слову ESIL	GOTO 5	

Код опе- ра- ции ESIL	Опе- ран- ды	На- звани- е	Опе- ра- ция	Пример
TODO	To Do	Оста- нав- лива- ет вы- пол- не- ние (при- чина: выра- же- ние ESIL не завер- ше- но)	TODO	

Флаги ESIL

Виртуальная машина ESIL по умолчанию предоставляет набор вспомогательных операций для вычисления флагов. Они выполняют свою задачу, сравнивая старое и новое значение операнда dst последней выполненной операции eq. На каждой операции eq (например, ==) ESIL сохраняет старое и новое значение операнда dst. Обратите внимание, что существуют также слабые операции eq (например, :=), которые не влияют на операции с флагами. Операция == влияет на операции с флагами, несмотря на то, что она не является операцией eq. Операции с флагами задаются префиксным символом \$.

z	- флаг zero (ноль) устанавливается только если результат был 0
b	- флаг borrow (заем), требует указывать из какого бита занимать (пример: 4,\$b - проверяет, можно ли занимать
c	- флаг carry (перенос), то же что выше(пример: 7,\$c - проверяет наличие переноса из бита 7)
o	- флаг overflow (переполнение)
p	- флаг parity (четность)
r	- флаг regsize (изменение размера) (asm.bits/8)
s	- флаг sign (знак)
ds	- delay slot state
jt	- jump target
js	- jump target set

Синтаксис и команды

Целевой оп-код транслируется в список выражений ESIL, разделенных запятыми.

```
xor eax, eax      ->    0, eax, =, 1, zf, =
```

Доступ к памяти определяется операциями ``скобки'':

```
mov eax, [0x80480]  ->  0x80480, [], eax, =
```

Размер данных по умолчанию определяется размером операнда назначения.

```
movb $0, 0x80480     ->  0, 0x80480, =[1]
```

Оператор ? использует значение своего аргумента, решает, следует ли вычислять выражение в фигурных скобках.

1. Значение равно нулю? -> Пропустить.
2. Значение не равно нулю? -> Вычислить.

```
cmp eax, 123  ->  123, eax, ==, $z, zf, =
jz eax        ->  zf, ?{, eax, eip, =, }
```

Запустить несколько выражений под условным выражением можно, поместив их в фигурные скобки:

```
zf, ?{, eip, esp, =[ ], eax, eip, =, $r, esp, -=, }
```

Пробелы, переводы строки и другие символы игнорируются. Итак, первое, что нужно сделать при подготовке ESIL-программы, --- удалить пробелы:

```
esil = r_str_replace (esil, " ", "", R_TRUE);
```

Системные вызовы нуждаются в особой обработке. Они обозначаются символом «\$» в начале выражения. Можно передать необязательное числовое значение - номер системного вызова. Эмулятор ESIL должен обрабатывать системные вызовы. Смотри (`r_esil_syscall`).

Порядок аргументов для неассоциативных операций

Как обсуждалось в IRC, текущая реализация работает следующим образом:

```
a,b,-      b - a  
a,b,/=    b /= a
```

Такой вариант более удобочитаем, но менее удобен для стека.

Специальные указания

NOP представлены в виде пустых строк. Как было сказано выше, прерывания помечаются командой ''.., «0x80,.». Он делегирует эмуляцию с машины ESIL callback-функции, реализующей обработчик прерывания для конкретной ОС/ядра/платформы.

Аппаратные прерывания (traps) реализованы при помощи команды TRAP. Они используются для создания исключений для недопустимых инструкций, деления на ноль, ошибки чтения памяти или любых других действий, необходимых для конкретных архитектур.

Экспресс-анализ

Вот список некоторых экспресс-проверок для извлечения информации из строки ESIL. Соответствующая информация, вероятно, будет найдена в первом выражении списка.

```
indexOf('[')      -> включает ссылки на ячейки памяти  
indexOf("["")     -> пишет в память  
indexOf("pc,=")   -> изменяет счетчик инструкций (branch, jump, call)  
indexOf("sp,=")   -> изменяет стек (что если мы нашли sp+= или sp-=?)  
indexOf("=")      -> получить src и dst  
indexOf(":")      -> впереди неизвестный esil-, raw-оп-код  
indexOf("$")       -> получает доступ к флагам виртуальной машины esil, например: $z  
indexOf("$")       -> системный вызов, например: 1,$  
indexOf("TRAP")   -> может перехватить  
indexOf('++')     -> включает итератор  
indexOf('--)      -> считает до 0  
indexOf("?{")     -> условный оператор  
equalsTo("")      -> пустая строка, вроде por (будет неправдой, если мы добавим pc+=x)
```

Общие операции: * Проверить dstreg. * Проверить srcreg. * Получить целевой адрес. * Переход? * Условие? * Вычислить.
* Системный вызов?

Флаги ЦП

Флаги ЦП обычно определяются как однобитные регистры в профиле RReg. Иногда их можно найти в типе регистра 'flg'.

Переменные

Свойства переменных виртуальной машины:

- У них нет предопределенной разрядности. Таким образом, их при необходимости, легко расширить до 128, 256 и 512 бит, например, для MMX, SSE, AVX, Neon SIMD.
- Может быть неограниченное количество переменных. Это сделано для совместимости с SSA-формами.
- Имена регистров не имеют определенного синтаксиса. Это просто строки.
- Числа могут быть указаны в любой системе счисления, поддерживаемой RNum (десятичная, шестнадцатеричная, восьмеричная, двоичная...).
- Каждый бэкенд ESIL должен иметь связанный профиль RReg для описания спецификаций регистра ESIL.

Битовые массивы

Что с ними можно сделать? А как насчет битовой арифметики, если использовать переменные вместо регистров?

Арифметика

1. ADD(``+"),
2. MUL(``*"),
3. SUB(``-"),
4. DIV(``/"),
5. MOD(``%").

Битовая арифметика

1. AND ``&",
2. OR ``|",
3. XOR ``^",
4. SHL ``<<",
5. SHR ``>>",
6. ROL ``<<<",
7. ROR ``>>>",
8. NEG ``!".

Поддержка устройств с плавающей запятой

На момент написания этой статьи ESIL еще не поддерживает FPU. Можете реализовать поддержку неподдерживаемых инструкций с помощью r2pipe. Рано или поздно мы получим надлежащую поддержку мультимедиа и операций с плавающей запятой.

Обработка префикса x86 REP в ESIL

ESIL указывает, что команды анализа потока управления должны быть в верхнем регистре. Имейте в виду, что некоторые архитектуры используют имена регистров в верхнем регистре. Соответствующий профиль регистра должен позаботиться о том, чтобы не использовать повторно ни одно из следующего:

3,SKIP	- пропустить N инструкций, создает относительные переходы GOTO вперед
3,GOTO	- переход к инструкции 3
LOOP	- псевдоним для 0,GOTO
BREAK	- закончить вычислять выражение
STACK	- вывести содержимое стека на экран
CLEAR	- очистить стек

Пример использования: rep cmpsb

```
cx,!,{,BREAK,},esi,[1],edi,[1],==,{,BREAK,},esi,++,edi,++,cx,--,0,GOTO
```

Нереализованные/необработанные инструкции

Они выражаются командой TODO. Они действуют как «BREAK», но отображают предупреждающее сообщение о том, что инструкция не реализована и не будет эмулироваться. Например:

```
fmulp ST(1), ST(0) => TODO,fmulp ST(1),ST(0)
```

Пример дизассемблирования ESIL:

```
[0x1000010f8]> e asm.esil=true
[0x1000010f8]> pd $r @ entry0
0x1000010f8      55          8,rsp,-=,rbp,rsd,=[8]
0x1000010f9      4889e5      rsp,rbp,=
0x1000010fc      4883c768    104,rdi,+=
0x100001100      4883c668    104,rsi,+=
0x100001104      5d          rsp,[8],rbp,=,8,rsd,+=
0x100001105      e950350000  0x465a,rip,= ;[1]
0x10000110a      55          8,rsd,-=,rbp,rsd,=[8]
0x10000110b      4889e5      rsp,rbp,=
0x10000110e      488d4668    rsi,104,+,rax,=
0x100001112      488d7768    rdi,104,+,rsi,=
0x100001116      4889c7      rax,rdi,=
0x100001119      5d          rsp,[8],rbp,=,8,rsd,+=
0x10000111a      e93b350000  0x465a,rip,= ;[1]
0x10000111f      55          8,rsd,-=,rbp,rsd,=[8]
0x100001120      4889e5      rsp,rbp,=
0x100001123      488b4f60    rdi,96,+, [8],rcx,=
0x100001127      4c8b4130    rcx,48,+, [8],r8,=
0x10000112b      488b5660    rsi,96,+, [8],rdx,=
0x10000112f      b801000000  1, eax,=
```

```

0x100001134 4c394230    rdx,48,+, [8],r8,==,cz,?= 
0x100001138 7f1a        sf,of,! ,^,zf,! ,&,{,0x1154,rip,=,} ;[2]
0x10000113a 7d07        of,! ,sf,^,{,0x1143,rip,} ;[3]
0x10000113c b8fffffff  0xffffffff, eax,= ; 0xffffffff
0x100001141 eb11        0x1154,rip,= ;[2]
0x100001143 488b4938   rcx,56,+, [8],rcx,=
0x100001147 48394a38   rdx,56,+, [8],rcx,==,cz,?= 

```

Интроспекция

Чтобы упростить синтаксический анализ ESIL, у нас должен быть способ представления выражений интроспекции для извлечения нужных нам данных. Например, мы можем захотеть получить целевой адрес перехода. Транслятор выражений ESIL должен предлагать API, позволяющий легко извлекать информацию путем анализа выражений.

```

> ao~esil,opcode
opcode: jmp 0x10000465a,rip,=
esil: 0x10000465a,rip,=

```

Нам нужен способ получить числовое значение `rip'. Это очень простой пример, но есть и посложнее, вроде условных. Нам нужны выражения, чтобы иметь возможность получить:

- тип кода операции,
- целевой адрес перехода,
- состояние, от которых что-то зависит,
- список измененных регистров (write),
- список использованных (accessed) регистров (read).

API HOOKS

Для эмуляции важно иметь возможность устанавливать хуки в синтаксическом анализаторе, так, чтобы можно было расширить его для реализации анализа без необходимости изменять его снова и снова. То есть каждый раз, непосредственно перед выполнением операции, вызывается пользовательский хук. Его можно использовать, например, для определения того, собирается ли RIP измениться, обновляет ли стек инструкция. Позже можно разделить этот callback на несколько и выполнить событийный анализ на основе API, который можно расширить в JavaScript следующим образом:

```

esil.on('regset', function(){..}
esil.on('syscall', function(){esil.regset('rip'

```

API содержит функции `hook_flag_read()`, `hook_execute()` и `hook_mem_read()`. Callback должен возвращать true или 1, если надо переопределить выполняемое им действие. Например, чтобы запретить чтение памяти в регионе или отменить запись в память, фактически сделав ее доступной только для чтения. Верните false или 0, если вы хотите отслеживать синтаксический анализ выражения ESIL.

Для работы других операций требуются привязки к внешним функциям. То есть к `r_ref` и `r_io`. Это должно быть определено при инициализации виртуальной машины ESIL.

- Io Get/Set Out ax, 44 44,ax,:ou
- Селекторы (cs, ds, gs...) Mov eax, ds:[ebp+8] Ebp,8,+,:ds,eax,=

Анализ данных и кода

Radare2 имеет очень богатый набор команд и параметров конфигурации для выполнения анализа данных и кода, извлекать полезную информацию из двоичного файла, такую как указатели, ссылки на строки, базовые блоки, оп-коды, адреса переходов, перекрестные ссылки и многое другое. Эти операции обрабатываются семейством команд a (analyze):

```

Usage: a[abdefFghoprxtsc] [...]
| aa[?]           анализировать все (fcns + bbs) (aa0, если не требуется переименование подпрограмм)
| a8 [hexpairs]  анализировать байты
| ab[b] [addr]   анализировать блок по заданному адресу
| abb [len]       анализировать N базовых блоков при ограничении [len] или section.size по умолчанию
| abt [addr]     найти траектории (paths) в графе функций bb с текущего смещения до заданного адреса
| ac [cycles]    анализировать, какие оп-коды могут быть запущены в [cycles]
| ad[?]          анализировать "трамплин" данных (wip)
| ad [from] [to]  анализировать указатели на данные (from-to)
| ae[?] [expr]   анализировать результат вычисления выражения из оп-кодов (смотрите ao)
| af[?]          анализировать функции
| aF              то же, что выше, но с использованием anal.depth=1
| ag[?] [options] отрисовывать граф в различных форматах
| ah[?]          запустить анализ с подсказками (с указанием размера оп-кода, ...)
| ai [addr]      информация о адресе (показать разрешения, стек, хип, ...)
| an [name] [@addr] показать/преименовать/создать флаги/функции, привязываемые к адресу
| ao[?] [len]    анализировать оп-коды (или провести их эмуляцию)
| a0[?] [len]    анализировать N инструкций в M байтах

```

```

| ap               найти прелюдию для текущего смещения
| ar[?]           как 'dr', но для виртуальной машины ESIL (регистры)
| as[?] [num]     анализировать системный вызов, используя dbg.reg
| av[?] [.]
| ax[?]          показать переменные
|                  управление ссфлками и кросс-ссылками (xrefs), смотрите также afix

```

Фактически пространство имен `a` является одним из самых больших в radare2 и позволяет управлять разными частями процедуры анализа:

- Анализ тела функций,
- Анализ ссылок на данные,
- Использование загруженных символов,
- Управление различными типами графов, таких как CFG и граф вызовов,
- Управление переменными,
- Управление типами,
- Эмуляция с помощью виртуальной машины ESIL,
- Интроспекция оп-кодов,
- Информация об объектах, например виртуальных таблиц.

Анализ кода

Анализ кода является распространенным методом, используемым для извлечения информации из ассемблерного кода. Radare2 включает различные методы анализа кода, реализованные в ядре и доступные в разных командах. Все функциональные возможности r2 доступны также при помощи API. API дает возможность реализовать новые этапы анализа, используя какой-либо язык программирования, и даже при помощи командной строки r2, скриптов в операционной системе, реализовать алгоритмы при помощи плагинов. Цель анализа --- выявить внутренние структуры данных для идентификации базовых блоков (наборов инструкций процессора, ограниченных метками и адресными переходами), вызовов функций и извлечения информации на уровне *op-кодов*.

Наиболее распространенной последовательностью команд анализа radare2 является `aa`, что расшифровывается как «*analyze all*». Это относится ко всем символам и точкам входа. Если в двоичном файле удалены (stripped) символы, нужно использовать другие команды, такие как `aaa`, `aab`, `aar`, `aac` и им подобные. Каждая команда выявляет свою информацию, и для эффективного их использования необходимо ознакомиться в ними и подобрать набор для решения конкретных задач.

```

[0x08048440]> aa
[0x08048440]> pdf @ main
    ; DATA XREF from 0x08048457 (entry0)
/ (fcn) fcn.08048648 141
|   ;-- main:
|   0x08048648  8d4c2404  lea ecx, [esp+0x4]
|   0x0804864c  83e4f0    and esp, 0xffffffff0
|   0x0804864f  ff71fc    push dword [ecx-0x4]
|   0x08048652  55         push ebp
|   ; CODE (CALL) XREF from 0x08048734 (fcn.080486e5)
|   0x08048653  89e5      mov ebp, esp
|   0x08048655  83ec28    sub esp, 0x28
|   0x08048658  894df4    mov [ebp-0xc], ecx
|   0x0804865b  895df8    mov [ebp-0x8], ebx
|   0x0804865e  8975fc    mov [ebp-0x4], esi
|   0x08048661  8b19      mov ebx, [ecx]
|   0x08048663  8b7104    mov esi, [ecx+0x4]
|   0x08048666  c744240c000. mov dword [esp+0xc], 0x0
|   0x0804866e  c7442408010. mov dword [esp+0x8], 0x1 ; 0x00000001
|   0x08048676  c7442404000. mov dword [esp+0x4], 0x0
|   0x0804867e  c7042400000. mov dword [esp], 0x0
|   0x08048685  e852fdffff call sym..imp.ptrace
|       sym..imp.ptrace(unk, unk)
|   0x0804868a  85c0      test eax, eax
,=< 0x0804868c  7911      jns 0x804869f
|   0x0804868e  c70424cf870. mov dword [esp], str.Don_tuseadebuguer_ ; 0x080487cf
|   0x08048695  e882fdffff call sym..imp.puts
|       sym..imp.puts()
|   0x0804869a  e80dfdffff call sym..imp.abort
|       sym..imp.abort()
`-> 0x0804869f  83fb02    cmp ebx, 0x2
,=< 0x080486a2  7411      je 0x80486b5
|   0x080486a4  c704240c880. mov dword [esp], str.Youmustgiveapasswordforusethisprogram_ ; 0x0804880c
|   0x080486ab  e86cfdffff call sym..imp.puts
|       sym..imp.puts()
|   0x080486b0  e8f7fcffff call sym..imp.abort
|       sym..imp.abort()
`--> 0x080486b5  8b4604    mov eax, [esi+0x4]

```

```

| 0x080486b8    890424      mov [esp], eax
| 0x080486bb    e8e5ffff    call fcn.080485a5
|   fcn.080485a5() ; fcn.080484c6+223
| 0x080486c0    b800000000  mov eax, 0x0
| 0x080486c5    8b4df4      mov ecx, [ebp-0xc]
| 0x080486c8    8b5df8      mov ebx, [ebp-0x8]
| 0x080486cb    8b75fc      mov esi, [ebp-0x4]
| 0x080486ce    89ec        mov esp, ebp
| 0x080486d0    5d          pop ebp
| 0x080486d1    8d61fc      lea esp, [ecx-0x4]
\ 0x080486d4    c3          ret

```

В этом примере анализируется весь файл (aa), а затем печатается дизассемблирование функции `main()` (pdf). Команда `aa` относится к семейству команд автоматического анализа и выполняет только самые основные этапы автоматического анализа. В `radare2` существует множество различных типов команд автоматического анализа с различной "глубиной" анализа, включая частичную эмуляцию: `aa`, `aaa`, `aab`, `aaa`, ... Существует также аналоги этих команд в виде флагов командной строки `r2: r2 -A, r2 -AA` и так далее.

Полностью автоматизированный анализ может дать противоречивые результаты. Ранее было сказано, что `radare2` предоставляет отдельные команды для конкретных этапов анализа, что позволяет контролировать процесс выявления информации об алгоритме и структурах данных. Существует большой набор конфигурационных переменных для управления результатами анализа. Их можно найти в пространстве имен `anal.*` и `emu.*`.

Анализ функций

Одной из наиболее важных команд анализа является набор команд `af`. Комбинация `af` означает "анализ функций" (`analyze function`). Команда запускает автоматический анализ заданной функции. При желании описать функцию можно и вручную.

```

[0x0000000000]> af?
Usage: af
| af ([name]) ([addr])           анализ функций, начиная с заданного смещения или с $$  

| afr ([name]) ([addr])         анализировать функции рекурсивно  

| af+ addr name [type] [diff]  ручной анализ функции (требует afb+)  

| af- [addr]                   стереть все данные анализа функции (можно задавать смещением)  

| afa                         анализировать аргументы функции в точке вызова (afal управляет переменной  

| afb+ fcnA bbA sz [j] [f] ([t]( [d])) добавить bb к функции @ fcnaddr  

| afb[?] [addr]                перечислить базовые блоки заданной функции  

| afbFc[0|1]                  скрыть/раскрыть базовый блок (переключить атрибут 'folded')  

| afB 16                      сделать текущую функцию как "основную" (thumb) (настраивается переменной asm)  

| afC[lc] ([addr])@addr       вычислить Cycles (afC) или Cyclomatic Complexity (afCc)  

| afc[?] type @addr           установить соглашение о вызове для функции  

| afd[addr]                   показать функцию + адрес для заданного смещения  

| afF[1|0|]                   скрыть/раскрыть/переключить  

| afi [addr|fcn.name]         показать информацию о функциях (более подробный afl)  

| afj [tableaddr] [count]     анализировать таблицу переходов функции  

| afl[?] [ls*] [fcn name]    перечислить функции (смещение, размер, bbs, имя) (смотрите afll)  

| afm name                   объединить две функции  

| afM name                   вывести карту функций  

| afn[?] name [addr]         переименование функции по заданному адресу (поменять и флаг)  

| afna                       предложить имя функции автоматически для текущего смещения  

| afo[?]j [fcn.name]         показать адрес функции по ее адресу или текущему смещению  

| afs[!] ([fcnsign])         показать/установить сигнатуру функции по текущему смещению (afs! использует  

| afS[stack_size]            установить размер фрейма стека для функции по текущему смещению  

| afsr [function_name] [new_type] изменить тит для заданной функции  

| aft[?]                     сопоставление и распространение (propagation) типов  

| afu addr                  изменить размер текущей функции в соответствии с заданным конечным смещением,  

| afv[absrx]?                изменить данные об аргументах, регистрах и переменных функции  

| afx                        перечислить ссылки на функцию

```

Используйте `afl` для перечисления найденных в результате анализа функций. В группе `afl` есть еще много полезных команд, таких как `aflj`, которая выводит данные о функции в формате JSON, `aflm`, представляющая функции в виде, аналогичном `makefile`. Есть также `afl=`, отображающая ASCII-арт-схему памяти функции. Остальные команды можно найти, набрав `afl?`.

Некоторые из наиболее сложных задач при выполнении анализа функций --- это слияние, обрезка и изменение ее размера. Как и в случае с другими командами анализа, есть два режима: полуавтоматический и ручной. В полуавтоматическом режиме можно использовать `afm <function name>` для объединения текущей функции с другой, указанной именем в аргументе команды, `aff` проведет повторный анализ функции после сделанных изменений или ее редактирования, `afu <address>` --- изменение размера функции, задаваемого конечным смещением, и повторный ее анализ.

Помимо перечисленных выше полуавтоматических способов редактирования / анализа функции, анализ можно проводить в ручном режиме с помощью команды `af+`, редактировать базовые блоки при помощи команд `afb`. Перед изменением базовых блоков функции рекомендуется сначала ознакомиться с уже имеющимися:

```
[0x00003ac0]> afb
```

```

0x000003ac0 0x000003b7f 01:001A 191 f 0x000003b7f
0x000003b7f 0x000003b84 00:0000 5 j 0x000003b92 f 0x000003b84
0x000003b84 0x000003b8d 00:0000 9 f 0x000003b8d
0x000003b8d 0x000003b92 00:0000 5
0x000003b92 0x000003ba8 01:0030 22 j 0x000003ba8
0x000003ba8 0x000003bf9 00:0000 81

```

Анализ функции вручную

Давайте сначала создадим двоичный файл, например:

```

int code_block()
{
    int result = 0;

    for(int i = 0; i < 10; ++i)
        result += 1;

    return result;
}

```

Затем скомпилируем его `gcc -c example.c -m32 -O0 -fno-pie`, получим объектный файл `example.o`, откроите его с помощью `radare2`. Поскольку он еще не проанализирован, команда `pdf` не покажет дизассемблирование:

```

$ r2 example.o
[0x08000034]> pdf
p: Cannot find function at 0x08000034
[0x08000034]> pd
    ;-- section..text:
    ;-- .text:
    ;-- code_block:
    ;-- eip:
0x08000034      55          push ebp           ; [01] -r-x section size 41 named .text
0x08000035      89e5        mov ebp, esp
0x08000037      83ec10      sub esp, 0x10
0x0800003a      c745f8000000. mov dword [ebp - 8], 0
0x08000041      c745fc000000. mov dword [ebp - 4], 0
,=< 0x08000048     eb08        jmp 0x80000052
.-> 0x0800004a     8345f801   add dword [ebp - 8], 1
:| 0x0800004e     8345fc01   add dword [ebp - 4], 1
:|-> 0x08000052     837dfc09   cmp dword [ebp - 4], 9
`==< 0x08000056     7ef2        jle 0x800004a
0x08000058      8b45f8      mov eax, dword [ebp - 8]
0x0800005b      c9          leave
0x0800005c      c3          ret

```

Наша цель состоит в том, чтобы вручную создать функцию со следующей структурой

Создаем функцию по смещению `0x8000034` с именем `code_block`:

```
[0x8000034]> af+ 0x8000034 code_block
```

В большинстве случаев инструкции `jmp` или `call` используется в качестве границ блоков кода.

Таким образом, диапазон первого блока составляет от `0x08000034` `push ebp` до `0x08000048` `jmp 0x8000052`, используем команду `afb+` для его задания (добавления).

```
[0x08000034]> afb+ code_block 0x8000034 0x800004a-0x8000034 0x8000052
```

Обратите внимание, что базовым синтаксисом `afb+` является `afb+ function_address block_address block_size [jump] [fail]`. Последняя инструкция этого блока содержит новый адрес (`jmp 0x8000052`), поэтому добавляем этот адрес перехода (`0x8000052`) в аргументы команды, чтобы явным образом *отразить информацию о переходе*.

Следующий блок (`0x08000052 ~ 0x08000056`) является скорее условным оператором `if`, который включает две ветви. Переход (аргумент `jmp`) на `0x800004a` произойдет, если выполнится `jle` (меньше или равно), в противном случае (аргумент `fail`) произойдет переход к следующей инструкции по адресу `0x08000058`:

```
[0x08000034]> afb+ code_block 0x8000052 0x8000058-0x8000052 0x800004a 0x8000058
```

Следуя потоку управления, создаем оставшиеся два блока (для вышеупомянутых ветвей):

```
[0x08000034]> afb+ code_block 0x800004a 0x8000052-0x800004a 0x8000052
```

```
[0x08000034]> afb+ code_block 0x8000058 0x800005d-0x8000058
```

В результате получим:

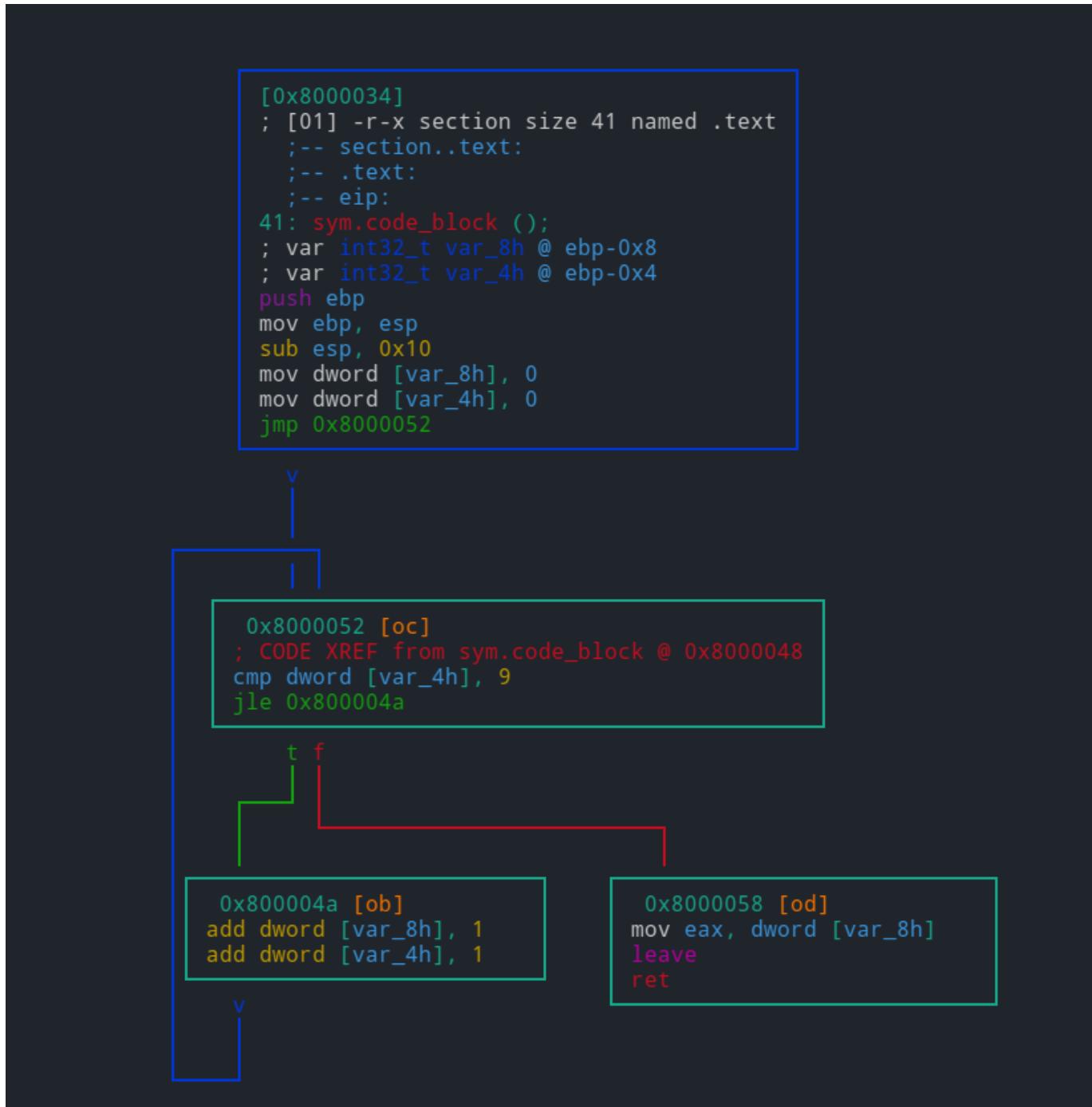


Рис. 20: analyze_one

```
[0x08000034]> afb
0x08000034 0x0800004a 00:0000 22 j 0x08000052
0x0800004a 0x08000052 00:0000 8 j 0x08000052
0x08000052 0x08000058 00:0000 6 j 0x0800004a f 0x08000058
0x08000058 0x0800005d 00:0000 5
[0x08000034]> vv
```

```

[0x8000034]
; [01] -r-x section size 41 named .text
;-- section..text:
;-- .text:
;-- code_block:
;-- eip:
41: code_block ();
; var int32_t var_8h @ ebp-0x8
; var int32_t var_4h @ ebp-0x4
push ebp
mov ebp, esp
sub esp, 0x10
mov dword [var_8h], 0
mov dword [var_4h], 0
jmp 0x8000052

v
|
||

0x8000052 [oc]
; CODE XREF from code_block @ 0x8000048
cmp dword [var_4h], 9
jle 0x800004a

t f
| |
||

0x800004a [ob]
add dword [var_8h], 1
add dword [var_4h], 1

v
|
||

0x8000058 [od]
mov eax, dword [var_8h]
leave
ret

```

Рис. 21: handcraft_one

Есть две очень важные команды: `a fc` и `a fB`. Последняя является обязательной командой для некоторых платформ, таких как ARM. Она предоставляет способ изменения "bitness" конкретной функции, позволяя выбирать между режимами ARM и Thumb. Команда `a fc` позволяет вручную указать соглашение о вызове функций. Более подробную информацию о ее использовании смотрите в `calling_conventions`.

Рекурсивный анализ

Существует пять важных команд полуавтоматического анализа всей программы:

- `a ab` - выполнение анализа базовых блоков (алгоритм "Nucleus"),

- **aac** - анализ вызовов функций из заданной или текущей функции,
- **aaf** - анализ всех вызовов функций,
- **aar** - анализ ссылок на данные,
- **aad** - анализ указателей на адреса.

И это только общие полуавтоматические алгоритмы поиска ссылок. Radare2 обеспечивает широкий выбор инструментов ручного задания ссылок. Детальное управление информационной разметкой функции --- использование команды **ax**.

```
Usage: ax[?d-l*] # смотрите также 'afx?'
| ax          перечислить ссылки
| ax*         вывести команды radare
| ax addr [at] добавить ссылку на код, ссылающийся на addr (из текущего смещения)
| ax- [at]    очистить все ссылки и ссылки с адреса
| ax-*        очистить все ссылки
| axc addr [at] добавить ссылку на код общего вида
| axC addr [at] добавить ссылку на вызываемый код
| axg [addr]  показать граф xrefs достижения текущей функции
| axg* [addr] показать график xrefs достижения заданного адреса, используйте .axg*;aggv
| axgj [addr] показать график xrefs достижения текущей функции в формате json
| axd addr [at] добавить ссылку на данные
| axq          перечислить ссылки в простом удобном для человека формате
| axj          перечислить ссылки в формате json
| axF [flg-glob] найти ссылки на данные или код флагов
| axm addr [at] копировать ссылки на данные/код, ссылающиеся на адрес и, возможно, на текущее смещение (аргумент а
| axt [addr]  найти ссылки на данные или код на заданный адрес
| axf [addr]  найти ссылки на данные или код с заданного адреса
| axv [addr]  перечислить ссылки чтения, изменения, запуска на локальные переменные
| ax. [addr]  найти ссылки на данные или код из и с заданного адреса
| axff[j] [addr] найти ссылки на данные или код из заданной функции
| axs addr [at] добавить ссылку на строку
```

Наиболее часто используемыми командами группы **ax** являются **axt** и **axf**, особенно в составе различных сценариев r2pipe. Допустим, мы видим строку в разделе данных или кода и хотим найти все места ссылок на нее, используем **axt**:

```
[0x0001783a]> pd 2
;-- str.02x:
; STRING XREF from 0x00005de0 (sub.strlen_d50)
; CODE XREF from 0x00017838 (str.._s_s_s + 7)
0x0001783a     .string "%%%02x" ; len=7
;-- str.src_ls.c:
; STRING XREF from 0x0000541b (sub.free_b04)
; STRING XREF from 0x0000543a (sub._assert_fail_41f + 27)
; STRING XREF from 0x00005459 (sub._assert_fail_41f + 58)
; STRING XREF from 0x00005f9e (sub._setjmp_e30)
; CODE XREF from 0x0001783f (str.02x + 5)
0x00017841     .string "src/ls.c" ; len=9
[0x0001783a]> axt
sub.strlen_d50 0x5de0 [STRING] lea rcx, str.02x
(nofunc) 0x17838 [CODE] jae str.02x
```

Есть также несколько полезных команд в подгруппе **axt**. Команда **axtg** - выполнения перечня команд radare2, анализирующих программные объекты, ссылающиеся на функцию, используя информацию в XREF).

```
[0x08048320]> s main
[0x080483e0]> axtg
agn 0x8048337 "entry0 + 23"
agn 0x80483e0 "main"
age 0x8048337 0x80483e0
```

Используйте **axt*** для просмотра команд radare2 и установки флагов на соответствующие адреса в XREF. Также в группе **ax** находится **axg**, которая находит путь между двумя точками в файле, показывая график XREF для достижения смещения или функции, например:

```
:> axg sym.imp.printf
- 0x08048a5c fcn 0x08048a5c sym.imp.printf
- 0x080483e5 fcn 0x080483e0 main
- 0x080483e0 fcn 0x080483e0 main
- 0x08048337 fcn 0x08048320 entry0
- 0x08048425 fcn 0x080483e0 main
```

Используйте **axg*** для генерации команд radare2, которые помогут вам создавать графики с использованием команд **agn** и **age** в соответствии с XREF. Помимо предопределенных алгоритмов для идентификации функций есть способ указать ``прелюдию'' (prelude) функции с параметром конфигурации **anal.prelude**. Например, если **anal.prelude = 0x554889e5** задает прелюдию на платформе x86_64:

```
push rbp
mov rbp, rsp
```

Она должна быть задана *перед* тем, как запускать команды анализа.

Конфигурация

Radare2 позволяет изменять поведение практически любых этапов анализа и поведение команд. Существуют различные типы параметров конфигурации:

- Управление потоком,
- Управление основными блоками,
- Элемент управления ссылками,
- Ввод-вывод, диапазоны,
- Управление анализом таблиц переходов,
- Конкретные параметры платформы/процессора.

Конфигурация потока управления

Двумя наиболее часто используемыми вариантами изменения поведения анализа потока управления в radare2 являются: anal.hasext и anal.jmp.after. Первая настройка заставляет radare2 продолжить анализ после окончания функции, даже если следующий кусок кода не вызывается откуда-либо, --- это анализ всех доступных функций. Вторая настройка заставляет radare2 продолжать анализ после безусловных переходов, найденных в коде тела функции.

В дополнение к ним anal.jmp.indir управляет отслеживанием косвенных переходов; anal.pushret - анализа последовательностей push ...; ret; anal.nopskip для пропуска NOP-последовательности в начале функции.

На данный момент radare2 также позволяет изменять максимальный размер базового блока при помощи переменной anal.bb.maxsize. Значение по умолчанию подходит к большинству случаев использования, полезно увеличивать его при работе с обfuscированным кодом, кодом в который внесены незначащие инструкции, но значительно усложняющие анализ кода тела функции. Некоторые из настроек управления базовыми блоками могут быть изменены в будущем в пользу более автоматизированных способов их оценки.

Для некоторых необычных двоичных файлов или архитектур есть опция anal.noncode. По умолчанию Radare2 не пытается анализировать разделы данных в виде кода. Но в некоторых случаях -- вредоносное ПО, упакованные двоичные файлы, двоичные файлы для встраиваемых систем --- требуют проводить такой анализ.

Управление ссылками

Управление ссылками --- наиболее важные настройки, кардинально меняющие результаты анализа. Некоторые настройки можно отключить для экономии времени и памяти при анализе больших двоичных файлов.

- analjmp.ref - разрешить создание ссылок на безусловные переходы,
- analjmp.cref - то же самое, но для условных переходов,
- anal.datarefs - отслеживание ссылок на данные в коде,
- anal.refstr - поиск строк в ссылках на данные,
- anal.strings - поиск строк и создание ссылок.

Флаг управления ссылками на строки по умолчанию отключен, так как увеличивает время анализа.

Диапазоны анализа

Настройки:

- anal.limits позволяет использовать пределы диапазона для операций анализа,
- anal.from - начальный адрес предельного диапазона,
- anal.to - соответствующий конец предельного диапазона.
- anal.in - границы поиска для анализа. Вы можете установить его на io.maps, io.sections.exec, dbg.maps и др. Например:
 - Чтобы проанализировать конкретную карту памяти с помощью anal.from и anal.to, установите anal.in = dbg.maps.
 - Для анализа в заданных границах, установите anal.from и anal.to, затем установите anal.in=range.
 - Для анализа в текущем отображаемом сегменте или разделе можно установить anal.in=bin.segment или anal.in=bin.section соответственно.
 - Для анализа в текущей карте памяти укажите anal.in=dbg.map.
 - Задать anal.in=dbg.stack или anal.in=dbg.heap для анализа в стеке или куче.
 - Для анализа в текущей функции или базовом блоке можно указать anal.in=anal.fcn или anal.in=anal.bb.

Полный список настроек доступен при помощи e anal.in=??.

Таблицы переходов

Таблицы переходов являются одной из самых сложных целей в двоичном реверс-ингенингге. Их есть сотни различных типов, конечный результат зависит от компилятора/компоновщика и этапов оптимизации LTO. Radare2 позволяет включать

некоторые экспериментальные алгоритмы обнаружения таблиц переходов с использованием anal.jmp.tbl. Наиболее сложные этапы анализа используют алгоритмы анализа адресов, упакованных в таблицы переходов. Этапы можно включить в перечень процедур, запускаемых по умолчанию для каждой поддерживаемой платформы. Есть еще две настройки, влияющие на результаты анализа таблиц переходов:

- anal.jmp.indir - отслеживать непрямые переходы, некоторые таблицы переходов опираются на них,
- anal.datarefs - отслеживать ссылки на данные, некоторые таблицы переходов используют их.

Элементы управления, специфичные для платформы

Существует две распространенные проблемы при анализе встроенных архитектур: обнаружение ARM/Thumb и значения MIPS GP. В случае двоичных файлов ARM radare2 поддерживает автоматическое обнаружение переключателей режима ARM/Thumb, но надо иметь в виду, что он использует частичную эмуляцию ESIL, тем самым замедляя процесс анализа. Если результаты вас не устраивают, определенные параметры функций могут быть переопределены при помощи afB.

Проблема MIPS GP еще сложнее. Известно, что значение GP может быть разным не только для всей программы, но также и для отдельных функций. Частично проблему решают настройки anal.gp и anal.gpfixed. Первый устанавливает значение GP для всей программы или конкретной функции. Второй позволяет «фиксировать» значение GP, если какой-то код захочет изменить его, всегда сбрасывая его, если факт смены обнаружен. Они в значительной степени экспериментальны и могут быть изменены в будущем в пользу более автоматизированного анализа.

Визуальный подход

Один из самых простых способов увидеть и проверить последствия изменений поведения команд анализа заключается в выполнении прокрутки в специальном визуальном режиме VV, позволяющем выполнять функции предварительного просмотра:

```
-[ functions ]-----
(a) add      (x)xrefs    (q)quit
(r) rename   (c)calls     (g)go
(d) delete   (v)variables (?)?help
>* 0x000005480 43 entry0
0x000150f0 46 sym._obstack_allocated_p
0x00014fe0 179 sym._obstack_begin_1
0x00014fc0 158 sym._obstack_begin
0x00015130 97 sym._obstack_free
0x00015000 283 sym._obstack_newchunk
0x000151a0 36 sym._obstack_memory_used
0x000033f0 6 sym.imp._ctype_toupper_loc
0x00003400 6 sym.imp._uflow
0x00003410 6 sym.imp.getenv
0x00003420 6 sym.imp.sigprocmask
0x00003430 6 sym.imp._snprintf_chk
0x00003440 6 sym.imp.raise
0x00000000 48 sym.imp.free
0x00003450 6 sym.imp.abort
0x00003460 6 sym.imp._errno_location
0x00003470 6 sym.imp.strncmp
0x00003480 6 sym.imp.localtime_r
0x00003490 6 sym.imp._exit
0x000034a0 6 sym.imp.strcpy
0x000034b0 6 sym.imp._fpending
0x000034c0 6 sym.imp.isatty

...  

Visual code review (pdf)
| (fcn) entry0 43
| entry0 ():  

|   0x000005480 xor ebp, ebp
|   0x000005482 mov r9, rdx
|   0x000005485 pop rsi
|   0x000005486 mov rdx, rsp
|   0x000005489 and rsp, 0xfffffffffffffff0
|   0x00000548d push rax
|   0x00000548e push rsp
|   0x00000548f lea r8, [0x00015e40]
|   0x000005496 lea rcx, [0x00015dd0]
|   0x00000549d lea rdi, [main] ; section..text ; 0x3ac0 ; "ANAVAUATUS\x89\xfdH\x89\xf3H\x83
|   0x0000054a4 call qword [reloc.__libc_start_main] ; [0x21efc8:8]=0
|   0x0000054aa hlt
```

Рис. 22: vv

Когда надо проверить, как изменения анализа влияют на результат в случае больших функций, вместо этого можно использовать миникарту, позволяющую видеть общий график управления на том же экране. Чтобы войти в режим миникарты введите VV, затем дважды нажмите клавишу p:

Этот режим позволяет увидеть дизассемблирование каждого базового блока в отдельности, просто перемещаясь между ними с помощью клавиши Tab.

Подсказки по анализу

Нередки случаи, когда результаты анализа не идеальны даже после того, как вы попробовали все до единого, включая параметры конфигурации. Именно здесь появляется механизм «analysis hints» radare2. Он позволяет переопределить некоторые основные свойства оп-кодов или метаданных или даже переписать всю строку оп-кода. Эти команды расположены в пространстве имен ah:

```
Usage: ah[lba-] Подсказки (hints), генерируемые в результате анализа
| ah?          показать данную подсказку
| ah? offset   показать подсказки для заданного смещения
| ah           перечислить подсказки в виде, удобном для чтения
| ah.
| ah-          перечислить подсказки в виде, удобном для чтения, начиная с текущего смещения
| ah-          удалить все подсказки
| ah- offset [size] удалить все подсказки, начиная с заданного смещения
| ah* offset   перечислить подсказки в формате команд radare
| aha ppc @ 0x42 задать архитектуру ppc для всех адресов >= 0x42 или до следующей подсказки
```

```
[0x100001200]> VV @ main (nodes 187 edges 266 zoom 0%) BB-MINI mouse:canvas-y mov-speed:5

[ 0x100001200 ]
;-- entry0:
;-- func.100001200:
;-- rip:
(fcn) main 2082
bp: 6 (vars 6, args 0)
sp: 0 (vars 0, args 0)
rg: 2 (vars 0, args 2)
    push rbp
    mov rbp, rsp
    push r15
    push r14
    push r13
    push r12
    push rbx
    sub rsp, 0x618
; arg2
    mov r15, rsi
; arg1
    mov r14d, edi
    lea rax, [local_240h]
    mov qword [local_30h], rax
    test r14d, r14d
[ga]jg 0x10000122f
```

The control flow graph (CFG) for the function main consists of four basic blocks:

- Block 122a [gd]:** Represented by a rectangle labeled "v". It has two outgoing edges: one to the top node of the entry block and another to the bottom node of block 122f [ga].
- Block 122f [ga]:** Represented by a rectangle labeled "f t". It has two outgoing edges: one to the bottom node of block 124b [gj] and another to the bottom node of block 12a9 [gg].
- Block 124b [gj]:** Represented by a rectangle labeled "f t". It has two outgoing edges: one to the bottom node of block 122a [gd] and another to the bottom node of block 12a9 [gg].
- Block 12a9 [gg]:** Represented by a rectangle labeled "f t". It has two outgoing edges: one to the bottom node of block 122f [ga] and another to the bottom node of block 124b [gj].

Each block is associated with a specific memory location and type:

- Block 122a [gd]: <@#####>
- Block 122f [ga]: f t
- Block 124b [gj]: f t
- Block 12a9 [gg]: f t

Рис. 23: vv2

```

| aha 0 @ 0x84      удалить вывод подсказок, касающихся архитектуры, для всех адресов >= 0x84 или до следующей подсказки
| ahb 16 @ 0x42     задать 16bit для всех адресов >= 0x42 или до следующей подсказки
| ahb 0 @ 0x84      отключить отображение подсказок класса "bits" для всех адресов >= 0x84 или до следующей подсказки
| ahc 0x804804     переназначить адрес call или jmp
| ahd foo a0,33    заменить мнемонику оп-кода
| ahe 3,eax,+     задать строковое отображение анализа виртуальной машины
| ahf 0x804840     переназначить fallback-адрес для call
| ahF 0x10         установить размер фрейма стека по текущему смещению
| ahh 0x804840     выделить визуально данный адрес в дизассемблировании
| ahi[?] 10        задать систему счисления для immediates (2, 8, 10, 10u, 16, i, p, S, s)
| ahj              перечислить подсказки в формате JSON
| aho call        сменить тип оп-кода (смотрите aho?) (устарело, перенесено в "ahd")
| ahp addr        установить подсказку для указателя
| ahr val         установить подсказку для возвращаемого значения функции
| ahs 4            установить размер оп-кода равным 4
| ahs jz          задать asm.syntax=jz для данного оп-кода
| aht [?] <type> пометить immediate как смещение типа (устарело, перенесено в "aho")
| ahv val         сменить поле val оп-кода (полезно при назначении размеров таблиц переходов в jmp rax)

```

Один из наиболее распространенных случаев --- установка системы счисления для чисел:

```

[0x00003d54]> ahi?
Usage: ahi [2|8|10|10u|16|bodhipSs] [@ offset]  Задание системы счисления
| ahi <base>  задать систему счисления (2, 8, 10, 16)
| ahi 10|d    задать десятичную систему счисления со знаком (10), бит знака зависит от размера операнда
| ahi 10u|du   задать десятичную систему счисления без знака (11)
| ahi b       задать двоичную систему счисления (2)
| ahi o       задать восмеричную систему счисления (8)
| ahi h       задать шестнадцатеричную систему счисления (16)
| ahi i       представлять как IP-адрес (32)
| ahi p       представлять как htons(port) (3)
| ahi S       представлять как syscall (80)
| ahi s       представлять в виде строки (1)

[0x00003d54]> pd 2
0x00003d54      0583000000  add eax, 0x83
0x00003d59      3d13010000  cmp eax, 0x113
[0x00003d54]> ahi d
[0x00003d54]> pd 2
0x00003d54      0583000000  add eax, 131
0x00003d59      3d13010000  cmp eax, 0x113
[0x00003d54]> ahi b
[0x00003d54]> pd 2
0x00003d54      0583000000  add eax, 10000011b
0x00003d59      3d13010000  cmp eax, 0x113

```

Примечательно, что некоторые этапы анализа или команды добавляют подсказки внутреннего анализа, которые можно проверить с помощью команды ah :

```

[0x00003d54]> ah
0x00003d54 - 0x00003d54 => immbase=2
[0x00003d54]> ah*
ahi 2 @ 0x3d54

```

Иногда нам нужно переопределить адрес перехода или вызова, например, в случае замысловатого перемещения, неизвестного radare2. Текущую аналитическую информацию о конкретном оп-коде можно проверить с помощью команды ao. Используем команду ahc для выполнения такого изменения:

```

[0x00003cee]> pd 2
0x00003cee      e83d080100  call sub.__errno_location_530
0x00003cf3      85c0        test eax, eax
[0x00003cee]> ao
address: 0x3cee
opcode: call 0x14530
mnemonic: call
prefix: 
id: 56
bytes: e83d080100
refptr: 0
size: 5
sign: false
type: call
cycles: 3
esil: 83248,rip,8,rsp,-=,rsp,=[],rip,=
jump: 0x00014530
direction: exec
fail: 0x00003cf3
stack: null
family: cpu
stackop: null

```

```
[0x00003cee]> ahc 0x5382
[0x00003cee]> pd 2
0x00003cee      e83d080100    call sub._errno_location_530
0x00003cf3      85c0        test eax, eax
[0x00003cee]> ao
address: 0x3cee
opcode: call 0x14530
mnemonic: call
prefix: 0
id: 56
bytes: e83d080100
refptr: 0
size: 5
sign: false
type: call
cycles: 3
esil: 83248,rip,8,rsp,-=,rsp,=[],rip,=
jump: 0x00005382
direction: exec
fail: 0x00003cf3
stack: null
family: cpu
stackop: null
[0x00003cee]> ah
0x00003cee - 0x00003cee => jump: 0x5382
```

Как видите, несмотря на неизменный вид дизассемблирования, адрес перехода в оп-коде был изменен (настройка `jump`).

Если что-то из ранее описанного не помогло, можно просто переопределить показанное дизассемблирование чем-либо, что вам нравиться:

```
[0x00003d54]> pd 2
0x00003d54      0583000000    add eax, 100000011b
0x00003d59      3d13010000    cmp eax, 0x113
[0x00003d54]> "ahd myopcode bla, foo"
[0x00003d54]> pd 2
0x00003d54      myopcode bla, foo
0x00003d55      830000        add dword [rax], 0
```

Управление переменными

Radare2 позволяет работать с локальными переменными независимо от их местоположения, в стеке они или в регистрах. Автоматический анализ переменных включен по умолчанию, но его можно отключить с помощью настройки `anal.vars`.

Основные команды управления переменными находятся в пространстве имен `afv`:

<code>Usage: afv [rbs]</code>	
<code> afv*</code>	вывести команды r2 для добавления аргумента/локальной переменной в пространство флагов
<code> afv-([name])</code>	удалить все или заданные переменные
<code> afv=</code>	перечислить переменные функции и ее аргументы с ссылками на дизассемблирование
<code> afva</code>	анализировать аргументы и локальные переменные функции
<code> afvb[?]</code>	изменять аргументы и локальные переменные, доступные по смещению в регистре bp
<code> afvd name</code>	вывести команды r2 для отображения значений аргументов и локальных переменных в отладчике
<code> afvf</code>	показать переменные во фрейме стека, задаваемого регистром bp
<code> afvn [new_name] ([old_name])</code>	переименовать аргумент или локальную переменную
<code> afvr[?]</code>	изменить аргумент, соответствующий регистру
<code> afvR [varname]</code>	перечислить адреса, где к переменной получают доступ (ЧТЕНИЕ)
<code> afvs[?]</code>	управление аргументами и локальными переменными, адрессируемыми при помощи регистра
<code> afvt [name] [new_type]</code>	сменить тип для заданного аргумента или переменной
<code> afvW [varname]</code>	перечислить адреса, где к переменным получают доступ (ИЗМЕНЕНИЕ)
<code> afvx</code>	показать перекрестные ссылки на переменные функции (то же самое, что и <code>afvR+afvW</code>)

Команды `afvr`, `afvb` и `afvs` унифицированы, но позволяют манипулировать аргументами и локальными переменными, хранящимися в регистрах, в стеке по смещениям в BP/FP, индексируемыми при помощи SP. Для группы команд `afvr` есть богатый набор операций над регистровыми переменными:

<code> Usage: afvr [reg] [type] [name]</code>	
<code> afvr</code>	перечислить переменные, соответствующие регистрам
<code> afvr*</code>	то же, что и <code>afvr</code> , но в виде команд <code>r2</code>
<code> afvr [reg] [name] ([type])</code>	определить аргумент, соответствующий регистру
<code> afvrj</code>	показать перечень регистровых переменных в формате JSON
<code> afvr- [name]</code>	удалить регистровый аргумент
<code> afvrg [reg] [addr]</code>	определить get-интерфейс аргумента
<code> afvrs [reg] [addr]</code>	определить set-интерфейс аргумента

Как и для многих других программных объектов, обнаружение переменных происходит автоматически, результаты можно изменить с помощью команд управления аргументами/локальными переменными. Этот вид анализа в значительной степени полагается на предварительно загруженные прототипы функций и соглашение о вызовах, таким образом загружая

символы из отладочной информации, можно значительно улучшить результаты анализа. Более того, внеся изменения можно перезапустить анализ переменных с помощью команды `a fva`. Довольно часто анализ переменных сопровождается анализом типов, ознакомьтесь с инструкциями команды `afta`.

Самый важный аспект реверс-инжениринга --- именование программных объектов и сущностей. После переименования переменных в текстах дизассемблирования автоматически меняются все ссылки на эти переменные. Переименование аргументов и локальных переменных осуществляется при помощи команды `a fvn`. Удаление делается при помощи команды `a fv-`.

Как упоминалось ранее, анализ в значительной степени зависит от информации о типах переменных. Следующая очень важная команда - `a fv t`, она позволяет изменять тип переменной:

```
[0x00003b92]> afvs
var int local_8h @ rsp+0x8
var int local_10h @ rsp+0x10
var int local_28h @ rsp+0x28
var int local_30h @ rsp+0x30
var int local_32h @ rsp+0x32
var int local_38h @ rsp+0x38
var int local_45h @ rsp+0x45
var int local_46h @ rsp+0x46
var int local_47h @ rsp+0x47
var int local_48h @ rsp+0x48
[0x00003b92]> afvt local_10h char*
[0x00003b92]> afvs
var int local_8h @ rsp+0x8
var char* local_10h @ rsp+0x10
var int local_28h @ rsp+0x28
var int local_30h @ rsp+0x30
var int local_32h @ rsp+0x32
var int local_38h @ rsp+0x38
var int local_45h @ rsp+0x45
var int local_46h @ rsp+0x46
var int local_47h @ rsp+0x47
var int local_48h @ rsp+0x48
```

Пока еще редко используемый инструмент, находящаяся в стадии интенсивной разработки, --- построение списков переменных, которые читаются и изменяются. Команда `a fVR` перечисляет переменные, которые читаются, а `a fvW` - изменяются. Обе команды предоставляют список адресов, где эти операции выполняются:

```
[0x00003b92]> afvR
local_48h 0x48ee
local_30h 0x3c93,0x520b,0x52ea,0x532c,0x5400,0x3cfb
local_10h 0x4b53,0x5225,0x53bd,0x50cc
local_8h 0x4d40,0x4d99,0x5221,0x53b9,0x50c8,0x4620
local_28h 0x503a,0x51d8,0x51fa,0x52d3,0x531b
local_38h
local_45h 0x50a1
local_47h
local_46h
local_32h 0x3cb1
[0x00003b92]> afvW
local_48h 0x3adf
local_30h 0x3d3e,0x4868,0x5030
local_10h 0x3d0e,0x5035
local_8h 0x3d13,0x4d39,0x5025
local_28h 0x4d00,0x52dc,0x53af,0x5060,0x507a,0x508b
local_38h 0x486d
local_45h 0x5014,0x5068
local_47h 0x501b
local_46h 0x5083
local_32h
[0x00003b92]>
```

Вывод типа

Вывод типа для локальных переменных и аргументов хорошо интегрирован с командой `afta`. Посмотрим пример с простым бинарным файлом `hello_world`

```
[0x000007aa]> pdf
|      ;-- main:
/ (fcn) sym.main 157
| sym.main ();
| ; var int local_20h @ rbp-0x20
| ; var int local_1ch @ rbp-0x1c
| ; var int local_18h @ rbp-0x18
| ; var int local_10h @ rbp-0x10
| ; var int local_8h @ rbp-0x8
```

```

| ; DATA XREF from entry0 (0x6bd)
| 0x000007aa push rbp
| 0x000007ab mov rbp, rsp
| 0x000007ae sub rsp, 0x20
| 0x000007b2 lea rax, str.Hello      ; 0x8d4 ; "Hello"
| 0x000007b9 mov qword [local_18h], rax
| 0x000007bd lea rax, str.r2_folks ; 0x8da ; "r2-folks"
| 0x000007c4 mov qword [local_10h], rax
| 0x000007c8 mov rax, qword [local_18h]
| 0x000007cc mov rdi, rax
| 0x000007cf call sym.imp.strlen    ; size_t strlen(const char *s)

```

- После применения `afta`:

```

[0x000007aa]> afta
[0x000007aa]> pdf
| ;-- main:
| ;-- rip:
/ (fcn) sym.main 157
sym.main ();
| ; var size_t local_20h @ rbp-0x20
| ; var size_t size @ rbp-0x1c
| ; var char *src @ rbp-0x18
| ; var char *s2 @ rbp-0x10
| ; var char *dest @ rbp-0x8
| ; DATA XREF from entry0 (0x6bd)
0x000007aa push rbp
0x000007ab mov rbp, rsp
0x000007ae sub rsp, 0x20
0x000007b2 lea rax, str.Hello      ; 0x8d4 ; "Hello"
0x000007b9 mov qword [src], rax
0x000007bd lea rax, str.r2_folks ; 0x8da ; "r2-folks"
0x000007c4 mov qword [s2], rax
0x000007c8 mov rax, qword [src]
0x000007cc mov rdi, rax          ; const char *s
0x000007cf call sym.imp.strlen   ; size_t strlen(const char *s)

```

Также информация о типе извлекается из форматных строк, таких как `printf("fmt: %s, %u, %d ...")`, спецификации форматов находятся в `anal/d/spec.sdb`.

Можно создать новый профиль для указания набора символов определения формата в зависимости от различных библиотек/операционных систем/языков программирования, например:

```
win=spec
spec.win.u32=unsigned int
```

Затем изменить спецификацию по умолчанию на вновь созданную, используя переменную конфигурации `e anal.spec = win`

Для получения дополнительной информации о поддержке примитивных и определяемых пользователем типов в radare2 читайте раздел типы.

Типы данных

Radare2 поддерживает описание типов данных при помощи синтаксиса языка C. Описания синтаксически анализируются, сохраняются во внутренний SDB и доступны для интроспекции при помощи команды `k`. Синтаксис описаний совместим с C11. Большинство связанных с этой функцией команд находятся в пространстве имен `t`:

```

[0x00000000]> t?
| Usage: t  # команды задания типов (подсистемы cparse)
| t          Перечислить все загруженные типы
| tj         Перечислить все загруженные типы в формате json
| t <type>   Показать тип в синтаксисе 'rf'
| t*        Показать информацию о типах в виде команд r2
| t- <name>  Удалить типы по их именам
| t-*       Удалить все типы
| tail [filename] Вывести последнюю часть файла
| tc [type.name] Перечислить все/заданные типы в формате C
| te[?]     Перечислить все загруженные enum-ы
| td[?] <string> Загрузить типы из строк
| tf        Перечислить все загруженные функции
| tk <sdb-query> Выполнить запрос sdb
| tl[?]    Показать/привязать тип по/к адресу
| tn[?] [-][addr] Управление атрибутами и метками noreturn-функции
| to -      Открыть cfg.editor для загрузки типов
| to <path>  Загрузить типы из заголовочных файлов C
| toe [type.name] Открыть cfg.editor для редактирования типов
| tos <path>  Загрузить типы из базы данных sdb, представленной в текстовом виде

```

```

| tp <type> [addr|varname]    Преобразовать данные по <addr-есу> в <type> и распечатать результат (XXX: тип может содер-
| tpv <type> @ [value]        Показать смещение, представленное в формате, соответствующем заданному типу
| tpx <type> <hexpairs>      Показать значение для типа для заданной последовательности байтов (XXX: тип может содер-
| ts[?]                      Показать загруженные типы-структур
| tu[?]                      Показать загруженные типы-юнионы
| tx[f?]                     Распечатать xrefs
| tt[?]                      Перечислить все загруженные typedef-ы

```

Обратите внимание, что базовые (атомарные) типы не соответствуют стандарту C --- нет `char`, `_Bool` и `short`. Эти типы могут быть разными на разных платформах, radare2 использует *строго определенные* типы, такие как `int8_t` или `uint64_t`, и конвертирует `int` в `int32_t` или в `int64_t` в зависимости от бинарной или отлаживаемой платформы/компилятора. Базовые типы перечисляются с помощью команды `t`. Для типов структур нужно использовать `ts`, для юнионов --- `tu`, а для перечислений --- `te`.

```
[0x00000000]> t
char
char *
double
float
gid_t
int
int16_t
int32_t
int64_t
int8_t
long
long long
pid_t
short
size_t
uid_t
uint16_t
uint32_t
uint64_t
uint8_t
unsigned char
unsigned int
unsigned short
void *
```

Загрузка типов

Есть три способа определить новый тип:

- Непосредственно из командной строки с помощью команды `td`,
- Из файла с помощью `to <имя файла>`,
- Открыть `$EDITOR` для ввода определений с помощью `to -`.

```
[0x00000000]> "td struct foo {char* a; int b;}"
[0x00000000]> cat ~/radare2-regressions/bins/headers/s3.h
struct S1 {
    int x[3];
    int y[4];
    int z;
};
[0x00000000]> to ~/radare2-regressions/bins/headers/s3.h
[0x00000000]> ts
foo
S1
```

Radare2 позволяет организовать папку для хранения описаний типов и включений.

```
[0x00000000]> e? dir.types
dir.types: Месторасположение по умолчанию, где надо искать файлы формата cparse
[0x00000000]> e dir.types
/usr/include
```

Распечатка структур

Команда `ts` преобразует описание типа C (или, если быть точным, представление SDB) в последовательность команд `pf`. Смотрите подробнее о печати структур. Команда `tp` использует строку `pf` для печати всех полей типа по текущему смещению/данному адресу:

```
[0x00000000]> "td struct foo {char* a; int b;}"
[0x00000000]> wx 68656c6c6f000c000000
[0x00000000]> wz world @ 0x00000010 ; wx 17 @ 0x00000016
[0x00000000]> px
[0x00000000]> ts foo
```

```

pf zd a b
[0x00000000]> tp foo
a : 0x00000000 = "hello"
b : 0x00000006 = 12
[0x00000000]> tp foo @ 0x00000010
a : 0x00000010 = "world"
b : 0x00000016 = 23

```

Можно также заполнить структуру своими данными и распечатать ее, используя команду `tpx`.

```

[0x00000000]> tpx foo 414243440010000000
a : 0x00000000 = "ABCD"
b : 0x00000005 = 16

```

Типы ссылок

Команда `tp` просто выполняет приведение типов. Но если надо связать какой-то адрес или переменную с выбранным типом нужно использовать команду `tl` для сохранения этого отношения в SDB.

```

[0x000051c0]> tl S1 = 0x51cf
[0x000051c0]> tll
(S1)
x : 0x000051cf = [ 2315619660, 1207959810, 34803085 ]
y : 0x000051db = [ 2370306049, 4293315645, 3860201471, 4093649307 ]
z : 0x000051eb = 4464399

```

Причем ссылка будет показана в дизассемблированном выводе или в визуальном режиме:

```

[0x000051c0 15% 300 /bin/ls]> pd $r @ entry0
;-- entry0:
0x000051c0    xor ebp, ebp
0x000051c2    mov r9, rdx
0x000051c5    pop rsi
0x000051c6    mov rdx, rsp
0x000051c9    and rsp, 0xfffffffffffffff0
0x000051cd    push rax
0x000051ce    push rsp
(S1)
x : 0x000051cf = [ 2315619660, 1207959810, 34803085 ]
y : 0x000051db = [ 2370306049, 4293315645, 3860201471, 4093649307 ]
z : 0x000051eb = 4464399
0x000051f0    lea rdi, loc._edata      ; 0x21f248
0x000051f7    push rbp
0x000051f8    lea rax, loc._edata      ; 0x21f248
0x000051ff    cmp rax, rdi
0x00005202    mov rbp, rsp

```

После того как структура связана, radare2 пытается распространить смещение структуры внутри функции, которой принадлежит текущее смещение. Чтобы запустить этот анализ во всей программе или в любых целевых функциях после того, как все структуры будут связаны, --- команда `aat`:

```

[0x00000000]> aa?
| aat [fcn]           Анализировать все/заданную функцию, преобразовать immediate-ы в смещения связанных структур (см.

```

Иногда эмуляция может быть неточной, например, как показано ниже:

```

|0x000006da  push rbp
|0x000006db  mov rbp, rsp
|0x000006de  sub rsp, 0x10
|0x000006e2  mov edi, 0x20
|0x000006e7  call sym.imp.malloc      ; "@"
|0x000006ec  mov qword [local_8h], rax
|0x000006f0  mov rax, qword [local_8h]

```

Возвращаемое значение `malloc` может отличаться в разных запусках эмуляции, нужно установить подсказку (`hint`) для возвращаемого значения вручную, используя команду `ahr`, потом снова запустить `tl` или `aat`.

```

[0x000006da]> ah?
| ahr val           установить подсказку для возвращаемого значения функции

```

Structure Immediates

Есть еще один важный аспект использования типов в Radare2 --- использование `aht`, что позволяет изменять `immediate`-значение в коде операции на смещение структуры. Рассмотрим простой пример относительной (RSI) адресации:

```

[0x000052f0]> pd 1
0x000052f0  mov rax, qword [rsi + 8]    ; [0x8:8]=0

```

Здесь 8 - некоторое смещение в памяти, где `rsi`, вероятно, содержит указатель структуры. Представьте, что у нас есть следующие структуры

```
[0x0000052f0]> "td struct ms { char b[8]; int member1; int member2; };"  
[0x0000052f0]> "td struct ms1 { uint64_t a; int member1; };"  
[0x0000052f0]> "td struct ms2 { uint16_t a; int64_t b; int member1; };"
```

Теперь нам нужно установить правильное смещение поля структуры вместо 8-ки, упомянутой в инструкции. Сначала нужно перечислить доступные типы, соответствующие этому смещению:

```
[0x0000052f0]> ahts 8  
ms.member1  
ms1.member1
```

Обратите внимание, что `ms2` не указан, так как в нем нет элементов со смещением 8. После перечисления доступных вариантов связываем их с выбранным смещением:

```
[0x0000052f0]> aht ms1.member1  
[0x0000052f0]> pd 1  
0x0000052f0      488b4608      mov rax, qword [rsi + ms1.member1] ; [0x8:8]=0
```

Управление перечислениями

- Печать всех полей в перечислении --- команда `te`:

```
[0x00000000]> "td enum Foo {COW=1,BAR=2};"  
[0x00000000]> te Foo  
COW = 0x1  
BAR = 0x2
```

- Поиск соответствующего члена перечисления для заданного битового поля и наоборот:

```
[0x00000000]> te Foo 0x1  
COW  
[0x00000000]> tbe Foo COW  
0x1
```

Внутреннее представление

Чтобы увидеть внутреннее представление типов, используйте команду `tk`:

```
[0x000051c0]> tk~S1  
S1=struct  
struct.S1=x,y,z  
struct.S1.x=int32_t,0,3  
struct.S1.x.meta=4  
struct.S1.y=int32_t,12,4  
struct.S1.y.meta=4  
struct.S1.z=int32_t,28,0  
struct.S1.z.meta=0  
[0x000051c0]>
```

Определение примитивных типов требует понимания основных форматов `pf`, вот список спецификаторов формата `pf??`:

format	explanation
b	byte (unsigned)
c	char (signed byte)
d	0x%08x шестнадцатеричное значение (4 байта)
f	значение float (4 байта)
i	%i значение integer (4 байта)
o	0x%08o восьмеричное значение (4 байта)
p	указатель (2, 4 или 8 байт)
q	quadword (8 байт)
s	32bit-указатель на строку (4 байта)
S	64bit-указатель на строку (8 байт)
t	UNIX timestamp (4 байта)
T	показать первых десять (Ten) байт буфера
u	uleb128 (variable length)
w	word (2 байта, unsigned short в шестнадцатеричном виде)
x	0x%08x шестнадцатеричное значение и флаг (fd @ addr)
X	показать последовательность шестнадцатеричных кодов
z	строка, заканчивающаяся \0
Z	wide-строка, заканчивающаяся \0

Есть только три обязательных ключа для определения базовых типов:

```
X=type  
type.X=format_specifier  
type.X.size=size_in_bits
```

Например, определим `UNIT` согласно спецификации в документации Microsoft. Тип `UINT` - это лишь эквивалент стандартному типу C `unsigned int` (или `uint32_t` в терминах системы TCC). Тип определяется так:

```
UINT=type  
type.UINT=d  
type.UINT.size=32
```

Определение некоторого типа можно дополнить необязательной строкой:

```
X.type.pointto=Y
```

Она используется только в случае задания указателя, когда `type.X=p`. Хорошим примером выступает определение типа `LPFILETIME`, указателя на структуру `_FILETIME`. Предполагая, что мы работаем только с 32-разрядной машиной Windows, структура будет определена следующим образом:

```
LPFILETIME=type  
type.LPFILETIME=p  
type.LPFILETIME.size=32  
type.LPFILETIME.pointto=_FILETIME
```

Последнее поле не является обязательным, поскольку иногда в структурах данных внутренности бывают скрытыми от глаз разработчиков согласно коммерческим лицензиям (proprietary), а у нас может не быть информации для них четкого определения.

Есть еще одно необязательное определение:

```
type.UINT.meta=4
```

Эта форма предназначена для интеграции с синтаксическим анализатором С и содержит информацию о классе типов: *целочисленный размер, знаковый/беззнаковый* и т.д.

Структуры

Вот основные ключи для структур (всего два элемента):

```
X=struct  
struct.X=a,b  
struct.X.a=a_type,a_offset,a_number_of_elements  
struct.X.b=b_type,b_offset,b_number_of_elements
```

Первая строка используется для определения структуры, называемой `X`, вторая строка определяет элементы `X` как значения, разделенные запятыми. После этого просто определяем информацию о каждом элементе.

Приведем примеры. Предположим в исходном коде есть такая структура:

```
struct _FILETIME {  
    DWORD dwLowDateTime;  
    DWORD dwHighDateTime;  
}
```

Полагая, что `DWORD` определено, структура будет выглядеть так:

```
_FILETIME=struct  
struct._FILETIME=dwLowDateTime,dwHighDateTime  
struct._FILETIME.dwLowDateTime=DWORD,0,0  
struct._FILETIME.dwHighDateTime=DWORD,4,0
```

Обратите внимание, что поле количества элементов используется только в случае массивов, по умолчанию оно равно нулю.

Юнионы

Юнионы определяются точно так же, как структуры, с той лишь разницей, что вы заменяете слово `struct` на `union`.

Прототипы функций

Представление прототипов функций является наиболее подробным и наиболее важным среди всех определений. На самом деле, это то, что используется непосредственно для сопоставления типов.

```
X=func  
func.X.args=NumberOfArgs  
func.x.arg0=Arg_type,arg_name  
. . .  
func.X.ret=Return_type  
func.X.cc=calling_convention
```

Все и так понятно. Определим прототип для strncasecmp в качестве примера для архитектуры x86 и машин Linux. Согласно справке в man функция strncasecmp определяется следующим образом:

```
int strncasecmp(const char *s1, const char *s2, size_t n);
```

При преобразовании в представление sdb ее прототип будет выглядеть следующим образом:

```
strncasecmp=func
func.strncasecmp.args=3
func.strncasecmp.arg0=char *,s1
func.strncasecmp.arg1=char *,s2
func.strncasecmp.arg2=size_t,n
func.strncasecmp.ret=int
func.strncasecmp.cc=cdecl
```

Обратите внимание, что часть .cc является необязательной, и если она не используется, то используется соглашение о вызовах по умолчанию для вашей целевой архитектуры. Есть еще один дополнительный необязательный ключ

```
func.x.noreturn=true/false
```

Этот ключ используется для обозначения функций, из которых невозможно вернуться в вызывающую подпрограмму, например `exit` и `_exit`.

Соглашения вызова функции

Radare2 использует предположения о соглашении о вызове в процессе анализа функции для распознавания формальных аргументов функции. На основе предположения производится вывод типа возвращаемого значения.

```
[0x00000000]> afc?
[Usage: afc[agl?]
| afc convention Задать вручную соглашение о вызове для текущей функции
| afc Вывести соглашение о вызове для текущей функции
| afc=([ctype]) Выбрать/показать соглашение о вызове по умолчанию
| afcr[j] Показать распределение регистров по параметрам
| afca Анализировать функцию с целью определения соглашения о вызове для текущей функции
| afcf[j] [name] Вывести тип возвращаемого значения function(arg1, arg2...),смотрите afij
| afck Перечислить детали SDB для вызова для загруженных соглашений о вызове
| afcl Перечислить все поддерживаемые соглашения о вызове
| afco path Открыть профиль sdb для соглашения о вызове из файла
| afcR Зарегистрировать телескопирование (telescoping), используя порядок соглашений о вызовах
[0x00000000]>
```

- Вывод списка всех доступных соглашений о вызовах для текущей архитектуры - команда `afcl`.

```
[0x00000000]> afcl
amd64
ms
```

- Для отображения прототипа функции стандартной библиотеки есть команда `afc f`.

```
[0x00000000]> afcf printf
int printf(const char *format)
[0x00000000]> afcf fgets
char *fgets(char *s, int size, FILE *stream)
```

Вся эта информация загружается через sdb в `/libr/anal/d/cc-[arch]-[bits].sdb`

```
default.cc=amd64
```

```
ms=cc
cc.ms.name=ms
cc.ms.arg1=rcx
cc.ms.arg2=rdx
cc.ms.arg3=r8
cc.ms.arg3=r9
cc.ms.argv=stack
cc.ms.ret=rax
```

`cc.x.argv=rax` используется для задания i-го аргумента функции, регистрации имени `rax`,

`cc.x.argv=stack` означает, что все аргументы (или остальные аргументы в случае наличия `argi` для любого i) будут храниться в стеке слева направо,

`cc.x.argv=stack_rev` то же, что и `cc.x.argv=stack`, но аргументы передаются справа налево.

Виртуальные таблицы

В radare2 есть базовая поддержка анализа виртуальных таблиц (RTTI и др.). Самое главное, прежде чем вы начнете выполнять такой анализ, надо проверить, что параметр `anal.cpp.abi` установлен правильно.

Все команды для работы с виртуальными таблицами находятся в пространстве имён `av`. В настоящее время поддержка достаточно проста, она позволяет только проверять таблицы, построенные в результате анализа.

```
|Usage: av[?jr*] Таблицы виртуальных методов C++ и RTTI
| av          поиск виртуальных таблиц в сегменте данных и печать результата
| avj         подобно av, но в виде json
| av*        подобно av, но в виде комнд r2
| avr[jaddr] попробовать странслировать RTTI по адресу виртуальной таблицы (смотрите anal.cpp.abi)
| avra[j]    поиск виртуальных таблиц и попробовать странслировать RTTI в каждой из них
```

Основные команды --- это `av` и `avr`. Команда `av` перечисляет все найденные виртуальные таблицы в открытом файле. Если вы недовольны результатом, можно попытаться проанализировать виртуальную таблицу по определенному адресу с помощью `avr`. Команда `avra` выполняет поиск и анализ всех виртуальных таблиц в бинарном файле, как это делает `r2` при открытии файла.

Системные вызовы

Radare2 позволяет вручную искать ассемблерный код, похожий на операцию системного вызова. Например, на платформе ARM они обычно представлены инструкциями `SVC`, на других архитектурах это какие-то другие варианты, например `syscall` на x86.

```
[0x0001ece0]> /ad/ svc
...
0x000187c2 # 2: svc 0x76
0x000189ea # 2: svc 0xa9
0x00018a0e # 2: svc 0x82
...
```

Обнаружение системных вызовов управляется `asm.os`, `asm.bits`, а также `asm.arch`. Для проверки настроек поддержки системных вызовов используется команда `asl`. Команда перечисляет системные вызовы, поддерживаемые вашей платформой.

```
[0x0001ece0]> asl
...
sd_softdevice_enable = 0x80.16
sd_softdevice_disable = 0x80.17
sd_softdevice_is_enabled = 0x80.18
...
```

Если настроен стек ESIL командами `aei` или `aeim`, можно использовать команду `/as` для поиска адресов детектированных системных вызовов и их перечисления.

```
[0x0001ece0]> aei
[0x0001ece0]> /as
0x000187c2 sd_ble_gap_disconnect
0x000189ea sd_ble_gatts_sys_attr_set
0x00018a0e sd_ble_gap_sec_info_reply
...
```

Для сокращения времени поиска можно ограничить диапазон поиска на сегменты кода или сегменты из `/as @e:search.in=io.maps.x`.

Используя эмуляцию ESIL в radare2 можно печатать аргументы системного вызова в дизассемблировании. Чтобы включить линейную (но очень грубую) эмуляцию, используйте переменную конфигурации `asm.emu`:

```
[0x0001ece0]> e asm.emu=true
[0x0001ece0]> s 0x000187c2
[0x000187c2]> pdf~svc
  0x000187c2  svc 0x76 ; 118 = sd_ble_gap_disconnect
[0x000187c2]>
```

В случае выполнения команды `aae` (или `aaaa`, которая вызывает `aae`) radare2 будет помещать найденные системные вызовы в специальное пространство флагов `syscall.`, что полезно для автоматизации:

```
[0x000187c2]> fs
0 0 * imports
1 0 * symbols
2 1523 * functions
3 420 * strings
4 183 * syscalls
[0x000187c2]> f~syscall
...
```

```
0x0000187c2 1 syscall.sd_ble_gap_disconnect.0
0x0000189ea 1 syscall.sd_ble_gatts_sys_attr_set
0x000018a0e 1 syscall.sd_ble_gap_sec_info_reply
...
...
```

По пространству флагов также можно интерактивно перемещаться в режиме HUD (`V_`).

```
0> syscall.sd_ble_gap_disconnect
- 0x000187b2  syscall.sd_ble_gap_disconnect
  0x000187c2  syscall.sd_ble_gap_disconnect.0
  0x00018a16  syscall.sd_ble_gap_disconnect.1
  0x00018b32  syscall.sd_ble_gap_disconnect.2
  0x0002ac36  syscall.sd_ble_gap_disconnect.3
```

В режиме отладки в radare2 можно использовать `dcs` для продолжения выполнения до следующего системного вызова. Также можно запустить `dcs*` для отслеживания всех системных вызовов.

```
[0xf7fb9120]> dcs*
Running child until syscalls:-1
child stopped with signal 133
--> SN 0xf7fd3d5b syscall 45 brk (0xffffffffda)
child stopped with signal 133
--> SN 0xf7fd28f3 syscall 384 arch_prctl (0xffffffffda 0x3001)
child stopped with signal 133
--> SN 0xfc81b2 syscall 33 access (0xffffffffda 0xf7fd8bf1)
child stopped with signal 133
```

В radare2 включена функция преобразования имени системного вызова в номер системного вызова. Можно получить и имя системного вызова для заданного номера.

```
[0x08048436]> asl 1
exit
[0x08048436]> asl write
4
[0x08048436]> ask write
0x80,4,3,izi
```

Почитайте `as?` для получения дополнительной информации о функции.

Эмуляция ESIL

Одна из самых важных вещей в обратном инжиниринге --- понимание различия между статическим и динамическим анализом. Статический анализ страдает от проблемы комбинаторного взрыва, что невозможно решить даже самым простым способом без хотя бы частичной эмуляции. Многие профессиональные инструменты обратного проектирования используют эмуляцию кода в процессе реверс-инжиниринга, и radare2 здесь не исключение.

Для частичной эмуляции (или неточной полной эмуляции) radare2 использует свой собственный язык ESIL и виртуальную машину. Radare2 поддерживает эмуляцию для всех платформ, для которых заданы ESIL-эмодуляторы (x86/x86_64, ARM, arm64, MIPS, powerpc, sparc, AVR, 8051, Gameboy, ...). Одним из распространенных применений такой эмуляции является расчет косвенных и условных переходов.

Чтобы увидеть ESIL-представление программы, нужно использовать команду `aO` или включить в конфигурации при помощи переменной `asm.esil`. Использование этого режима помогает проверить, правильно ли понимается код программы, освоить эмуляцию ESIL:

```
[0x00001660]> pdf
(fcn) fcn.00001660 40
  fcn.00001660 ();
    ; CALL XREF from 0x00001713 (entry2.fini)
  0x00001660  lea rdi, obj.__progname      ; 0x207220
  0x00001667  push rbp
  0x00001668  lea rax, obj.__progname      ; 0x207220
  0x0000166f  cmp rax, rdi
  0x00001672  mov rbp, rsp
.-< 0x00001675  je 0x1690
  | 0x00001677  mov rax, qword [reloc._ITM_deregisterTMCloneTable] ; [0x206fd8:8]=-0
  | 0x0000167e  test rax, rax
.---< 0x00001681  je 0x1690
  || 0x00001683  pop rbp
  || 0x00001684  jmp rax
`-> 0x00001690  pop rbp
  0x00001691  ret
```

```
[0x000001660]> e asm.esil=true
[0x000001660]> pdf
(fcn) fcn.00001660 40
fcn.00001660 ();
    ; CALL XREF from 0x00001713 (entry2.fini)
0x00001660 0x205bb9,rip,+,rdi,=
0x00001667 rbp,8,rsp,-=,rsp,=[8]
0x00001668 0x205bb1,rip,+,rax,=
0x0000166f rdi,rax,==$z,zf,==$b64,cf,==$p,pf,==$s,sf,==$o,of,=
0x00001672 rsp,rbp,=
.-< 0x00001675 zf,{,5776,rip,=,}
| 0x00001677 0x20595a,rip,+, [8],rax,=
| 0x0000167e 0,rax,rax,&==$z,zf,==$p,pf,==$s,sf,==$o,cf,==$o,of,=
.—< 0x00001681 zf,{,5776,rip,=,}
|| 0x00001683 rsp,[8],rbp,=,8,rsp,+=
|| 0x00001684 rax,rip,=
`-> 0x00001690 rsp,[8],rbp,=,8,rsp,+=
0x00001691 rsp,[8],rip,=,8,rsp,+=
```

Чтобы вручную настроить эмуляцию ESIL, необходимо выполнить следующую последовательность команд:

- `aei` для инициализации виртуальной машины ESIL,
- `aeim` для инициализации памяти виртуальной машины ESIL (стек),
- `aeip` - задание начального IP-адреса виртуальной машины ESIL (указатель инструкций),
- последовательность команд `aer` для задания начальных значений регистра.

При выполнении эмуляции помните, что виртуальная машина ESIL не может эмулировать внешние или системные вызовы вместе с инструкциями SIMD. Таким образом, наиболее распространенным сценарием использования является эмуляция небольшого куска кода, например шифрование/расшифрования, распаковка или вычисления. После успешной настройки виртуальной машины ESIL можно взаимодействовать с ней как в обычном режиме отладки. Интерфейс командной строки виртуальной машины ESIL практически идентичен интерфейсу отладчика:

- `aes` сделать шаг (или клавиша `S` в визуальном режиме, `step`),
- `aesi` для перешагивания через вызовы функций (`step over`),
- `aesu <address>` - выполнить шаг до указанного адреса (`continue`),
- `aesue <ESIL expression>` - выполнять шаги до тех пор, пока не будет истинным какое-либо указанное выражение ESIL,
- `aec` продолжать до прерывания по `Ctrl-C`, этот редко используется из-за вездесущности внешних вызовов.

В визуальном режиме все горячие клавиши отладки будут работать также в режиме эмуляции ESIL.

Наряду с обычной эмуляцией, есть возможность записи и режим воспроизведения (R&R):

- `aets` для перечисления всех текущих сессий ESIL R&R,
- `aets+` для создания новой сессии R&R,
- `aesb` - движение вперед и назад в текущей сессии ESIL R&R.

Подробнее об этом режиме работы можете прочесть в главе Отладка в обратном направлении.

Эмуляция в цикле анализа

Помимо ручного режима эмуляции, ESIL можно использовать автоматически в цикле анализа. Например, команда `aaaa` выполняет этап эмуляции ESIL вместе с другими. Для отключения или включения использования ESIL есть переменная конфигурации `anal.esil`. Есть еще одна важная настройка, хотя и довольно опасная, особенно в случае вредоносного ПО - `emu.write`, которое позволяет виртуальной машине ESIL изменять содержимое памяти. Эта возможность очень полезна в процессе деобфускации или распаковки кода.

Для отображения данных в процессе эмуляции настраивается переменная `asm.emu`. В результате `radare2` будет показывать вычисляемое значение регистра и памяти в комментариях к дизассемблированному коду:

```
[0x000001660]> e asm.emu=true
[0x000001660]> pdf
(fcn) fcn.00001660 40
fcn.00001660 ();
    ; CALL XREF from 0x00001713 (entry2.fini)
0x00001660 lea rdi, obj.__progname ; 0x207220 ; rdi=0x207220 -> 0x464c457f
0x00001667 push rbp ; rsp=0xfffffffffffff8
0x00001668 lea rax, obj.__progname ; 0x207220 ; rax=0x207220 -> 0x464c457f
```

```

0x0000166f cmp rax, rdi          ; zf=0x1 -> 0x2464c45 ; cf=0x0 ; pf=0x1 -> 0x2464c45 ; sf=0x0 ; of=0x0
0x00001672 mov rbp, rsp          ; rbp=0xfffffffffffffff8
.-< 0x00001675 je 0x1690         ; rip=0x1690 -> 0x1f0fc35d ; likely
| 0x00001677 mov rax, qword [reloc._ITM_deregisterTMCloneTable] ; [0x206fd8:8]=0 ; rax=0x0
| 0x0000167e test rax, rax        ; zf=0x1 -> 0x2464c45 ; pf=0x1 -> 0x2464c45 ; sf=0x0 ; cf=0x0 ; of=0x0
.|--< 0x00001681 je 0x1690         ; rip=0x1690 -> 0x1f0fc35d ; likely
|| 0x00001683 pop rbp            ; rbp=0xfffffffffffffff -> 0x4c457fff ; rsp=0x0
|| 0x00001684 jmp rax             ; rip=0x0 ..
`-> 0x00001690 pop rbp            ; rbp=0x10102464c457f ; rsp=0x8 -> 0x464c457f
0x00001691 ret                  ; rip=0x0 ; rsp=0x10 -> 0x3e0003

```

Обратите внимание на комментарии с меткой `likely`, где эмуляция ESIL предсказала переход. Помимо базовой настройки виртуальной машины ESIL, можно изменять ее поведение с помощью других доступных параметров в пространствах имен конфигурации `emu`. и `esil`. Для управления ESIL, работающей с памятью и стеком, можно использовать следующие варианты настроек:

- `esil.stack` для включения или отключения временного стека для режима `asm.emu`,
- `esil.stack.addr` для задания адреса стека в виртуальной машине ESIL (аналогично команде `aeim`),
- `esil.stack.size` для установки размера стека в виртуальной машине ESIL (аналогично команде `aeim`),
- `esil.stack.depth` ограничивает количество операций PUSH в стеке,
- `esil.romem` определяет доступ к памяти ESIL ``только для чтения'',
- `esil.fillstack` и `esil.stack.pattern` позволяют использовать различные шаблоны для заполнения стека виртуальной машины ESIL при инициализации,
- `esil.nonull` - остановить выполнение ESIL при чтении или записи нулевого указателя.

Символы

Radare2 автоматически анализирует доступные разделы импорта и экспорта в двоичном файле, он также может загружать отладочную информацию, если таковая будет найдена. Поддерживаются два основных формата: DWARF и PDB (для двоичных файлов Windows). Обратите внимание, что, в отличие от многих инструментов, `raide2` не использует Windows API для анализа PDB-файлов, поэтому их можно загружать на любой другой поддерживаемой платформе, включая Линукс или OS X.

Отладочная информация DWARF загружается автоматически по умолчанию, поскольку она сохраняется прямо в исполняемом файле. PDB используется по-другому, так как он хранится как отдельный двоичный файл. Одним из распространенных сценариев является анализ файла из дистрибутива Windows. В этом случае все файлы PDB доступны на сервере Microsoft, который по умолчанию есть в настройках. Посмотреть все параметры `pdb` в радаре2:

```

pdb.autoload = 0
pdb.extract = 1
pdb.server = https://msdl.microsoft.com/download/symbols
pdb.useragent = Microsoft-Symbol-Server/6.11.0001.402

```

Используя переменную `pdb.server` можно изменить адрес, по которому Radare2 будет загружать PDB по известному GUID, хранящемуся в заголовке исполняемого файла. Можно использовать несколько серверов, предоставляющих символы, разделив их URL-ы точкой с запятой:

```
e pdb.server = https://msdl.microsoft.com/download/symbols;https://symbols.mozilla.org/
```

В Windows можно также использовать общие сетевые пути (UNC-пути) в качестве серверов символов. Обычно нет особых причин изменять значение по умолчанию `pdb.useragent`, но кто знает...? Поскольку файлы PDB хранятся на сервере в виде CAB-архивов, переменная `pdb.extract=1` настраивает ядро на автоматическую их распаковку. Обратите внимание, что для автоматической загрузки вам нужна утилита операционной системы «`cabextract`» и `wget/curl`.

Иногда нужно грузить символы не в radare2, а в других инструментах, например в `rabin2`:

```

-P          показать отладочную/pdb информацию
-PP         загрузить файл pdb

```

Здесь `-PP` автоматически загружает `pdb` для выбранного двоичного файла, используя настройки в `pdb.*`. Флаг `-P` сбросит дамп файла PDB, его можно открыть и посмотреть хранящихся в нем символы. Помимо основного сценария, простого открытия файла, информацией PDB можно дополнительно манипулировать при помощи команды `id`:

```

[0x0000051c0]> id?
|Usage: id Отладочная информация
| Режим вывода:
| '*'          Выводить в виде команд radare
| id           Строки исходного кода
| idp [file.pdb] Загрузить информацию о файле pdb
| idpi [file.pdb] Показать информацию о файле pdb

```

```
| idpd          Загрузить pdb-файл с удаленного сервера
```

Здесь команда `idpr` делает то же самое, что и `rabin2 -P`. Команду `idp` также можно использовать не только во время статического анализа, но и в режиме отладки, даже если отладка ведется при помощи WinDbg.

Для упрощения использования PDB, особенно в процессе анализа двоичных файлов с большим количеством связанных DLL, radare2 автоматически загружает все необходимые PDB, нужно просто установить `e pdb.autoload=true`. Затем, при открытии какого-либо файла в режиме отладки в Windows, используя `r2 -d` файл.exe или `p2 -d 2345` (присоединиться к pid 2345), все связанные PDB-файлы будут автоматически загружены.

Загрузка информации в формате DWARF полностью автоматизирована. Не нужно запускать какие-либо команды/изменять параметры:

```
r2 `which rabin2`  
[0x000002437 8% 300 /usr/local/bin/rabin2]> pd $r  
0x000002437 jne 0x2468          ;[1]  
0x000002439 cmp qword reloc._cxa_finalize_224, 0  
0x000002441 push rbp  
0x000002442 mov rbp, rsp  
0x000002445 je 0x2453          ;[2]  
0x000002447 lea rdi, obj._dso_handle ; 0x207c40 ; "@| "  
0x00000244e call 0x2360          ;[3]  
0x000002453 call sym.deregister_tm_clones ;[4]  
0x000002458 mov byte [obj.completed.6991], 1 ; obj._TMC_END_ ; [0x2082f0:1]=0  
0x00000245f pop rbp  
0x000002460 ret  
0x000002461 nop dword [rax]  
0x000002468 ret  
0x00000246a nop word [rax + rax]  
;-- entry1.init:  
;-- frame_dummy:  
0x000002470 push rbp  
0x000002471 mov rbp, rsp  
0x000002474 pop rbp  
0x000002475 jmp sym.register_tm_clones ;[5]  
;-- blob_version:  
0x00000247a push rbp          ; ../blob/version.c:18  
0x00000247b mov rbp, rsp  
0x00000247e sub rsp, 0x10  
0x000002482 mov qword [rbp - 8], rdi  
0x000002486 mov eax, 0x32      ; ../blob/version.c:24 ; '2'  
0x00000248b test al, al       ; ../blob/version.c:19  
0x00000248d je 0x2498          ;[6]  
0x00000248f lea rax, str.2.0.1_182_gf1aa3aa4d ; 0x60b8 ; "2.0.1-182-gf1aa3aa4d"  
0x000002496 jmp 0x249f          ;[7]  
0x000002498 lea rax, 0x0000060cd  
0x00000249f mov rsi, qword [rbp - 8]  
0x0000024a3 mov r8, rax  
0x0000024a6 mov ecx, 0x40      ; section_end.ehdr  
0x0000024ab mov edx, 0x40c0  
0x0000024b0 lea rdi, str._s_2.1.0_git__d__linux_x86__d_git._s_n ; 0x60d0 ; "%s 2.1.0-git %d @ linux-x86-%d git.%s\n"  
0x0000024b7 mov eax, 0  
0x0000024bc call 0x2350          ;[8]  
0x0000024c1 mov eax, 0x66      ; ../blob/version.c:25 ; 'f'  
0x0000024c6 test al, al  
0x0000024c8 je 0x24d6          ;[9]  
0x0000024ca lea rdi, str.commit:_f1aa3aa4d2599c1ad60e3ecbe5f4d8261b282385_build:_2017_11_06__12:18:39 ; ../blob/vers  
11-06__1  
0x0000024d1 call sym.imp.puts ;[?]  
0x0000024d6 mov eax, 0          ; ../blob/version.c:28  
0x0000024db leave              ; ../blob/version.c:29  
0x0000024dc ret  
;-- rabin_show_help:  
0x0000024dd push rbp          ; ..//rabin2.c:27
```

Как видите, загружаются имена функций и информация об исходной строке.

Сигнатуры

Radare2 реализует собственный формат сигнатур ``библиотечных'' функций, позволяющий как их загружать/применять, так и создавать на лету. Они доступны в пространстве имён команд `Z`:

```
[0x00000000]> z?  
Usage: z[*j-aof/cs] [args] # Управление сигнатурами  
| z          показать сигнтуры  
| z.         найти сигнтуры, коотрьые соответствуют текущему смещению
```

```

| zb[?][n=5]    поиск наилучшего соответствия
| z*            показать сигнатуры в формате radare
| zq            показать сигнатуры в "тихом" режиме
| zj            показать сигнатуры в формате json
| zk            показать сишнатуры в формате sdb
| z-signature   удалить сигнатуру
| z-*           удалить все сигнатуры
| za[?]         добавить сигнатуру
| zg            сгенерировать сигнатуру (псевдоним для zaF)
| zo[?]         управление файлами сигнатур
| zf[?]         управление сигнатурами формата FLIRT
| z/[?]         поиск сигнатур
| zc[?]         сравнить текущее пространство сигнатур с другим
| zs[?]         управление пространством сигнатур
| zi            показать информацию о соответствии сигнатуре

```

Для загрузки созданного файла сигнатур необходимо загрузить его из SDB-файла с помощью команды `ZO` или из сжатого SDB-файла - `ZOZ`.

Для создания сигнатуры нужно сначала определить функцию, затем из нее сигнатуру:

```

r2 /bin/ls
[0x0000051c0]> aaa # this creates functions, including 'entry0'
[0x0000051c0]> zaf entry0 entry
[0x0000051c0]> z
entry:
  bytes: 31ed4989d15e4889e24883e4f050544c.....48.....48.....ff.....f4
  graph: cc=1 nbbs=1 edges=0 ebbs=1
  offset: 0x0000051c0
[0x0000051c0]>

```

Как видите, сделана новая сигнатура с именем `entry` из функции `entry0`. Также можете представить сигнатуру в формате JSON, что полезно для использования в сценариях:

```

[0x0000051c0]> zj~{}
[
  {
    "name": "entry",
    "bytes": "31ed4989d15e4889e24883e4f050544c.....48.....48.....ff.....f4",
    "graph": {
      "cc": "1",
      "nbbs": "1",
      "edges": "0",
      "ebbs": "1"
    },
    "offset": 20928,
    "refs": [
    ]
  }
]
[0x0000051c0]>

```

Чтобы удалить эту сигнатуру, просто выполните `z-entry`.

Все созданные сигнатур сохраняются в SDB-файл с помощью команды `ZOS myentry`. Позже их можно применить, снова откроем файл:

```

r2 /bin/ls
-- Войти. Взломать. Перейти куда угодно. Получить все.
[0x0000051c0]> zo myentry
[0x0000051c0]> z
entry:
  bytes: 31ed4989d15e4889e24883e4f050544c.....48.....48.....ff.....f4
  graph: cc=1 nbbs=1 edges=0 ebbs=1
  offset: 0x0000051c0
[0x0000051c0]>

```

Это означает, что сигнатурны успешно загружены из файла `myentry` и теперь можно искать соответствующие функции:

```

[0x0000051c0]> z.
[+] searching 0x0000051c0 - 0x0000052c0
[+] searching function metrics
hits: 1
[0x0000051c0]>

```

Обратите внимание, что команда `Z.` просто проверяет сигнатуру по текущему адресу. Чтобы искать сигнатурны по всему файлу, нужно выполнить другую команду. Есть важный момент, если просто запустить ее «как есть», то она ничего не найдет:

```

[0x0000051c0]> z/
[+] searching 0x0021dfd0 - 0x002203e8

```

```
[+] searching function metrics
hits: 0
[0x0000051c0]>
```

Обратите внимание на поисковый адрес --- нам нужно сначала настроить диапазон поиска:

```
[0x0000051c0]> e search.in=io.section
[0x0000051c0]> z/
[+] searching 0x000038b0 - 0x00015898
[+] searching function metrics
hits: 1
[0x0000051c0]>
```

Мы устанавливаем режим поиска в `io.section` (это был файл по умолчанию) на поиск в текущей секции (при условии, что мы все еще находимся в разделе `.text`). Теперь можно посмотреть, что именно нашел radare2:

```
[0x0000051c0]> pd 5
;-- entry0:
;-- sign.bytes.entry_0:
0x0000051c0      31ed          xor ebp, ebp
0x0000051c2      4989d1        mov r9, rdx
0x0000051c5      5e             pop rsi
0x0000051c6      4889e2        mov rdx, rsp
0x0000051c9      4883e4f0      and rsp, 0xfffffffffffffff0
[0x0000051c0]>
```

Здесь видно комментарий `entry0`, который взят из дизассемблирования ELF, а также `sign.bytes.entry_0`, которая является именно результатом совпадения сигнатуры.

Конфигурация для механизма сигнатур хранится в переменных среды в пространстве имен `zign`:

```
[0x0000051c0]> e? zign.
    zignautoload: Автозагрузка всех зигнатур, расположенных в ~/.local/share/radare2/zigns
    zign.bytes: Использование шаблонов байтов для сопоставления
    zign.diff.bthresh: Пороговое значение для дифференциации zign-байтов [0, 1] (см. zc?)
    zign.diff.gthresh: Порог для дифференцированных графиков zign [0, 1] (см. zc?)
    zign.graph: Использование метрик графа для сопоставления
    zign.hash: Использование хеша для сопоставления
    zign.maxsz: Максимальная длина сигнатуры
    zign.mincc: Минимальная цикломатическая сложность для сопоставления
    zign.minsz: Минимальная длина сигнатуры для сопоставления
    zign.offset: Используйте исходное смещение для сопоставления
    zign.prefix: Префикс по умолчанию для совпадений сигнатур
    zign.refs: Использовать ссылки для сопоставления
    zign.threshold: Минимальное сходство, необходимое для включения в результаты zb
    zign.types: Использование типов для сопоставления
[0x0000051c0]>
```

Поиск лучших совпадений `zb`

Часто бывает, что сигнатура должна существовать где-то в двоичном файле, но `z/` и `z.` все еще терпят неудачу. Это часто связано с очень незначительными различиями между сигнатурой и функцией. Например, компилятор поменял местами две инструкции или сигнатура не для подходящей версии функции. В этих ситуациях команды `zb` по-прежнему могут помочь указать вам правильное направление, перечислив примерные совпадения.

```
[0x000040a0]> zb?
Usage: zb[r?] [args] # поиск сигнатуры, имеющей наилучшее совпадение
| zb [n]           найти n наиболее подходящих сигнатур для функции по текущему смещению
| zbr zigname [n]  поиск n наиболее похожих функций на заданную сигнатуру
```

Команда `zb` (`zign best`) покажет пять наиболее близких сигнатур для заданной функции. Каждая из них будет содержать оценку от 0.0 до 1.0.

```
[0x0041e390]> s sym.fclose
[0x0040fc10]> zb
0.96032 0.92400 B 0.99664 G  sym.fclose
0.65971 0.35600 B 0.96342 G  sym._nl_expand_alias
0.65770 0.37800 B 0.93740 G  sym.fdopen
0.65112 0.35000 B 0.95225 G  sym._run_exit_handlers
0.62532 0.34800 B 0.90264 G  sym._cxa_finalize
```

В приведенном выше примере `zb` правильно связала подпись `sym.fclose` с текущей функцией. Команды `z/` и `z.` не сработали бы здесь, так как оценки Byte и Ghash меньше 1.0. Разница 30% между результатами первого и второго места также является хорошим показателем правильного соответствия.

Команда `zbr` (`zign best reverse`) принимает имя сигнатуры и пытается найти наиболее близко соответствующие функции. Используйте команду анализа, например `aA`, чтобы найти сначала функции.

```
[0x00401b20]> aa
[x] Analyze all flags starting with sym. and entry0 (aa)
[0x00401b20]> zo ./libc.sdb
[0x00401b20]> zbr sym._libc_malloc 10
0.94873 0.89800 B 0.99946 G sym.malloc
0.65245 0.40600 B 0.89891 G sym._mid_memalign
0.59470 0.38600 B 0.80341 G sym._IO_flush_all_lockp
0.59200 0.28200 B 0.90201 G sym._IO_file_underflow
0.57802 0.30400 B 0.85204 G sym._libc_realloc
0.57094 0.35200 B 0.78988 G sym._calloc
0.56785 0.34000 B 0.79570 G sym._IO_un_link.part.0
0.56358 0.36200 B 0.76516 G sym._IO_cleanup
0.56064 0.26000 B 0.86127 G sym.intel_check_word.constprop.0
0.55726 0.28400 B 0.83051 G sym.linear_search_fdes
```

Команды графа управления

При анализе данных обычно удобно иметь разные способы их представления, чтобы получить новый обобщенный взгляд, позволяющий аналитику понять, как взаимодействуют различные части программы. Представление базовых блоков (далее в разделе, просто блоков), вызовов функций, строковых ссылок в виде графов (графов управления) показывает очень четкое представление этой информации. Radare2 поддерживает различные типы графов, доступные с помощью команд, начинающихся с `ag`:

```
[0x000005000]> ag?
|Usage: ag<graphtype><format> [addr]
| Команды графов:
| aga[format] Граф ссылок данных
| agA[format] Глобальный граф ссылок данных
| agc[format] Граф функций
| agC[format] Глобальный граф вызовов функций
| agd[format] [fcn addr] Граф различных
| agf[format] Граф базовых блоков функции
| agi[format] Граф импортов
| agr[format] Граф ссылок
| agR[format] Глобальный граф ссылок
| agx[format] Граф перекрестных ссылок
| agg[format] Граф общего вида, настраиваемый
| ag- Очистить граф общего вида
| agn[?] title body Добавить узел в настраиваемый граф
| age[?] title1 title2 Добавить дугу в настраиваемый граф

Форматы вывода:
| <blank> Псевдографика Ascii
| * Команды r2
| d Формат dot Graphviz-a
| g Формат gml (Graph Modelling Language)
| j JSON ('J' для форматированного дизассемблирования)
| k Ключ-значение SDB
| t Крошечный ascii-art
| v Интерактивный ascii-art
| w [path] Записать в файл или показать на экране изображение (смотрите graph.gv.format и graph.web)
```

Структура команд следующая: `ag <graph type> <output format>`. Например, `agid` отображает граф импорта в точечном формате, а `aggj` выводит граф в формате JSON. Вот краткое описание для каждого доступного формата вывода:

Ascii Art ** (например, `agf`)

Выводит граф на `stdout`, используя ASCII-art для представления блоков и ребер. *Предупреждение: отображение больших графов в stdout может оказаться вычислительно дорогостоящим, r2 может перестать реагировать на команды в течение некоторого времени.* В случае сомнений, используйте интерактивное представление (рассматривается ниже).

Интерактивный Ascii-art (например, `agfv`)

Отображает граф в ASCII в интерактивном представлении, аналогичном `VV`, позволяющее перемещать экран, увеличивать/уменьшать, ...

Крошечный Ascii-art (например, `agft`)

Отображает график ASCII непосредственно в `stdout` в этом режиме (что аналогично достижению максимального уровня уменьшения масштаба в интерактивном представлении).

Graphviz dot (например, `agfd`)

Печатает исходный код dot, представляющий граф, интерпретируемый программами типа graphviz или онлайн-вьюверами, подобными этому.

JSON (например, `agfj`)

Выводит строку JSON, представляющую граф.

- В случае формата f (блоки функции), будет отображена подробно информация о функции, ее дизассемблирование (используйте формат J для форматированного дизассемблирования).
- Во всех остальных случаях он будет содержать только основную информацию об узлах графа (идентификатор, заголовок, тело и ребра).

Язык моделирования графа (например, `agfg`)

Выводит исходный код GML, представляющий граф, интерпретируемый, например, программой yEd

Ключ-значение SDB (например, `agfk`)

Печатает строки ``ключ-значение", представляющие граф, сохраненный в sdb (строковая база данных radare2).

Пользовательские команды R2 для графа (например, `agf*`)

Печатает команды r2, которые воссоздают нужный график. Команды для построения графа: agn [title] [body] - добавление узла и age [title1] [title2] - добавление ребра. Поле [body] можно представлять в base64 для включения специального форматирования (например, новых строк).

Чтобы выполнять напечатанные команды, можно добавить точку к команде (. agf*).

Веб / изображение (например, `agfw`)

Radare2 преобразует график в формат dot. Программа dot преобразует его в .gif-изображение, а затем результат можно просмотреть в вашей операционной системе (xdg-open, open, eog, xviewer, ...).

Расширение файла для изображения задается с помощью переменной конфигурации graph.extension. Доступны расширения png, jpg, gif, pdf, ps.

Примечание: для особенно больших графиков рекомендуемым расширением является SVG, так как оно будет производить изображения гораздо меньшего размера.

Если переменная конфигурации graph.web включена, radare2 попытается отобразить график с помощью браузера (эта функция является экспериментальной и незавершенной, отключена по умолчанию.)

Скрипты

Radare2 предоставляет широкий набор функций для автоматизации ``буровых работ''. Он варьируется от простой последовательности команд до вызова скриптов/программы через IPC (межпроцессное взаимодействие), называемое r2pipe. Как уже упоминалось несколько раз, есть возможность писать последовательности команд, используя знак ;.

```
[0x00404800]> pd 1 ; ao 1
      0x00404800      b827e66100      mov eax, 0x61e627      ; "tab"
address: 0x404800
opcode: mov eax, 0x61e627
prefix: 0
bytes: b827e66100
ptr: 0x0061e627
refptr: 0
size: 5
type: mov
esil: 6415911,rax,=
stack: null
family: cpu
[0x00404800]>
```

Такая форма просто запускает вторую команду после завершения первой, как в командной строке операционной системы. Второй способ создать последовательность команд --- использовать обычный канал |.

адрес ao|grep

Обратите внимание, что | может передавать вывод команды r2 только во внешнюю команду оболочки или в другую программу. Существует аналогичный способ перечислять команды r2 с использованием оператора обратной кавычки `команда`. Часть в кавычках подвергнется подстановке команд, а вывод будет использоваться в качестве аргумента командной строки.

Например, мы хотим увидеть несколько байтов памяти по указанному адресу, на который ссылаются инструкции `mov eax, addr'. Сделать это можно, используя последовательность команд:

```
[0x00404800]> pd 1
0x00404800      b827e66100    mov eax, 0x61e627      ; "tab"
[0x00404800]> ao
address: 0x404800
opcode: mov eax, 0x61e627
prefix: 0
bytes: b827e66100
ptr: 0x0061e627
refptr: 0
size: 5
type: mov
esil: 6415911,rax,=
stack: null
family: cpu
[0x00404800]> ao~ptr[1]
0x0061e627
0
[0x00404800]> px 10 @ `ao~ptr[1]`
- offset - 0 1 2 3 4 5 6 7 8 9 A B C D E F 0123456789ABCDEF
0x0061e627 7461 6200 2e69 6e74 6572           tab..inter
[0x00404800]>
```

Можно перенаправить вывод команды r2 в файл, используя > и >>.

```
[0x00404800]> px 10 @ `ao~ptr[1]` > example.txt
[0x00404800]> px 10 @ `ao~ptr[1]` >> example.txt
```

Radare2 также предоставляет много команд обработки файлов типа Unix, таких, как head, tail, cat, grep и многие другие. Одна из таких команд - Uniq, которую можно использовать для фильтрации и отображения только неповторяющегося содержимого. Итак, чтобы создать новый файл только с уникальными строками, можно сделать:

```
[0x00404800]> uniq file > uniq_file
```

Команда head может использоваться для просмотра первых N строк в файле, аналогично tail, позволяющих видеть последние N строк.

```
[0x00404800]> head 3 foodypes.txt
1 Protein
2 Carbohydrate
3 Fat
[0x00404800]> tail 2 foodypes.txt
3 Shake
4 Milk
```

Команда join может использоваться для объединения двух разных файлов с общим первым полем.

```
[0x00404800]> cat foodypes.txt
1 Protein
2 Carbohydrate
3 Fat
[0x00404800]> cat foods.txt
1 Cheese
2 Potato
3 Butter
[0x00404800]> join foodypes foods.txt
1 Protein Cheese
2 Carbohydrate Potato
3 Fat Butter
```

Точно так же делается сортировка контента при помощи команды sort. Типичный пример:

```
[0x00404800]> sort file
eleven
five
five
great
one
one
radare
```

Команда ?\$? предоставляет информацию о нескольких полезных переменных, похожих на переменные \$v - ``immediate value'' - и \$m - ссылка на память из оп-кода.

Итераторы

Одной из наиболее распространенных задач в автоматизации является циклическая обработка однотипных элементов, и в radare2 есть несколько способов выполнять такие операции.

Перебор флагов:

```
穷 flagname-regex
```

Например, надо увидеть информацию о функции с помощью команды `afi`:

```
[0x004047d6]> afi
#
offset: 0x004047d0
name: entry0
size: 42
realsz: 42
stackframe: 0
call-convention: amd64
cyclomatic-complexity: 1
bits: 64
type: fcn [NEW]
num-bbs: 1
edges: 0
end-bbs: 1
call-refs: 0x00402450 C
data-refs: 0x004136c0 0x00413660 0x004027e0
code-xrefs:
data-xrefs:
locals:0
args: 0
diff: type: new
[0x004047d6]>
```

Теперь предположим, мы хотели бы видеть определенное поле из этого текста для всех функций, найденных в результате анализа. Можем сделать это с помощью цикла по всем флагам функций, имя которых начинается с `fcn`:

```
[0x004047d6]> fs functions
[0x004047d6]> afi 穷 fcn.* ~name
```

Эта команда извлекает поле `name` из вывода `afi` каждого флага с именем, соответствующим регулярному выражению `fcn.*`. Для управления популярных операций в r2 реализован специальный цикл `穷f`, запускающий команду для каждой функции, найденной r2:

```
[0x004047d6]> afi 穷f ~name
```

Также можно просмотреть список смещений, используя следующий синтаксис:

```
穷=1 2 3 ... N
```

Предположим, что надо увидеть информацию о коде операции для двух смещений, текущего и текущего + 2:

```
[0x004047d6]> ao 穷=$$ $$+2
address: 0x4047d6
opcode: mov rdx, rsp
prefix: 0
bytes: 4889e2
refptr: 0
size: 3
type: mov
esil: rsp,rdx,=
stack: null
family: cpu
address: 0x4047d8
opcode: loop 0x404822
prefix: 0
bytes: e248
refptr: 0
size: 2
type: cjmp
esil: 1,rcx,-=,rcx,{,4212770,rip,=,}
jump: 0x00404822
fail: 0x004047da
stack: null
cond: al
family: cpu
[0x004047d6]>
```

Обратите внимание, что используется переменная `$$` равная текущему смещению. Выражение `$$+2` вычисляется перед циклом, поэтому можно использовать простые арифметические выражения.

Третий способ создания циклов --- загрузка смещений из файла. Файл должен содержать одно смещение на строку.

```
[0x0040407d0]> ?v $$ > offsets.txt
[0x0040407d0]> ?v $$+2 >> offsets.txt
[0x0040407d0]> !cat offsets.txt
4047d0
4047d2
[0x0040407d0]> pi 1 @.offsets.txt
xor ebp, ebp
mov r9, rdx
```

Radare2 также предлагает различные конструкции `foreach`. Одним из наиболее полезных является перебор всех инструкций функции:

```
[0x0040407d0]> pdf
[fcn] entry0 42
; UNKNOWN XREF from 0x00400018 (unk)
; DATA XREF from 0x004064bf (sub.strlen_460)
; DATA XREF from 0x00406511 (sub.strlen_460)
; DATA XREF from 0x0040b080 (unk)
; DATA XREF from 0x0040b0ef (unk)
0x0040407d0 xor ebp, ebp
0x0040407d2 mov r9, rdx
0x0040407d5 pop rsi
0x0040407d6 mov rdx, rsp
0x0040407d9 and rsp, 0xfffffffffffff0
0x0040407dd push rax
0x0040407de push rsp
0x0040407df mov r8, 0x4136c0
0x0040407e6 mov rcx, 0x413660 ; "AWA..AVI..AUI..ATL.%.. "
0A..AVI..AUI.
0x0040407ed mov rdi, main ; "AWAVAUATUH..S..H...." @
0
0x0040407f4 call sym.imp.__libc_start_main
0x0040407f9 hlt
[0x0040407d0]> pi 1 @i
mov r9, rdx
pop rsi
mov rdx, rsp
and rsp, 0xfffffffffffff0
push rax
push rsp
mov r8, 0x4136c0
mov rcx, 0x413660
mov rdi, main
call sym.imp.__libc_start_main
hlt
```

В этом примере команда `pi 1` печатает все инструкции в текущей функции (`entry0`). Есть и другие варианты (неполный список,смотрите инструкции `@@@?`): - `@@@k sdbquery` - перебрать все смещения, полученные в запросе к sdb, - `@@@t` - перебрать все нити (threads,смотрите `dp`), - `@@@b` - перебрать все базовые блоки текущей функции (смотрите `afb`), - `@@@f` - перебрать все функции (смотрите `aflq`).

Последний вид циклов позволяет перебирать предопределенные типы итераторов:

- символы,
- импорты,
- реестры,
- нити,
- комментарии,
- функции,
- флаги.

Перебор делается с помощью команды `@@@@`. Предыдущий пример вывода информации о функциях также можно выполнить с помощью команды `@@@@`:

```
[0x0040407d6]> afi @@@ functions ~name
```

В результате удается извлечь поле `name` из вывода `afi`, при этом будет выведен огромный список названий функций. Можно выбрать, например, только второй столбец, убрать лишнюю строку `name:` в каждой строке:

```
[0x0040407d6]> afi @@@ functions ~name[1]
```

Осторожно, @@@ несовместим с командами JSON.

Макросы

Помимо просто последовательности команд и циклов, radare2 позволяет записывать простые макросы, используя конструкцию:

```
[0x00404800]> (qwe; pd 4; ao)
```

Эта форма определяет макрос под названием `qwe`, запускающий последовательно сначала `pd 4`, затем `ao`. Вызов макроса использует синтаксис `.(имя_макроса)`:

```
[0x00404800]> (qwe; pd 4; ao)
[0x00404800]> .(qwe)
0x00404800  mov eax, 0x61e627      ; "tab"
0x00404805  push rbp
0x00404806  sub rax, section_end.LOAD1
0x0040480c  mov rbp, rsp
```

```
address: 0x404800
opcode: mov eax, 0x61e627
prefix: 0
bytes: b827e66100
ptr: 0x0061e627
refptr: 0
size: 5
type: mov
esil: 6415911,rax,=
stack: null
family: cpu
[0x00404800]>
```

Чтобы просмотреть доступные макросы, вызовите `(*)`:

```
[0x00404800]> (*
(qwe ; pd 4; ao)
```

А если хотите удалить какой-то макрос, просто добавьте `'-'` перед названием:

```
[0x00404800]> (-qwe)
Macro 'qwe' removed.
[0x00404800]>
```

Можно создать макрос, принимающий аргументы, --- удобный механизм при создании простых сценариев. Чтобы создать макрос, принимающий аргументы, просто добавьте их в определение макроса.

```
[0x00404800]
[0x004047d0]> (foo x y; pd $0; s +$1)
[0x004047d0]> .(foo 5 6)
;-- entry0:
0x004047d0      xor ebp, ebp
0x004047d2      mov r9, rdx
0x004047d5      pop rsi
0x004047d6      mov rdx, rsp
0x004047d9      and rsp, 0xfffffffffffffff0
[0x004047d6]>
```

Как видите, аргументы называются по индексу, начиная с 0: \$0, \$1, ...

Псевдонимы

Radare2 также включает псевдонимы, помогающие сэкономить время, быстро выполняя наиболее часто используемые команды. Они находятся в `$?`

Общее использование функции - `$alias=cmd`

```
[0x00404800]> $disas=pdf
```

Приведенная выше команда создаст псевдоним `disas` для `pdf`. Следующая команда выводит разборку основной функции.

```
[0x00404800]> $disas @ main
```

Помимо команд можно печатать текст.

```
[0x00404800]> $my_alias=$test input
[0x00404800]> $my_alias
test input
```

Чтобы отменить определение псевдонима, используйте `$alias=:`

```
[0x00404800]> $pmore='b 300;px'  
[0x00404800]> $  
$pmore  
[0x00404800]> $pmore=  
[0x00404800]> $
```

Один \$ в приведенном выше списке будет перечислять все определенные псевдонимы. Также можно посмотреть команду псевдонима:

```
[0x00404800]> $pmore?  
b 200; px
```

Можно ли создать псевдоним, включающий другой псевдоним? Ответ --- да.

```
[0x00404800]> $pStart='s 0x0;$pmore'  
[0x00404800]> $pStart  
- offset - 0 1 2 3 4 5 6 7 8 9 A B C D E F 0123456789ABCDEF  
0x00000000 7f45 4c46 0201 0100 0000 0000 0000 0000 .ELF.....  
0x00000010 0300 3e00 0100 0000 1014 0000 0000 0000 ..>.....  
0x00000020 4000 0000 0000 0000 5031 0000 0000 0000 @.....P1.....  
0x00000030 0000 0000 4000 3800 0d00 4000 1e00 1d00 .....@.8..@.....  
0x00000040 0600 0000 0400 0000 4000 0000 0000 0000 .....@.....  
0x00000050 4000 0000 0000 4000 0000 0000 0000 0000 @.....@.....  
0x00000060 d802 0000 0000 0000 d802 0000 0000 0000 .....  
0x00000070 0800 0000 0000 0000 0300 0000 0400 0000 .....  
0x00000080 1803 0000 0000 0000 1803 0000 0000 0000 .....  
0x00000090 1803 0000 0000 0000 1c00 0000 0000 0000 .....  
0x000000a0 1c00 0000 0000 0000 0100 0000 0000 0000 .....  
0x000000b0 0100 0000 0400 0000 0000 0000 0000 0000 .....  
0x000000c0 0000 0000 0000 0000 .....  
[0x00000000]>
```

Программа R2pipe

API r2pipe первоначально разработан для NodeJS, чтобы поддерживать повторное использование веб-API r2.js из командной строки. Модуль r2pipe позволяет взаимодействовать с процессами r2 различными методами:

- каналы после запуска spawn (r2 -0),
- запросы http (удобный для облачных сервисов),
- сокет tcp (r2 -c).

	pipe	spawn	async	http	tcp	rap	json
nodejs	x	x	x	x	x	-	x
python	x	x	-	x	x	x	x
swift	x	x	x	x	-	-	x
dotnet	x	x	x	x	-	-	-
haskell	x	x	-	x	-	-	x
java	-	x	-	x	-	-	-
golang	x	x	-	-	-	-	x
ruby	x	x	-	-	-	-	x
rust	x	x	-	-	-	-	x
vala	-	x	x	-	-	-	-
erlang	x	x	-	-	-	-	-
newlisp	x	-	-	-	-	-	-
dlang	x	-	-	-	-	-	x
perl	x	-	-	-	-	-	-

Примеры

Python

```
$ pip install r2pipe  
import r2pipe  
  
r2 = r2pipe.open("/bin/ls")  
r2.cmd('aa')  
print(r2.cmd("afl"))  
print(r2.cmdj("aflj")) # вычисляет JSON-ы и возвращает объект
```

NodeJS

Используйте команду для установки модулей привязки к r2pipe

```
$ npm install r2pipe

Вот пример ``hello world``:

const r2pipe = require('r2pipe');
r2pipe.open('/bin/ls', (err, res) => {
    if (err) {
        throw err;
    }
    r2.cmd ('af @ entry0', function (o) {
        r2.cmd ("pdf @ entry0", function (o) {
            console.log (o);
            r.quit ()
        });
    });
});
});
```

В репозитории GIT есть другие примеры и дополнительные сведения.

<https://github.com/radareorg/radare2-r2pipe/blob/master/nodejs/r2pipe/README.md>

Go

```
$ r2pm -i r2pipe-go

https://github.com/radare/r2pipe-go
```

```
package main

import (
    "fmt"
    "github.com/radare/r2pipe-go"
)

func main() {
    r2p, err := r2pipe.NewPipe("/bin/ls")
    if err != nil {
        panic(err)
    }
    defer r2p.Close()
    buf1, err := r2p.Cmd("?E Hello World")
    if err != nil {
        panic(err)
    }
    fmt.Println(buf1)
}
```

Rust

```
$ cat Cargo.toml
...
[dependencies]
r2pipe = "*"

#[macro_use]
extern crate r2pipe;
use r2pipe::R2Pipe;
fn main() {
    let mut r2p = open_pipe!(Some("/bin/ls")).unwrap();
    println!("{:?}", r2p.cmd("?e Hello World"));
    let json = r2p.cmdj("ij").unwrap();
    println!("{}", serde_json::to_string_pretty(&json).unwrap());
    println!("ARCH {}", json["bin"]["arch"]);
    r2p.close();
}
```

Ruby

```
$ gem install r2pipe
```

```

require 'r2pipe'
puts 'r2pipe ruby api demo'
puts '=====
r2p = R2Pipe.new '/bin/ls'
puts r2p.cmd 'pi 5'
puts r2p.cmd 'pij 1'
puts r2p.json(r2p.cmd 'pij 1')
puts r2p.cmd 'px 64'
r2p.quit

```

Perl

```
#!/usr/bin/perl
```

```

use R2::Pipe;
use strict;

my $r = R2::Pipe->new ("/bin/ls");
print $r->cmd ("pd 5")."\n";
print $r->cmd ("px 64")."\n";
$r->quit ();

```

Erlang

```

#!/usr/bin/env escript
%% -- erlang --
%%! -smp enable

%% -sname hr
-mode(compile).

-export([main/1]).

main(_Args) ->
    %% добавление r2pipe в modulepath, задайте расположение r2pipe_erl
    R2pipePATH = filename:dirname(escript:script_name()) ++ "/ebin",
    true = code:add_pathz(R2pipePATH),

    %% инициализация ссылки с помощью r2
    H = r2pipe:init(lpipe),

    %% вся работа идет здесь
    io:format("~s", [r2pipe:cmd(H, "i")]).
```

Haskell

```

import R2pipe
import qualified Data.ByteString.Lazy as L

showMainFunction ctx = do
    cmd ctx "s main"
    L.putStr =<< cmd ctx "pD `fl $$`"

main = do
    -- Запуск r2 локально
    открыть "/bin/ls" >>= showMainFunction
    -- Соединение с r2 через HTTP (например, если "r2 -qc=h /bin/ls" запущен)
    открытие "http://127.0.0.1:9090" >>= showMainFunction

```

Dotnet

```

using System;
using System.Collections.Generic;
```

```

using System.Diagnostics;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using r2pipe;

namespace LocalExample {
    class Program {
        static void Main(string[] args) {
#if __MonoCS__
        using(IR2Pipe pipe = new R2Pipe("/bin/ls")) {
#else
        using (IR2Pipe pipe = new R2Pipe(@"C:\Windows\notepad.exe",
            @"C:\radare2\radare2.exe")) {
#endif
            Console.WriteLine("Hello r2! " + pipe.RunCommand("?V"));
            Task<string> async = pipe.RunCommandAsync("?V");
            Console.WriteLine("Hello async r2!" + async.Result);
            QueuedR2Pipe qr2 = new QueuedR2Pipe(pipe);
            qr2.Enqueue(new R2Command("x", (string result) => {
                Console.WriteLine("Result of x:\n {0}", result); }));
            qr2.Enqueue(new R2Command("pi 10", (string result) => {
                Console.WriteLine("Result of pi 10:\n {0}", result); }));
            qr2.ExecuteCommands();
        }
    }
}
}

```

Java

```

import org.radare.r2pipe.R2Pipe;

public class Test {
    public static void main (String[] args) {
        try {
            R2Pipe r2p = new R2Pipe ("/bin/ls");
            // new R2Pipe ("http://cloud.rada.re/cmd/", true);
            System.out.println (r2p.cmd ("pd 10"));
            System.out.println (r2p.cmd ("px 32"));
            r2p.quit();
        } catch (Exception e) {
            System.err.println (e);
        }
    }
}

```

Swift

```

if let r2p = R2Pipe(url:nil) {
    r2p.cmd ("?V", closure:{(str:String?) in
        if let s = str {
            print ("Version: \(s)");
            exit (0);
        } else {
            debugPrint ("R2PIPE. Error");
            exit (1);
        }
    });
    NSRunLoop.currentRunLoop().run();
} else {
    print ("Needs to run from r2")
}

```

Vaal

```
public static int main (string[] args) {
    MainLoop loop = new MainLoop ();
    var r2p = new R2Pipe ("/bin/ls");
    r2p.cmd ("pi 4", (x) => {
        stdout.printf ("Disassembly:\n%s\n", x);
        r2p.cmd ("ie", (x) => {
            stdout.printf ("Entrypoint:\n%s\n", x);
            r2p.cmd ("q");
        });
    });
    ChildWatch.add (r2p.child_pid, (pid, status) => {
        Process.close_pid (pid);
        loop.quit ();
    });
    loop.run ();
    return 0;
}
```

NewLisp

```
(load "r2pipe.lsp")
(setq pd (r2pipe:cmd "pd 3"))
(exit)
```

Dlang

```
import std.stdio;
import r2pipe;

void main() {
    auto r2 = r2pipe.open ();
    writeln ("Hello ~ r2.cmd("?e World"));
    writeln ("Hello ~ r2.cmd("?e Works"));

    string uri = r2.cmdj("ij")["core"]["uri"].str;
    writeln ("Uri: ", uri);
}
```

Отладчик

Отладчики реализуются при помощи плагинов ввода-вывода. Поэтому radare2 может обрабатывать различные типы URI, порождая, присоединяясь и управляя процессами. Полный список плагинов доступен при использовании флага `r2 -L` при запуске. Плагины с буквой ``d'' в первом столбце (`rwd') таблицы поддерживают отладку. Например:

```
r_d debug      Отладить программу или процесс по pid. dbg://bin/ls, dbg://1388 (LGPL3)
rwd gdb        Подсоединиться к серверу gdb, 'qemu -s', gdb://localhost:1234 (LGPL3)
```

Существуют разные бэкенды для многих целевых архитектур и операционных систем, например, GNU/Linux, Windows, MacOS X, (Net, Free, Open)BSD и Solaris.

Память процесса обрабатывается как обычный файл. Все отображаемые страницы памяти отлаживаемой программы и ее библиотек читаются и интерпретируются как код или структуры данных.

Связь между radare и уровнем ввода-вывода отладчика обрамлена в вызовы `System()`, принимающий строку в качестве аргумента и выполняющий ее как команду. Затем ответ буферизуется в консоли вывода, его содержимое может быть дополнительно обработано скриптом. Доступ к системе ввода-вывода осуществляется с помощью `=!`. Большинство подключаемых модулей ввода-вывода снабжены инструкциями, доступными при помощи `=!?` и `=!help`. Например:

```
$ r2 -d /bin/ls
...
[0x7fc15afa3cc0]> =!help
Usage: =!cmd args
=!ptrace - использовать ptrace io
=!mem   - использовать /proc/pid/mem io, если возможно
=!pid   - показать pid
=!pid <#> - выбрать новый pid
```

Как правило, команды отладчика переносимы между архитектурами и операционными системами. Radare пытается поддерживать одну и ту же функциональность для всех целевых архитектур и операционных систем, но некоторые вещи приходится решать отдельно. Они включают внедрение shell-кодов и обработку исключений. Например, в архитектурах MIPS нет аппаратной поддержки пошагового запуска инструкций. В этом случае radare2 предоставляет собственную реализацию пошагового выполнения, используя сочетание анализа кода и программных точек останова.

Базовая справка по отладчику доступна по команде `d?`:

```
Usage: d # Команды отладчика
| db[?] Управление точками останова
| dbt[?] Показать трассировку стека, учитывая значения в dbg.btdepth и dbg.btalgo
| dc[?] Продолжить выполнение (c)
| dd[?] Файловые дескрипторы (!fd в r1)
| de[-sc] [perm] [rm] [e] Отладить при помощи ESIL (смотрите de?)
| dg <file> Сгенерировать core-файл (WIP)
| dH [handler] Трансплантировать процесс на новый обработчик
| di[?] Показать информацию о механизме отладчика (смотрите dh)
| dk[?] Перечислить, отправить, получить, задать обработчик сигнала у дочернего процесса
| dL[?] Перечислить или установить обработчик отладчика
| dm[?] Показать карты памяти
| do[?] Открыть процесс (перезагрузить, псевдоним для 'oo')
| doo[args] Открыть в режиме отладки с аргументами (псевдоним для 'ood')
| doof[file] Открыть в режиме отладки из файла (псевдоним для 'oodf')
| doc Закрыть сессию отладки
| dp[?] Перечислить, присоединиться к процессу или нити по идентификатору
| dr[?] Регистры процессора
| ds[?] Сделать шаг отладки (step, step over), исходный код
| dt[?] Показать трассировку инструкций
| dw <pid> Блокировать командную строку, пока процесс не завершился
| dx[?] Добавить и запустить код в процесс (смотрите gs)
```

Чтобы перезапустить сеанс отладки, можно ввести OO или OO+ в зависимости от задачи.

```
oo заново открыть текущий файл (kill+fork в отладчике)
oo+ заново открыть текущий файл в режиме перезаписи
```

Начало работы

Небольшая сессия в отладчике Radeone2

- r2 -d /bin/ls: открывает radare2 с файлом /bin/ls в режиме отладчика, используя встроенный отладчик, при этом программа сразу останавливается; запустится командная строка radare2;
- db flag: установить точку останова на флаг, где флаг - это либо адрес, либо имя функции;
- db - flag: удалить точку останова на флаге, где флаг - это либо адрес, либо имя функции;
- db: показать список точек останова;
- dc: запуск программы;
- dr: показать состояние регистров;
- drr: показать ссылки из регистров (telescoping) (как peda);
- ds: выполнить инструкцию, войдя в нее, если можно (step into);
- dso: выполнить инструкцию без входа внутрь (step over);
- dbt: показать трассировку стека;
- dm: показать карты памяти;
- dk <signal>: отправить сигнал KILL дочернему процессу;
- ood: переоткрыть в режиме отладки;
- ood arg1 arg2: переоткрыть в режиме отладки с параметрами arg1 и arg2.

Переход с IDA, GDB и WinDBG

Как запустить программу с помощью отладчика

r2 -d /bin/ls - запуск в режиме отладки => [видео].

Как присоединиться/отсоединиться от исполняющегося процесса? (gdb -p)

r2 -d <pid> - присоединившихся к процессу,
r2 ptrace://pid - подсоединиться к процессу, но только для io (не используется бэкенд отладчика),
[0x7fff6ad90028]> o-225 - закрыть fd=225 (перечень из o~[1]:0),
r2 -D gdb gdb://localhost:1234 - подключиться к серверу gdb.

Как установить аргументы/переменные среды/загрузить динамическую библиотеку в сеансе отладки radare

Используйте rarun2 (libpath=\$PWD:/tmp/lib, arg2=hello, setenv=FOO=BAR ...), инструкции - rarun2 -h /man rarun2.

Как запускать скрипты в radare2 ?

r2 -i <scriptfile> ... - запуск скрипта **после** загрузки файла => [video],
r2 -I <scriptfile> ... - запуск скрипта **до** загрузки файла,
r2 -c \$@ | awk \$@ - пропустить через awk для получения asm-кода функции => [link],
[0x80480423]> . scriptfile - интерпретировать файл => [video],
[0x80480423]> #!c - вход в C-шный repl (см. #! перечень доступных плагинов RLang) => [video], все должно быть сделано в одной строке, можно передать файл.c в качестве аргумента.

Если нужен #!python и другие варианты, просто соберите модуль radare2-bindings.

Как показать исходный код подобно gdb?

CL @ sym.main - данная функция еще недостаточно стабилизирована.

Сочетания клавиш

Команда	IDA Pro	Программа radare2	r2 (визуальный режим)	GDB	WinDbg
Анализ					
Анализ всех данных и кода	Автоматически запускается при открытии бинарника	aaa или -A (aaaa или -AA для запуска новых экспериментальных видов анализа)	отсутствует	отсутствует	отсутствует
Управление интерфейсом					
переход по внешней ссылке	x	axt	x	отсутствует	отсутствует
переход с внешней ссылки	ctrl + j	axf	X	отсутствует	отсутствует
xref в граф	?	agt [offset]	?	отсутствует	отсутствует
xref из графа	?	agf [offset]	?	отсутствует	отсутствует
список функций	alt + 1	afl;is	t	отсутствует	отсутствует
Листинг	alt + 2	pdf	p	отсутствует	отсутствует
режим hex	alt + 3	pxa	P	отсутствует	отсутствует
импорты	alt + 6	ii	:ii	отсутствует	отсутствует
экспорты	alt + 7	is~FUNC	?	отсутствует	отсутствует

Команда	IDA Pro	Программа radare2	r2 (визуальный режим)	GDB	WinDbg
переход по jmp/call	enter	s offset	enter или 0-9	отсутствует	отсутствует
отмена seek	esc	s -	u	отсутствует	отсутствует
вернуть прежний seek	ctrl+enter	s +	U	отсутствует	отсутствует
показать график управления	space	agv	V	отсутствует	отсутствует
Правка					
переименование	n	a fn	dr	отсутствует	отсутствует
режим графа управления	space	agv	V	отсутствует	отсутствует
определить как блок данных	d	Cd [size]	dd,db,dw,dW	отсутствует	отсутствует
определить как блок кода	c	C- [size]	d- or du	отсутствует	отсутствует
определить как неопределенный блок	u	C- [size]	d- or du	отсутствует	отсутствует
определить как строку	A	Cs [размер]	ds	отсутствует	отсутствует
определить как структуру	Alt+Q	Cf [размер]	dF	отсутствует	отсутствует
Отладчик					
начать / продолжить процесс	F9	dc	F9	r и c	g
завершить процесс	Ctrl+F2	dk 9	?	kill	q
отсоединиться	?	o -	?	detach	
step into	F7	ds	s	n	t
step into 4	?	ds 4	F7	n 4	t 4
инструкции					
step over	F8	dso	S	s	p
step until указанный адрес	?	dsu <addr>	?	s	g
исполнять до оператора return	Ctrl+F7	dcr	?	finish	gu
выполнять до курсора	F4	#249	#249	отсутствует	отсутствует
показать стек	?	dbt	?	bt	
показать регистр	в окне регистров	dr all	автоматом в визуальном режиме	info r	registers
распечатать eax	в окне регистров	dr?eax	автоматом в визуальном режиме	info r	rax registers eax
показать старое состояние всех регистров	?	dro	?	?	?
показать что по function addr + N	?	afi \$\$ - показать информацию о функции в текущем смещении (\$\$)	?	?	?
отображать состояние кадра	?	pxw rbp-rsp@rsp	?	i f	?
исполнять до тех пор, пока условие не станет истинным	?	dsi	?	?	?
присвоить значение регистру	?	dr rip=0x456	?	set r	\$rip=0x456=456

Команда	IDA Pro	Программа radare2	r2 (визуальный режим)	GDB	WinDbg
Дизассемблирование					
дизассемблирование далее	отсутствует	pd	Vp	disas	uf, u
дизассемблировать N инструкций	отсутствует	pd X	Vp	x/i	u <addr> LX
дизассемблировать N в обратном направлении	отсутствует	pd -X	Vp	disas <a-o> <a>	ub
Информация о секциях					
секции/регионы	Menu sections	iS или S (добавить j для json)	отсутствует	maint info sections	!address
Загрузка файла символов					
секции/регионы	pdb menu	asm.dwarf.file, pdb.XX)	отсутствует	add-symbol-file	r
Стек					
стек	отсутствует	dbt	отсутствует	bt	k
стек в Json	отсутствует	dbtj	отсутствует	bt	k
частичное отображение стека (внутренняя часть)	отсутствует	dbt (dbg.btdepth dbg.btalgo)	отсутствует	bt -	
частичное отображение стека (внешняя часть)	отсутствует	dbt (dbg.btdepth dbg.btalgo)	отсутствует	thread ~* k apply all bt	
стеки всех нитей	отсутствует	dbt@t	отсутствует		
Точки останова					
список точек останова	Ctrl+Alt+B	db	?	info breakpoints	bl
добавление точки останова	F2	db [offset]	F2	break	bp
Нити					
Переключится в нить	Thread menu	dp	отсутствует	thread <N>	
Фреймы стека					
номера фреймов	отсутствует	?	отсутствует	any bt	kn
выбор фрейма	отсутствует	?	отсутствует	command frame	.frame
Формальные/Локальные переменные					
отображение формальных параметров	отсутствует	afv	отсутствует	info args	/t /i /V
отображение локальных параметров	отсутствует	afv	отсутствует	info locals	/t /i /V
сохранить параметры/локальные объекты в json	отсутствует	afvj	отсутствует	info locals	/t /i /V
список адресов, где есть доступ к переменным (R/W)	отсутствует	afvR/afvW	отсутствует	?	?

Команда	IDA Pro	Программа radare2	r2 (визуальный режим)	GDB	WinDbg
Поддержание проекта					
открыть проект		Po [file]		?	
сохранить проект	automatic	Ps [file]		?	
информация о проекте		Pi [file]		?	
Разное					
Дамп массива в виде символов	отсутствует	pc? (json, C, char, etc.)	Vppp	x/bc	db
настройки	option menu	e?	e		
поиск	search menu	/?	выбор зоны курсором С затем /		s

Эквивалент команды gdb ``set-follow-fork-mode``

Делается с помощью двух команд:

1. dc f - пока не произойдет форк,
2. затем используйте dP - выбор процесса для отладки.

Общие функции

- r2 принимает подписи FLIRT,
- r2 подключается к GDB, LLVM и WinDbg,
- r2 может писать/исправлять загруженный файл,
- в r2 встроены fortune [s], пасхальные [/s] и стальные яйца,
- r2 выполняет загрузку слепков (core dump) файлов ELF и MDMP (минидампы Windows).

Регистры

Регистры являются частью среды пользователя, они хранятся в одной из структур среды (контексте), используемой планировщиком. Эту структуру можно изменять для получения и установки значений регистров, и, например, на хостах Intel можно напрямую манипулировать даже аппаратными регистрами DR0-DR7 для установки аппаратных точек останова.

Есть разные команды для получения значений регистров. Для регистров общего назначения используют

```
[0x4A13B8C0]> dr
r15 = 0x00000000
r14 = 0x00000000
r13 = 0x00000000
r12 = 0x00000000
rbp = 0x00000000
rbx = 0x00000000
r11 = 0x00000000
r10 = 0x00000000
r9 = 0x00000000
r8 = 0x00000000
rax = 0x00000000
rcx = 0x00000000
rdx = 0x00000000
rsi = 0x00000000
rdi = 0x00000000
oeax = 0x0000003b
rip = 0x7f20bf5df630
rsp = 0x7fff515923c0

[0x7f0f2dbe630]> dr rip ; get value of 'rip'
0x7f0f2dbe630

[0x4A13B8C0]> dr rip = esp ; set 'rip' as esp
```

Взаимодействие между плагином и ядром осуществляется командами, возвращающими инструкции radare. Это используется, например, для установки флагов в ядре и значений регистров.

```
[0x7f0f2dbe630]> dr*      ; Добавление '*' покажет команды radare
f r15 1 0x0
f r14 1 0x0
f r13 1 0x0
```

```

f r12 1 0x0
f rbp 1 0x0
f rbx 1 0x0
f r11 1 0x0
f r10 1 0x0
f r9 1 0x0
f r8 1 0x0
f rax 1 0x0
f rcx 1 0x0
f rdx 1 0x0
f rsi 1 0x0
f rdi 1 0x0
f oeax 1 0x3b
f rip 1 0x7fff73557940
f rflags 1 0x200
f rsp 1 0x7fff73557940

[0x4A13B8C0]> .dr* ; include common register values in flags

```

Старая копия регистров сохраняется на все время отладки, она позволяет отслеживать изменения, сделанные во время выполнения анализируемой программы. Доступ к копии можно получить с помощью `oregs`.

```

[0x7f1fab84c630]> dro
r15 = 0x00000000
r14 = 0x00000000
r13 = 0x00000000
r12 = 0x00000000
rbp = 0x00000000
rbx = 0x00000000
r11 = 0x00000000
r10 = 0x00000000
r9 = 0x00000000
r8 = 0x00000000
rax = 0x00000000
rcx = 0x00000000
rdx = 0x00000000
rsi = 0x00000000
rdi = 0x00000000
oeax = 0x0000003b
rip = 0x7f1fab84c630
rflags = 0x00000200
rsp = 0x7fff386b5080

```

Текущее состояние регистров

```

[0x7f1fab84c630]> dr
r15 = 0x00000000
r14 = 0x00000000
r13 = 0x00000000
r12 = 0x00000000
rbp = 0x00000000
rbx = 0x00000000
r11 = 0x00000000
r10 = 0x00000000
r9 = 0x00000000
r8 = 0x00000000
rax = 0x00000000
rcx = 0x00000000
rdx = 0x00000000
rsi = 0x00000000
rdi = 0x7fff386b5080
oeax = 0xfffffffffffffff
rip = 0x7f1fab84c633
rflags = 0x00000202
rsp = 0x7fff386b5080

```

Значения, хранящиеся в eax, oeax и eip, изменились.

Если надо сохранять и восстанавливать значения регистров, сбрасывайте их дамп командой '`dr*`' на диск, а затем можно снова проинтерпретировать его:

```

[0x4A13B8C0]> dr* > regs.saved ; сохранить регистры
[0x4A13B8C0]> drp regs.saved ; восстановить их

```

Аналогичным образом меняются EFLAGS. Например, установка выбранных флагов:

```

[0x4A13B8C0]> dr eflags = pst
[0x4A13B8C0]> dr eflags = azsti

```

Получение информации об изменении регистров - команда `drd` (`diff registers`):

```
[0x4A13B8C0]> drd
oeax = 0x0000003b was 0x00000000 delta 59
rip = 0x7f00e71282d0 was 0x00000000 delta -418217264
rflags = 0x00000200 was 0x00000000 delta 512
rsp = 0x7ffffe85a09c0 was 0x00000000 delta -396752448
```

Карты памяти

Понимание и манипулирование картами памяти отлаживаемой программы важна для многих задач реверс-инженеринга. Radare2 предлагает богатый набор команд просмотра и изменения карт памяти в двоичном файле. Он включает в себя перечисление карт памяти отлаживаемого двоичного файла, удаление карт памяти, обработку загруженных библиотек и многое другое.

Посмотрим на инструкцию команды `dm`, отвечающей за просмотр и изменение карт памяти:

```
[0x55f2104cf620]> dm?
Usage: dm # Команды управления картами памяти
| dm                         Перечислить карты памяти процесса
| dm address size             Выделить <size> байт по <address>-у (где угодно, если адрес = -1) в дочернем проце...
| dm=                        Перечислить карты памяти процесса в виде ascii-art
| dm.                        Показать название карты, содержащей текущий адрес
| dm*                        Перечислить карты памяти в юрматре команд radare
| dm- address                Освободить карту памяти по <address>-у
| dmd[a] [file]              Сделать дамп текущего (всех) регионов отладочной карты в файл (from-to.dmp) (смотрите Sd)
| dmh[?]                     Показать карту кучи (heap)
| dmi [addr|libname] [symname] Перечислить символы в библиотеке
| dmi* [addr|libname] [symname] Перечислить символы в библиотеке в формате команд radare
| dmi.                       Список ближайших к текущему адресу символов
| dmiv                       Показать адреса заданного символа указанной библиотеки
| dmj                         Перечислить карты памяти в формате JSON
| dml <file>                 Загрузить содержимое файла в текущий регион карты памяти
| dmm[?][j*]                  Перечислить модули (библиотеки, двоичные файлы в памяти)
| dmp[?] <address> <size> <perms> Изменить страницу по <address>-у: изменить <size>, защиту <perms>
| dms[?] <id> <mapaddr>      Сделать снимок памяти
| dms- <id> <mapaddr>        Восстановить из снимка памяти
| dmS [addr|libname] [sectname] Перечислить секции заданной библиотеки
| dmS* [addr|libname] [sectname] Перечислить секции заданной библиотеки в формате команд radare
| dmL address size            Выделить <size> байт по <address>-у и объявить результат huge-страницей
```

В этой главе рассмотрим только наиболее полезные команды группы `dm`, используя простые примеры. Будем использовать программу `helloworld` для Linux, для других видов архитектур все будет выглядеть аналогично.

Сначала откройте программу в режиме отладки:

```
$ r2 -d helloworld
Process with PID 20304 started...
= attach 20304 20304
bin.baddr 0x56136b475000
Using 0x56136b475000
asm.bits 64
[0x7f133f022fb0]>
```

Обратите внимание, что мы подставили «`helloworld`» в `radare2` без «`./`». Программа `radare2` сначала попытается найти бинарик в текущем каталоге, а затем в `$PATH`. Это несколько противоречит идеям UNIX, но делает ее удобной для пользователей Windows.

Воспользуемся `dm` и распечатаем карты памяти открытого двоичного файла:

```
[0x7f133f022fb0]> dm
0x00000563a0113a000 - usr 4K s r-x /tmp/helloworld /tmp/helloworld ; map.tmp_helloworld.r_x
0x00000563a0133a000 - usr 8K s rw- /tmp/helloworld /tmp/helloworld ; map.tmp_helloworld.rw
0x000007f133f022000 * usr 148K s r-x /usr/lib/ld-2.27.so /usr/lib/ld-2.27.so ; map usr lib ld_2.27.so.r_x
0x000007f133f246000 - usr 8K s rw- /usr/lib/ld-2.27.so /usr/lib/ld-2.27.so ; map usr lib ld_2.27.so.rw
0x000007f133f248000 - usr 4K s rw- unk0 unk0 ; map.unk0.rw
0x000007ffd25ce000 - usr 132K s rw- [stack] [stack] ; map.stack_.rw
0x000007ffd25f6000 - usr 12K s r-- [vvar] [vvar] ; map.vvar_.r
0x000007ffd25f9000 - usr 8K s r-x [vdso] [vdso] ; map.vdso_.r_x
0xffffffffffff600000 - usr 4K s r-x [vsyscall] [vsyscall] ; map.vsyscall_.r_x
```

Для тех из вас, кто предпочитает более наглядный способ, можно использовать `dm=`, карты памяти отобразятся при помощи ASCII-art. Это удобно, если надо посмотреть расположение карт в оперативной памяти.

Для печати названия карты памяти для текущего смещения используйте `dm.` :

```
[0x7f133f022fb0]> dm.
0x000007f947eed9000 # 0x000007f947eef000 * usr 148K s r-x /usr/lib/ld-2.27.so /usr/lib/ld-2.27.so ; map usr lib ld_
```

Используя `dmm` можно «Получить список модулей (библиотек, двоичных файлов, загруженных в память)» - удобная команда для просмотра перечня загруженных модулей.

```
[0x7fa80a19dfb0]> dmm  
0x55ca23a4a000 /tmp/helloworld  
0x7fa80a19d000 /usr/lib/ld-2.27.so
```

Обратите внимание, что формат вывода команды `dm` и конкретно `dmm` будет разный для различных операционных систем и двоичных файлов.

Вместе с нашим бинарным файлом `helloworld` загружена динамическая библиотека `ld-2.27.so`. И мы пока не видим `libc`: radare2 останавливает исполнение перед загрузкой `libc` в память. Используем `dcu` (`debug continue until`) и выполним программу до ее точки входа, помеченной `radar`-ом как `entry0`.

```
[0x7fa80a19dfb0]> dcu entry0  
Continue until 0x55ca23a4a520 using 1 bpsize  
hit breakpoint at: 55ca23a4a518  
[0x55ca23a4a520]> dmm  
0x55ca23a4a000 /tmp/helloworld  
0x7fa809de1000 /usr/lib/libc-2.27.so  
0x7fa80a19d000 /usr/lib/ld-2.27.so
```

Теперь `libc-2.27.so` тоже загрузился, отлично!

Говоря о `libc`, популярной задачей для бинарного эксплоита является поиск адреса определенного символа в библиотеке. Имея эту информацию, можно создать экспloit, использующий ROP. Адрес можно получить при помощи команды `dmi`. Попробуем найти адрес `system()` в загруженном `libc`: выполним следующую команду:

```
[0x55ca23a4a520]> dmi libc system  
514 0x00000000 0x7fa809de1000 LOCAL FILE 0 system.c  
515 0x00043750 0x7fa809e24750 LOCAL FUNC 1221 do_system  
4468 0x001285a0 0x7fa809f095a0 LOCAL FUNC 100 svcerr_systemerr  
5841 0x001285a0 0x7fa809f095a0 LOCAL FUNC 100 svcerr_systemerr  
6427 0x00043d10 0x7fa809e24d10 WEAK FUNC 45 system  
7094 0x00043d10 0x7fa809e24d10 GLBAL FUNC 45 system  
7480 0x001285a0 0x7fa809f095a0 GLBAL FUNC 100 svcerr_systemerr
```

Подобно команде `dm`, используя `dmi`, получим ближайший символ к текущему адресу.

Еще одна полезная команда --- вывод списка секций динамической библиотеки. В следующем примере перечислены секции `ld-2.27.so`:

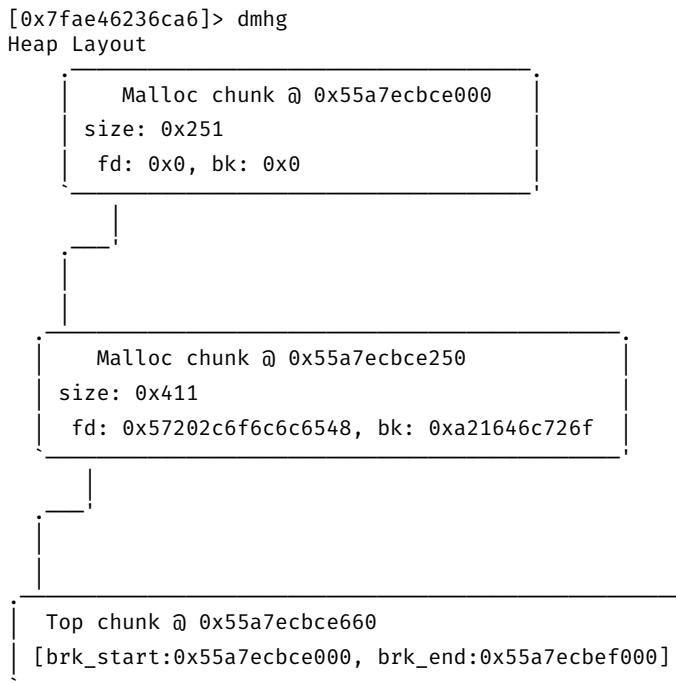
```
[0x55a7ebf09520]> dmS ld-2.27  
[Sections]  
00 0x00000000 0 0x00000000 0 ---- ld-2.27.so.  
01 0x000001c8 36 0x4652d1c8 36 -r-- ld-2.27.so..note.gnu.build_id  
02 0x000001f0 352 0x4652d1f0 352 -r-- ld-2.27.so..hash  
03 0x00000350 412 0x4652d350 412 -r-- ld-2.27.so..gnu.hash  
04 0x000004f0 816 0x4652d4f0 816 -r-- ld-2.27.so..dynsym  
05 0x00000820 548 0x4652d820 548 -r-- ld-2.27.so..dynstr  
06 0x00000a44 68 0x4652da44 68 -r-- ld-2.27.so..gnu.version  
07 0x00000a88 164 0x4652da88 164 -r-- ld-2.27.so..gnu.version_d  
08 0x00000b30 1152 0x4652db30 1152 -r-- ld-2.27.so..rela.dyn  
09 0x00000fb0 11497 0x4652dfb0 11497 -r-x ld-2.27.so..text  
10 0x0001d0e0 17760 0x4654a0e0 17760 -r-- ld-2.27.so..rodata  
11 0x00021640 1716 0x4654e640 1716 -r-- ld-2.27.so..eh_frame_hdr  
12 0x00021cf8 9876 0x4654ecf8 9876 -r-- ld-2.27.so..eh_frame  
13 0x00024660 2020 0x46751660 2020 -rw- ld-2.27.so..data.rel.ro  
14 0x00024e48 336 0x46751e48 336 -rw- ld-2.27.so..dynamic  
15 0x00024f98 96 0x46751f98 96 -rw- ld-2.27.so..got  
16 0x00025000 3960 0x46752000 3960 -rw- ld-2.27.so..data  
17 0x00025f78 0 0x46752f80 376 -rw- ld-2.27.so..bss  
18 0x00025f78 17 0x00000000 17 ---- ld-2.27.so..comment  
19 0x00025fa0 63 0x00000000 63 ---- ld-2.27.so..gnu.warning.llseek  
20 0x00025fe0 13272 0x00000000 13272 ---- ld-2.27.so..symtab  
21 0x000293b8 7101 0x00000000 7101 ---- ld-2.27.so..strtab  
22 0x0002af75 215 0x00000000 215 ---- ld-2.27.so..shstrtab
```

Куча

Команды radare2-а семейства `dm` отображают карту кучи (heap map) - полезная функция при проверке кучи и ее содержимого. Выполнение команды `dmh` показывает карту кучи:

```
[0x7fae46236ca6]> dmh  
Malloc chunk @ 0x55a7ecbce250 [size: 0x411][allocated]  
Top chunk @ 0x55a7ecbce660 - [brk_start: 0x55a7ecbce000, brk_end: 0x55a7ecbef000]
```

Просмотр компоновки кучи в виде графа:



В разделе `dmh` находятся команды для отображения информации о куче, полный список команд выводится так - `dmh?`.

```
[0x00000000]> dmh?
|Usage: dmh # Куча, отображение памяти
|dmh      Перечислить куски в сегменте кучи
|dmh [malloc_state] Перечислить куски в области (arena)
|dmha     Перечислить все экземпляры malloc_state в приложении
|dmhb     Показать все найденные двунаправленные списки кусков в main_arena-е
|dmhb [bin_num|bin_num:malloc_state]
|          Показать все найденные двунаправленные списки кусков в конкретной области
|dmhbgb [bin_num]   Показать график двунаправленных списков кусков в main_arena [в разработке]
|dmhc @[chunk_addr] Показать структуру malloc_chunk для заданного malloc-куска
|dmhf     Показать все найденные fastbin-ы в main_arena fastbinY-е
|dmhf [fastbin_num|fastbin_num:malloc_state]
|          Показать найденные двунаправленные списки в fastbinY-е из конкретной области
|dmhg     Показать график сегмента кучи
|dmhg [malloc_state] Показать график конкретной области
|dmhi @[malloc_state] Показать heap_info-структуры для заданной области
|dmhm     Показать все элементы структуры malloc_state основной нити (main_arena)
|dmhm [malloc_state] Перечислить все malloc_state-ы конкретной области
|dmht     Показать все найденные thead cache bins в tcache-е main_arena-ы
|dmh?     Показать инструкцию по построению карты кучи
```

Чтобы напечатать безопасные (safe-linked) списки (glibc >= 2.32) с разыменованными указателями, переменная `dbg.glibc.demangle` должна равняться истине.

Файлы

Отладчик radare2 позволяет составлять список файловых дескрипторов исследуемого процесса, а также манипулировать ими. Это полезная функция, отсутствующая в других отладчиках, ее возможности похожи на инструмент командной строки `lsof`, но есть и дополнительные функции для изменения смещения, закрытия или дублирования дескрипторов. В любой момент можно заменить дескрипторы файлов, например `stdio`, на сетевые сокеты, созданные `r2`, или заменить существующее сетевое сокетное соединение с целью его перехвата. Функция также доступна в `r2frida` при использовании команды `dd` с бэкслешем в качестве префикса. Программа `r2` предоставляет дополнительные инструкции --- комбинация `dd?`.

Отладка в обратном направлении

Radare2 включает функцию реверс-отладки - перебор адресов инструкций программы в обратном направлении (`reverse-next`, `reverse-continue`). Сначала надо сохранить состояние программы в нужный момент. Синтаксис команды, начинающей запись:

```
[0x004028a0]> dts+
```

Команды `dts` используются для записи состояний программ и управления ими. После записи состояний можно устанавливать смещения счетчика инструкций вперед и назад в любые точки после точки начала записи. Таким образом, можно

попробовать сделать один шаг назад:

```
[0x004028a0]> 2dso
[0x004028a0]> dr rip
0x004028ae
[0x004028a0]> dsb
continue until 0x004028a2
hit breakpoint at: 4028a2
[0x004028a0]> dr rip
0x004028a2
```

При запуске **dsb** обратный отладчик восстанавливает предыдущее записанное состояние и запускает программу с этого состояния до нужной точки. Можно попробовать продолжить в обратном направлении:

```
[0x004028a0]> db 0x004028a2
[0x004028a0]> 10dso
[0x004028a0]> dr rip
0x004028b9
[0x004028a0]> dcbs
[0x004028a0]> dr rip
0x004028a2
```

Команда **dcb** устанавливает счетчик программ на последнюю точку останова. Как только точка останова установлена, можно вернуться к ней в любое время. Просмотр списка записей состояний программы - **dts**:

```
[0x004028a0]> dts
session: 0    at:0x004028a0    ""
session: 1    at:0x004028c2    ""
```

ПРИМЕЧАНИЕ: Записи состояний можно сохранять в любое время. Они представлены в diff-формате, что позволяет сохранять только изменения памяти в сравнении с предыдущими версиями. Это экономит место в памяти, не надо хранить весь дамп.

Можно добавлять комментарии:

```
[0x004028c2]> dtsc 0 program start
[0x004028c2]> dtsc 1 decryption start
[0x004028c2]> dts
session: 0    at:0x004028a0    "program start"
session: 1    at:0x004028c2    "decryption start"
```

Можно делать заметки для каждой записи - ассоциировать с информацией в уме. Команды **dsb** и **dcb** восстанавливают состояние программы из последней записи при наличии списка таких записей.

Записи программ экспортируются в файлы и импортируются из них. Экспорт/импорт записей в/из файла:

```
[0x004028c2]> dtst records_for_test
Session saved in records_for_test.session and dump in records_for_test.dump
[0x004028c2]> dtstf records_for_test
session: 0, 0x4028a0 diffs: 0
session: 1, 0x4028c2 diffs: 0
```

Кроме того, можно выполнять реверс-отладку в режиме ESIL. В режиме ESIL состоянием программы можно управлять при помощи команд **aets**.

```
[0x00404870]> aets+
```

Шаг назад - **aesb**:

```
[0x00404870]> aer rip
0x00404870
[0x00404870]> 5aeso
[0x00404870]> aer rip
0x0040487d
[0x00404870]> aesb
[0x00404870]> aer rip
0x00404879
```

В дополнение к собственным возможностям реверс-отладки в radare2, также можно использовать протокол **gdb** для реверс-отладки на удаленном сервере (**gdbserver**), если сервер поддерживает эту функцию. Команды **!dsb** и **!dcb** заменяют собой команды **dsb** и **dcb** для этой цели, изучайте раздел удаленной отладки.

Сообщения Windows

В Windows во время отладки можно использовать **dbW**, задавая точку останова в обработчике сообщений для определенного окна.

Получение списка активных окон процесса - **dW**:

```
[0x7ffe885c1164]> dw
-----
| Handle | PID | TID | Class Name |
)-----(
| 0x0023038e | 9432 | 22432 | MSCTFIME UI
| 0x0029049e | 9432 | 22432 | IME
| 0x002c048a | 9432 | 22432 | Edit
| 0x000d0474 | 9432 | 22432 | msctls_statusbar32
| 0x00070bd6 | 9432 | 22432 | Notepad
`-----
```

Установка точки останова с заданным типом сообщения и именем класса окна или его дескриптором:

```
[0x7ffe885c1164]> dbW WM_KEYDOWN Edit
Breakpoint set.
```

ИЛИ

```
[0x7ffe885c1164]> dbW WM_KEYDOWN 0x002c048a
Breakpoint set.
```

Если вы не уверены, в какое окно следует поместить точку останова, используйте dwi, и указывайте окно при помощи мыши:

```
[0x7ffe885c1164]> dwi
Move cursor to the window to be identified. Ready? y
Try to get the child? y
-----
| Handle | PID | TID | Class Name |
)-----(
| 0x002c048a | 9432 | 22432 | Edit
`-----
```

Возможности удаленного доступа

Radare, как правило, запускается локально, но можно запускать серверный процесс и контролировать его локальным radare2. Реализация управления использует подсистему ввода-вывода radare, которая абстрагирует доступ к system(), cmd() и все основные операции ввода-вывода для работы по сети.

Справка по командам, используемым для организации удаленного доступа к radare:

```
[0x00405a04]> =?
Usage: [=!:+=ghH] [...] # Соединиться с другим процессом r2
```

команды удаленного доступа:

```
| =           пересилить все открытые соединения
| <[fd] cmd  послать вывод локальной команды на удаленный fd
| =[fd] cmd   запустить команду на удаленном 'fd' (по умолчанию - последний открытый)
| != cmd      запустить команду через r_io_system
| += [proto://]host:port соединиться с удаленным host:port (*rap://, raps://, tcp://, udp://, http://)
| -=[fd]       удалить все хосты или 'fd'
| ==[fd]       открыть удаленную сессию с хостом 'fd', 'q' -выход
| !==          запретить режим удаленной командной строки
| !=!          установка режима удаленной командной строки
```

серверы:

```
| .:9000      запуск одного сервера (echo x|nc :1 9090 или curl ::1:9090/cmd/x)
| :=port     запуск rap-сервера (o rap://9999)
| =g[?]      запуск gdbserver-a
| =h[?]      запуск http-вебсервера
| =H[?]      запуск http-вебсервера вместе с браузером
```

другие:

```
| =&:port    запуск rap-сервера в фоновом режиме (аналогично '&_h')
| =:host:port cmd  запуск команды 'cmd' на удаленном сервере
```

примеры:

```
| +=tcp://localhost:9090/    соединиться: r2 -c.:9090 ./bin
| +=rap://localhost:9090/    соединиться: r2 rap://:9090
| +=http://localhost:9090/cmd/ соединиться: r2 -c'=h 9090' bin
| o rap://:9090/           запуск rap-сервера на tcp-порту 9090
```

Инструкции по удаленным возможностям radare2 отображаются списком поддерживаемых плагинов ввода-вывода: radare2 -L.

Типичный удаленный сеанс выглядит следующим образом:

На удаленном хосте 1:

```
$ radare2 rap://:1234
На удаленном хосте 2:
$ radare2 rap://:1234
На локальном хосте:
$ radare2 -
Добавление хостов
[0x004048c5]> += rap://<host1>:1234//bin/ls
Connected to: <host1> at port 1234
waiting... ok

[0x004048c5]> =
0 - rap://<host1>:1234//bin/ls
```

Можно открывать удаленные файлы в режиме отладки (или с помощью любого подключаемого модуля ввода-вывода), указывая URI при добавлении хостов:

```
[0x004048c5]> += += rap://<host2>:1234/dbg:///bin/ls
Connected to: <host2> at port 1234
waiting... ok
0 - rap://<host1>:1234//bin/ls
1 - rap://<host2>:1234/dbg:///bin/ls
```

Выполнение команд на хосте 1:

```
[0x004048c5]> =0 px
[0x004048c5]> = s 0x666
```

Открыть сеанс связи с хостом 2:

```
[0x004048c5]> ==1
fd:6> pi 1
...
fd:6> q
```

Удаление узлов и закрытие подключения:

```
[0x004048c5]> ==-
```

Можно также перенаправлять вывод radare на TCP- или UDP-сервер, например, при помощи `nc -l`. Сначала добавьте сервер при помощи `+= tcp://` или `+= udp://`, затем можно перенаправить выходные данные команды на сервер:

```
[0x004048c5]> += tcp://<host>:<port>
Connected to: <host> at port <port>
5 - tcp://<host>:<port>/
[0x004048c5]> =<5 cmd...
```

Команда `=<` отправит вывод команды `cmd` на удаленное подключение с номером `N` или на последнее подключение, если идентификатор не указан.

Отладка при помощи gdbserver-a

Radare2 позволяет удаленно отлаживать по протоколу gdb. Запускаем gdbserver и подключаемся к нему с помощью radare2. Синтаксис подключения:

```
$ r2 -d gdb://<host>:<port>
```

Обратите внимание, что следующая команда делает то же самое, r2 использует подключаемый модуль отладки, указанный в uri, если такой есть.

```
$ r2 -D gdb gdb://<host>:<port>
```

Плагин отладки можно изменить во время выполнения с помощью команд `dL` и `Ld`.

Если gdbserver работает в расширенном режиме, можно подключиться прямо к процессу на хосте:

```
$ r2 -d gdb://<host>:<port>/<pid>
```

Также возможно начать отладку после анализа файла с помощью команды `doof`, которая перебазирует (rebase) данные текущего сеанса при открытии gdb.

```
[0x00404870]> doof gdb://<host>:<port>/<pid>
```

После подключения можно использовать стандартные команды отладки r2.

Radare2 еще не умеет загружать символы из gdbserver, копия двоичного файла должна быть локально представлена для загрузки символов. В случае, если символы не загружены даже если двоичный файл присутствует, можно попробовать указать путь с помощью `e dbg.exe.path`:

```
$ r2 -e dbg.exe.path=<path> -d gdb://<host>:<port>
```

Если символы загружаются по неправильному базовому адресу, можно попробовать указать и базовый адрес `e bin.baddr`:

```
$ r2 -e bin.baddr=<baddr> -e dbg.exe.path=<path> -d gdb://<host>:<port>
```

Обычно gdbserver сообщает максимальный поддерживаемый размер пакета. В противном случае radare2 использует разумные значения по умолчанию. Можно указать максимальный размер пакета в переменной окружения `R2_GDB_PKTSZ`. Можно узнать и установить максимальный размер пакета во время сеанса с плагином ввода-вывода, `=!`.

```
$ export R2_GDB_PKTSZ=512
$ r2 -d gdb://<host>:<port>
= attach <pid> <tid>
Assuming filepath <path/to/exe>
[0x7ff659d9fcc0]> =!pktsz
packet size: 512 bytes
[0x7ff659d9fcc0]> =!pktsz 64
[0x7ff659d9fcc0]> =!pktsz
packet size: 64 bytes
```

Плагин ввода-вывода gdb предоставляет полезные команды, которые могут не подходить ни для одной стандартной команды rdare2. Получить список этих команд можно с помощью `=!?`. (Напомним, `=!` получает доступ к базовому плагину ввода-вывода `system()`).

```
[0x7ff659d9fcc0]> =!?
Usage: =!cmd args
=!pid          - показать pid
=!pkt s         - послать пакет 's'
=!monitor cmd   - закодировать в шестнадцатеричный вид команду монитора и передать ее интерпретатору
=!rd            - показать, есть ли возможность выполнять отладку в обратном направлении
=!dsb           - шаг назад
=!dcb           - продолжить в обратном направлении
=!detach [pid]  - отсоединиться от удаленного или локального процесса по pid
=!inv.reg       - сбросить кэш регистров
=!pktsz         - показать размер максимального использованного пакета
=!pktsz bytes   - установить размер максимального пакета в байтах
=!exec_file [pid] - показать файл, который был запущен в текущем или заданным по pid процессе
```

Заметим, что `=!dsb` и `=!dcb` доступны только в специальных реализациях gdbserver, таких, как гг Мозиллы, gdbserver по умолчанию не включает поддержку удаленной реверс-отладки. Используйте `=!rd`, чтобы посмотреть доступные возможности реверс-отладки.

Если есть интерес в отладке взаимодействия radare2-а с сервером gdbserver, полезно использовать `=!monitor set remote-debug 1` для включения журналирования пакетов протокола управления формата gdb консоли gdbserver-a, также `=!monitor set debug 1` - вывод сообщений отладки, приходящих от его консоли.

Radare2 также реализует собственный gdbserver:

```
$ r2 -
[0x00000000]> =g?
|Usage: =[g] [...] # сервер gdb
| gdbserver:
| =g port file [args]  взаимодействуя через 'port', отлаживать 'file', используя gdbserver
| =g! port file [args]  то же, что выше, но с выдачей сообщений протокола отладки (например, gdbserver --remote-debug)
```

Запускать его так:

```
$ r2 -
[0x00000000]> =g 8000 /bin/radare2 -
```

А затем подключайтесь к нему, как к любому gdbserver. Например, при помощи radare2:

```
$ r2 -d gdb://localhost:8000
```

Отладка в режиме ядра с WinDBG KD

Поддержка интерфейса WinDBG KD в r2 позволяет подключаться к работающей виртуальной машине Windows и отлаживать ее ядро через последовательный порт или сетевое соединение.

Также можно использовать удаленный интерфейса GDB для подключения и отладки ядер Windows вне зависимости от инструментов Windows.

Имейте в виду, что поддержка WinDBG KD все еще находится в стадии разработки, сейчас это базовая реализация, которая со временем станет лучше.

Настройка KD в Windows

Пошаговое руководство см. в документации корпорации Майкрософт.

Последовательный порт

Подключение KD через последовательный порт в Windows Vista и выше осуществляется следующим образом:

```
bcdedit /debug on  
bcdedit /dbgsettings serial debugport:1 baudrate:115200
```

Так это делается для Windows XP: Открыть boot.ini и добавить /debug /debugport=COM1 /baudrate=115200:

```
[boot loader]  
timeout=30  
default=multi(0)disk(0)rdisk(0)partition(1)\WINDOWS  
[operating systems]  
multi(0)disk(0)rdisk(0)partition(1)\WINDOWS="Debugging with Cable" /fastdetect /debug /debugport=COM1 /baudrate=57600
```

В случае VMWare:

```
Virtual Machine Settings -> Add -> Serial Port  
Device Status:  
[v] Connect at power on  
Connection:  
[v] Use socket (named pipe)  
[_/tmp/winkd.pipe_____]  
From: Server To: Virtual Machine
```

Настройка машины VirtualBox:

```
Preferences -> Serial Ports -> Port 1  
[v] Enable Serial Port  
Port Number: [_COM1_____ [v]]  
Port Mode: [_Host_Pipe__ [v]]  
[v] Create Pipe  
Port/File Path: [_/tmp/winkd.pipe____]
```

Создание виртуальной машины при помощи qemu:

```
$ qemu-system-x86_64 -chardev socket,id=serial0,\  
path=/tmp/winkd.pipe,nowait,server \  
-serial chardev:serial0 -hda Windows7-VM.vdi
```

Сеть

Подключение KD по сети (KDNet) в Windows 7 или более поздней версии выглядит следующим образом:

```
bcdedit /debug on  
bcdedit /dbgsettings net hostip:w.x.y.z port:n
```

Начиная с Windows 8, принудительно выполнить отладку в каждой загрузке невозможно, но всегда можно входить в расширенные параметры загрузки и там включать отладку ядра:

```
bcdedit /set {globalsettings} advancedoptions true
```

Подключение к интерфейсу KD на r2

Последовательный порт

Radare2 использует плагин ввода-вывода `winkd` для подключения к файлу сокета, создаваемому virtualbox-ом или qemu. Для плагина отладчика `winkd` нужно также указать x86-32. (поддерживаются 32- и 64-разрядная отладка)

```
$ r2 -a x86 -b 32 -D winkd winkd:///tmp/winkd.pipe
```

В Windows надо запустить следующую строку:

```
$ radare2 -D winkd winkd://\\.\pipe\com_1
```

Сеть

```
$ r2 -a x86 -b 32 -d winkd://<hostip>:<port>:w.x.y.z
```

Использование KD

При подключении к интерфейсу KD r2 отправит пакет разрыва для прерывания выполнения целевого процесса:

```
[0x828997b8]> pd 20
;-- eip:
0x828997b8 cc int3
0x828997b9 c20400 ret 4
0x828997bc cc int3
0x828997bd 90 nop
0x828997be c3 ret
0x828997bf 90 nop
```

Чтобы пропустить это прерывание, нужно изменить eip и запустить `dc` дважды:

```
dr eip=eip+1
dc
dr eip=eip+1
dc
```

Теперь виртуальная машина Windows снова будет интерактивной. Нам нужно будет убить r2 и присоединить его еще раз, чтобы перейти к управлению ядром.

Команда dr используется для перечисления всех процессов, а dpa или dr= для присоединения к процессу. Будет отображены базовые адреса процессов в карте физической памяти.

WinDBG бэкенд для Windows (DbgEng)

В Windows radare2 может использовать DbgEng.dll в качестве серверной части отладки, позволяя ему использовать возможности WinDBG, включая поддержку файлов дампа памяти, отладку режимов пользователя и ядра локально и удаленно.

Можно использовать DLL-библиотеки отладки, включенные в Windows, или загрузить последние версии со страницы загрузки Майкрософт, что рекомендуется сделать.

Использовать библиотеки DLL из папки приложения WinDbg Preview Microsoft Store напрямую **нельзя**, поскольку они не помечены как исполняемые для обычных пользователей.

radare2 попытается сначала загрузить dbgeng.dll из директория, указанного в переменной среды _NT_DEBUGGER_EXTENSION_PATH, перед тем как использовать путь поиска библиотек Windows по умолчанию.

Использование плагина

Чтобы использовать плагин windbg задавайте те же параметры командной строки, что и для WinDBG или kd (см. документацию Microsoft), добавляя кавычки и экранируя ``ами при необходимости:

```
> r2 -d "windbg:// -remote tcp:server=Server, port=Socket"
> r2 -d "windbg:// MyProgram.exe \"my arg\""
> r2 -d "windbg:// -k net:port=<n>, key=<MyKey>"
> r2 -d "windbg:// -z MyDumpFile.dmp"
```

Затем можно выполнять отладку как обычно (см. команду d?) или взаимодействовать с командной строкой на сервере непосредственно с помощью команды =!:

```
[0x7ffcac9fcea0]> dcu 0x0007ffc98f42190
Continue until 0x7ffc98f42190 using 1 bpsize
ModLoad: 00007ffc`ab6b0000 00007ffc`ab6e0000 C:\WINDOWS\System32\IMM32.DLL
Breakpoint 1 hit
hit breakpoint at: 0x7ffc98f42190

[0x7ffffcf232190]> =!k4
Child-SP RetAddr Call Site
00000033`73b1f618 00007ff6`c67a861d r_main!r_main_radare2
00000033`73b1f620 00007ff6`c67d0019 radare2!main+0x8d
00000033`73b1f720 00007ff6`c67cfbe radare2!invoke_main+0x39
00000033`73b1f770 00007ff6`c67cfde radare2!__scrt_common_main_seh+0x12e
```

Инструменты

Radare2 - это не единственный инструмент, предоставляемый проектом radare2. Остальные главы в этой книге сосредоточены на представлении инструментов radare2, эта глава будет сосредоточена на объяснении сопутствующих инструментов.

Все функциональные возможности, предоставляемые различными API и плагинами, включают различные инструменты, позволяющие легко использовать их из командной строки и легко интегрировать их со скриптами ОС.

Благодаря ``ортогональному'' дизайну фреймворка, можно делать все то, что умеет r2, при помощи следующих классов инструментов:

- сопутствующие инструменты,
- API нативных библиотек,
- запуск сценариев с помощью r2pipe,
- оболочка R2.

Программа Rax2

Утилита `rax2` - часть пакета radare, она является простым вычислителем выражений для командной строки ОС. Программа полезна для выполнения базовых преобразований между значениями с плавающей запятой, шестнадцатеричными представлениями, шестнадцатеричными строками в ascii, восьмеричных чисел в другие системы счисления. Поддерживается порядок байтов и запускает свой интерпретатор командной строки, если аргументы не приведены. В командной строке `rax2` может считывать значения из `stdin`, поэтому его можно использовать как многобазовый калькулятор значений во входном потоке.

Внутри r2 функциональность `rax2` доступна в группе команд `? <выражение>`. Например:

```
[0x00000000]> ? 3+4
```

Как видите, числовые выражения могут содержать математические выражения, такие как сложение, вычитание, ... а также операции со скобками. Синтаксис, в котором представлены числа, определяет систему счисления, например:

- 3 : decimal, base 10
- 0xface : hexadecimal, base 16
- 0472 : octal, base 8
- 2M : units, 2 megabytes
- ...

Справка доступна по `rax2 -h`, она покажет вам еще кучу синтаксических элементов

```
$ rax2 -h
Usage: rax2 [options] [expr ...]
=[base] ; rax2 =10 0x46 -> output in base 10
int    -> hex ; rax2 10
hex    -> int ; rax2 0xa
-int   -> hex ; rax2 -77
-hex   -> int ; rax2 0xfffffffffb3
int    -> bin ; rax2 b30
int    -> ternary ; rax2 t42
bin    -> int ; rax2 1010d
ternary -> int ; rax2 1010dt
float  -> hex ; rax2 3.33f
hex    -> float ; rax2 Fx40551ed8
oct    -> hex ; rax2 350
hex    -> oct ; rax2 0x12 (0 is a letter)
bin    -> hex ; rax2 1100011b
hex    -> bin ; rax2 Bx63
ternary -> hex ; rax2 212t
hex    -> ternary ; rax2 Tx23
raw    -> hex ; rax2 -S < /binfile
hex    -> raw ; rax2 -s 414141
-l     ; добавить перевод строки в вывод (для -E/-D/-r/..
-a     show ascii table ; rax2 -a
-b     bin -> str ; rax2 -b 01000101 01110110
-B     str -> bin ; rax2 -B hello
-d     force integer ; rax2 -d 3 -> 3 instead of 0x3
-e     swap endianness ; rax2 -e 0x33
-D     base64 decode ;
-E     base64 encode ;
-f     floating point ; rax2 -f 6.3+2.1
-F     stdin slurp code hex ; rax2 -F < shellcode.[c/py/js]
-h     help ; rax2 -h
-i     dump as C byte array ; rax2 -i < bytes
-k     keep base ; rax2 -k 33+3 -> 36
-K     randomart ; rax2 -K 0x34 1020304050
-L     bin -> hex(bignum) ; rax2 -L 111111111 # 0x1ff
-n     binary number ; rax2 -n 0x1234 # 34120000
-N     binary number ; rax2 -N 0x1234 # \x34\x12\x00\x00
-r     r2 style output ; rax2 -r 0x1234
-s     hexstr -> raw ; rax2 -s 43 4a 50
-S     raw -> hexstr ; rax2 -S < /bin/ls > ls.hex
```

```

-t      tstamp -> str      ;  rax2 -t 1234567890
-x      hash string        ;  rax2 -x linux osx
-u      units              ;  rax2 -u 389289238 # 317.0M
-w      signed word        ;  rax2 -w 16 0xffff
-v      version            ;  rax2 -v

```

Примеры:

```

$ rax2 3+0x80
0x83

$ rax2 0x80+3
131

$ echo 0x80+3 | rax2
131

$ rax2 -s 4142
AB

$ rax2 -S AB
4142

$ rax2 -S < bin.foo
...
$ rax2 -e 33
0x21000000
$ rax2 -e 0x21000000
33

$ rax2 -K 90203010
+--[0x10302090]---+
|Eo. .
|  . . .
|    o
|    .
|    S
+-----+

```

Программа rafind2

Rafind2 --- это командная строка библиотеки `r_search`. Она позволяет искать строки, последовательности байтов с двоичными масками и т.д.

```

$ rafind2 -h
Usage: rafind2 [-mXnzZhqv] [-a align] [-b sz] [-f/t from/to] [-[e|s|S] str] [-x hex] -[file|dir] ..
-a [align] принимать только выровненные хиты
-b [size] установить размер блока
-e [regex] поиск соответствий regex (может быть использовано несколько раз)
-f [from] начать поиск с адреса 'from'
-h показать это сообщение
-i идентифицировать тип файла (r2 -пqcрт файл)
-j выводить в формате JSON
-m поиск магических последовательностей, идентификация типа файла
-M [str] задать бинарную маску, применяемую к ключевым словам
-n не останавливаться при наличии ошибок чтения
-r выводить результат в виде команд radare
-s [str] поиск заданной строки (может быть использовано несколько раз)
-S [str] поиск заданной wide-строки (может быть использовано несколько раз), предполагается кодировка UTF-
8.
-t [to] остановить поиск на адресе 'to'
-q тихий режим, не показывать заголовки в результатах (имена файлов), совпадающий контент (по умолчанию для
-v напечатать версию и выйти
-x [hex] поиск шнадцатеричных строк (909090) (может быть использовано несколько раз)
-X выводить результат в виде шестнадцатеричного дампа
-z поиск строк, заканчивающихся нулем
-Z показывать строки для каждого хита

```

Вот как его использовать: сначала найдем ``lib'' внутри двоичного файла `/bin/ls`.

```

$ rafind2 -s lib /bin/ls
0x5f9
0x675
0x679
...
$
```

Обратите внимание, что выходные данные довольно минималистичны и показывают смещения там, где находится строка `lib`. Можно использовать этот вывод для подачи других инструментов.

Результаты подсчета:

```
$ rafind2 -s lib /bin/ls | wc -l
```

Отображение результатов с контекстом:

```
$ export F=/bin/ls
$ for a in `rafind2 -s lib $F` ; do \
    r2 -ns $a -qc'x 32' $F ; done
0x000005f9 6c69 622f 6479 6c64 .. lib/dyld.....
0x00000675 6c69 622f 6c69 6275 .. lib/libutil.dylib
0x00000679 6c69 6275 7469 6c2e .. libutil.dylib...
0x00000683 6c69 6200 000c 0000 .. lib.....8.....
0x000006a5 6c69 622f 6c69 626e .. lib/libncurses.5
0x000006a9 6c69 626e 6375 7273 .. libncurses.5.4.d
0x000006ba 6c69 6200 0000 0c00 .. lib.....8.....
0x000006dd 6c69 622f 6c69 6253 .. lib/libSystem.B.
0x000006e1 6c69 6253 7973 7465 .. libSystem.B.dylib
0x000006ef 6c69 6200 0000 0000 .. lib.....&.....
```

Rafind2 также может быть использован в качестве замены `file` для идентификации mime-type файла с помощью внутренней базы данных магических последовательностей radare2.

```
$ rafind2 -i /bin/ls
0x00000000 1 Mach-O
```

Также утилита работает как замена `strings` аналогично rabin2 -z, но без анализа заголовков и двоичных разделов.

```
$ rafind2 -z /bin/ls | grep http
0x000076e5 %http://www.apple.com/appleca/root.crl0\r
0x00007ae6 https://www.apple.com/appleca/0
0x00007fa9 )http://www.apple.com/certificateauthority0
0x000080ab $http://crl.apple.com/codesigning.crl0
```

Программа Rarun2

Rarun2 - инструмент, позволяющий настроить специальную среду выполнения - переопределять `stdin/stdout`, каналы, переменные среды, а также параметров, полезных для формирования ограниченной среды, в которой предполагается выполнить двоичный файл, включая режим отладки.

```
$ rarun2 -h
Usage: rarun2 -v|-t|script.rr2 [directive ...]
```

Rarun2 используется как в качестве отдельного инструмента, так и в составе radare2. Задания среды исполнения (профиля) используется текстовый файл в формате ключ=значение. Формат профиля очень прост. Наиболее важные ключи - `program` и `arg*`. Профиль rarun2 можно загружать и в radare2. Для загрузки профиля из файла необходимо использовать флаг `-r`, флаг `-R` позволяет задавать директивы в командной строке.

Один из наиболее распространенных вариантов использования - перенаправление вывода отлаживаемой программы в radare2. Для этого нужно использовать `stdio`, `stdout`, `stdin`, `input` и пару аналогичных ключей.

Вот пример профиля:

```
program=/bin/ls
arg1=/bin
# arg2=hello
# arg3="hello\nworld"
# arg4=:048490184058104849
# arg5=:!ragg2 -p n50 -d 10:0x8048123
# arg6=@arg.txt
# arg7=@300@ABCD # 300 chars filled with ABCD pattern
# system=r2 -
# aslr=no
setenv=FOO=BAR
# unsetenv=FOO
# clearenv=true
# envfile=environ.txt
timeout=3
# timeoutsig=SIGHUP # or 15
# connect=localhost:8080
# listen=8080
# pty=false
# fork=true
# bits=32
# pid=0
```

```

# pidfile=/tmp/foo.pid
# #sleep=0
# #maxfd=0
# #execve=false
# #maxproc=0
# #maxstack=0
# #core=false
# #stdio=blah.txt
# #stderr=foo.txt
# stdout=foo.txt
# stdin=input.txt # or !program to redirect input from another program
# input=input.txt
# chdir=
# chroot=/mnt/chroot
# libpath=$PWD:/tmp/lib
# r2preload=yes
# preload=/lib/libfoo.so
# setuid=2000
# seteuid=2000
# setgid=2001
# setegid=2001
# nice=5

```

Rabin2 --- свойства двоичного файла

Под этим именем radare скрывает мощный инструмент для обработки двоичных файлов, для получения информации об импорте, разделах, заголовках и других данных. Rabin2 может представить его в нескольких форматах, принятых другими инструментами, включая сам radare2. Rabin2 понимает многие форматы файлов: Java CLASS, ELF, PE, Mach-O или любой формат, поддерживаемый плагинами, и он может получать импорт / экспорт символов, зависимости библиотеки, строки разделов данных, внешние ссылки, адрес точки входа, разделы, тип архитектуры.

```

$ rabin2 -h
Usage: rabin2 [-AcdeEghHiIjLLMqrRsSvVxzZ] [-@ at] [-a arch] [-b bits] [-B addr]
               [-C F:C:D] [-f str] [-m addr] [-n str] [-N m:M] [-P[-P] pdb]
               [-o str] [-O str] [-k query] [-D lang symname] | file

-@ [addr]          показать секцию, символ или импорт по указанному адресу
-A                перечислить включенные бинарики и их битовые пары архитектуры
-a [arch]          установить архитектуру (x86, arm, .. or <arch>_<bits>)
-b [bits]          установить биты (32, 64 ...)
-B [addr]          заменить базовый адрес (pie bins)
-c                перечислить классы
-C [fmt:C:D]      создать [elf,mach0,re] с секциями Code и Data шеснадцатеричными кодами (see -a)
-d                показать отладочную/dwarf информацию
-D lang name      "распутать" имя символа (-D all для bin.demangle=true)
-e                точка входа
-E                глобально экспортируемые символы
-f [str]           выбрать строку в встроенный бинарике
-F [binfmt]        использовать насилино заданный бинарный плагин, игнорируя анализ заголовка
-g                то же что, и -SMZIHVRResizcld (показать всю информацию)
-G [addr]          загрузка адреса . смещения относительно заголовка
-h                это сообщение
-H                поля заголовка
-i                импорты (символы, импортированные из библиотек)
-I                информация о бинарике
-j                выводить в json
-k [sdb-query]    запустить запрос к sdb. пример: '*'
-K [algo]          вычислить контрольные суммы (md5, sha1, ...)
-l                прилинкованные библиотеки
-L [plugin]        перечислить поддерживаемые плагины и их детали
-m [addr]          показывать исходный код по адресу
-M                main (показать адрес символа main)
-n [str]           показать секцию, символ или импорт, поименованную строку
-N [min:max]      установить числа min:max в символах для строк (смотри -z и -zz)
-o [str]           вывод файла/папки для операций записи (выключено по умолчанию)
-O [str]           записать/экстрагировать операции (-O help)
-p                показать физический адрес
-P                показать отладочную/pdb информацию,
-PP               загрузить файл pdb.
-q                выдавать меньше информации
-qq              выдавать меньше информации (без смещений/размеров для -z, например)
-Q                показать адрес загрузки, используемый dlopen (не-aslr-библиотеки)
-r                вывода в формате radare
-R                релокации
-s                символы
-S                секции
-u                нефильтрованный вывод (без переименования символов-дубликатов и секций)
-v                показать версию и выйти

```

```

-V          показать версию бинарика
-x          экстрагировать бинарики, содержащиеся в файле
-X [fmt] [f] .. запаковать в fat или zip-файлы перечисленные файлы и бинарики, содержащиеся в файле
-z          строки (из сегмента данных)
-zz         строки (из raw-бинариков [e bin.rawstr=1])
-zzz        дамп raw-строк на стандартный вывод (для файлов большого размера)
-Z          попробовать угадать размер двоичной программы
.....

```

Идентификация свойств файла

Идентификация типа файла осуществляется с помощью **-I**. С помощью этого параметра rabin2 выводит информацию о двоичном типе, такую как кодировка, порядок байтов, класс, операционная система:

```

$ rabin2 -I /bin/ls
arch      x86
binsz    128456
bintype   elf
bits      64
canary    true
class     ELF64
crypto    false
endian   little
havecode  true
intrp    /lib64/ld-linux-x86-64.so.2
lang      c
linenum   false
lsyms     false
machine   AMD x86-64 architecture
maxopsz  16
minopsz  1
nx        true
os        linux
pcalign   0
pic       true
relocs    false
relro     partial
rpath    NONE
static    false
stripped  true
subsys   linux
va        true

```

Чтобы заставить rabin2 выдавать информацию в формате, понятном основной программе radare2, добавьте флаг **-Ir**:

```

$ rabin2 -Ir /bin/ls
e cfg.big endian=false
e asm.bits=64
e asm.dwarf=true
e bin.lang=c
e file.type=elf
e asm.os=linux
e asm.arch=x86
e asm.pc align=0

```

Точки ввода в код

Флаг **-e**, переданный rabin2, покажет точки входа для данного двоичного файла. Два примера:

```

$ rabin2 -e /bin/ls
[Entrypoints]
vaddr=0x000005310 paddr=0x000005310 baddr=0x00000000 laddr=0x00000000 haddr=0x00000018 type=program

1 entrypoints

$ rabin2 -er /bin/ls
fs symbols
f entry0 1 @ 0x000005310
f entry0_haddr 1 @ 0x000000018
s entry0

```

Импорты

Rabin2 способен находить импортированные объекты по исполняемому файлу, а также их смещения в своем PLT. Эта информация полезна, например, для понимания того, какая внешняя функция вызывается инструкцией **call**. Передайте флаг **-i** rabin2, чтобы получить список импортов. Пример:

```
$ rabin2 -i /bin/ls
[Imports]
nth vaddr      bind   type   lib name
-----
1 0x000032e0  GLOBAL  FUNC  __ctype_toupper_loc
2 0x000032f0  GLOBAL  FUNC  getenv
3 0x00003300  GLOBAL  FUNC  sigprocmask
4 0x00003310  GLOBAL  FUNC  __snprintf_chk
5 0x00003320  GLOBAL  FUNC  raise
6 0x00000000  GLOBAL  FUNC  free
7 0x00003330  GLOBAL  FUNC  abort
8 0x00003340  GLOBAL  FUNC  __errno_location
9 0x00003350  GLOBAL  FUNC  strncmp
10 0x00000000  WEAK   NOTYPE _ITM_deregisterTMCloneTable
11 0x00003360  GLOBAL  FUNC  localtime_r
12 0x00003370  GLOBAL  FUNC  _exit
13 0x00003380  GLOBAL  FUNC  strcpy
14 0x00003390  GLOBAL  FUNC  __fpending
15 0x000033a0  GLOBAL  FUNC  isatty
16 0x000033b0  GLOBAL  FUNC  sigaction
17 0x000033c0  GLOBAL  FUNC  iswcntrl
18 0x000033d0  GLOBAL  FUNC  wcswidth
19 0x000033e0  GLOBAL  FUNC  localeconv
20 0x000033f0  GLOBAL  FUNC  mbstowcs
21 0x00003400  GLOBAL  FUNC  readlink
...

```

Экспорты

Rabin2 способен находить экспорты. Пример:

```
$ rabin2 -E /usr/lib/libr_bin.so | head
[Exports]
nth  paddr      vaddr      bind   type size lib name
-----
210 0x000ae1f0 0x000ae1f0 GLOBAL  FUNC  200 r_bin_java_print_exceptions_attr_summary
211 0x000afc90 0x000afc90 GLOBAL  FUNC  135 r_bin_java_get_args
212 0x000b18e0 0x000b18e0 GLOBAL  FUNC  35 r_bin_java_get_item_desc_from_bin_cp_list
213 0x00022d90 0x00022d90 GLOBAL  FUNC  204 r_bin_class_add_method
214 0x000ae600 0x000ae600 GLOBAL  FUNC  175 r_bin_java_print_fieldref_cp_summary
215 0x000ad880 0x000ad880 GLOBAL  FUNC  144 r_bin_java_print_constant_value_attr_summary
216 0x000b7330 0x000b7330 GLOBAL  FUNC  679 r_bin_java_print_element_value_summary
217 0x000af170 0x000af170 GLOBAL  FUNC  65 r_bin_java_create_method_fq_str
218 0x00079b00 0x00079b00 GLOBAL  FUNC  15 LZ4_createStreamDecode

```

Символы (Экспорты)

В rabin2 формат списка сгенерированных символов аналогичен списку импорта. Используйте параметр **-S** для получения экспортов:

```
rabin2 -s /bin/ls | head
[Symbols]
nth paddr      vaddr      bind   type   size lib name
-----
110 0x000150a0 0x000150a0 GLOBAL FUNC  56 _obstack_allocated_p
111 0x0001f600 0x0021f600 GLOBAL OBJ   8 program_name
112 0x0001f620 0x0021f620 GLOBAL OBJ   8 stderr
113 0x00014f90 0x00014f90 GLOBAL FUNC  21 _obstack_begin_1
114 0x0001f600 0x0021f600 WEAK   OBJ   8 program_invocation_name
115 0x0001f5c0 0x0021f5c0 GLOBAL OBJ   8 alloc_failed_handler
116 0x0001f5f8 0x0021f5f8 GLOBAL OBJ   8 optarg
117 0x0001f5e8 0x0021f5e8 GLOBAL OBJ   8 stdout
118 0x0001f5e0 0x0021f5e0 GLOBAL OBJ   8 program_short_name

```

С опцией **-sr** rabin2 создает сценарий radare2. Позже он может быть передан в ядро, чтобы автоматически помечать все символы и определять соответствующие диапазоны байтов как функции и блоки данных.

```
$ rabin2 -sr /bin/ls | head
fs symbols
f sym._obstack_allocated_p 56 0x000150a0
f sym.program_invocation_name 8 0x0021f600
f sym.stderr 8 0x0021f620
f sym._obstack_begin_1 21 0x00014f90
f sym.program_invocation_name 8 0x0021f600
f sym._alloc_failed_handler 8 0x0021f5c0
f sym.optarg 8 0x0021f5f8
```

```
f sym.stdout 8 0x0021f5e8
f sym.program_invocation_short_name 8 0x0021f5e0
```

Список библиотек

Rabin2 перечисляет библиотеки, используемые двоичным файлом, параметр - `-l`:

```
$ rabin2 -l `which r2`
[Linked libraries]
libr_core.so
libr_parse.so
libr_search.so
libr_cons.so
libr_config.so
libr_bin.so
libr_debug.so
libr_anal.so
libr_reg.so
libr_bp.so
libr_io.so
libr_fs.so
libr_asm.so
libr_syscall.so
libr_hash.so
libr_magic.so
libr_flag.so
libr_egg.so
libr_crypto.so
libr_util.so
libpthread.so.0
libc.so.6

22 libraries
```

Давайте проверим вывод с помощью команды `ldd`:

```
$ ldd `which r2`
linux-vdso.so.1 (0x00007ffffba38e000)
libr_core.so => /usr/lib64/libr_core.so (0x00007f94b4678000)
libr_parse.so => /usr/lib64/libr_parse.so (0x00007f94b4425000)
libr_search.so => /usr/lib64/libr_search.so (0x00007f94b421f000)
libr_cons.so => /usr/lib64/libr_cons.so (0x00007f94b4000000)
libr_config.so => /usr/lib64/libr_config.so (0x00007f94b3dfa000)
libr_bin.so => /usr/lib64/libr_bin.so (0x00007f94b3af000)
libr_debug.so => /usr/lib64/libr_debug.so (0x00007f94b38d2000)
libr_anal.so => /usr/lib64/libr_anal.so (0x00007f94b2fdb000)
libr_reg.so => /usr/lib64/libr_reg.so (0x00007f94b2db4000)
libr_bp.so => /usr/lib64/libr_bp.so (0x00007f94b2ba000)
libr_io.so => /usr/lib64/libr_io.so (0x00007f94b2944000)
libr_fs.so => /usr/lib64/libr_fs.so (0x00007f94b270e000)
libr_asm.so => /usr/lib64/libr_asm.so (0x00007f94b1c69000)
libr_syscall.so => /usr/lib64/libr_syscall.so (0x00007f94b1a63000)
libr_hash.so => /usr/lib64/libr_hash.so (0x00007f94b185a000)
libr_magic.so => /usr/lib64/libr_magic.so (0x00007f94b164d000)
libr_flag.so => /usr/lib64/libr_flag.so (0x00007f94b1446000)
libr_egg.so => /usr/lib64/libr_egg.so (0x00007f94b1236000)
libr_crypto.so => /usr/lib64/libr_crypto.so (0x00007f94b1016000)
libr_util.so => /usr/lib64/libr_util.so (0x00007f94b0d35000)
libpthread.so.0 => /lib64/libpthread.so.0 (0x00007f94b0b15000)
libc.so.6 => /lib64/libc.so.6 (0x00007f94b074d000)
libr_lang.so => /usr/lib64/libr_lang.so (0x00007f94b0546000)
libr_socket.so => /usr/lib64/libr_socket.so (0x00007f94b0339000)
libm.so.6 => /lib64/libm.so.6 (0x00007f94afffaf000)
libdl.so.2 => /lib64/libdl.so.2 (0x00007f94afda0000)
/lib64/ld-linux-x86-64.so.2 (0x00007f94b4c79000)
libssl.so.1.0.0 => /usr/lib64/libssl.so.1.0.0 (0x00007f94afb3c000)
libcrypto.so.1.0.0 => /usr/lib64/libcrypto.so.1.0.0 (0x00007f94af702000)
libutil.so.1 => /lib64/libutil.so.1 (0x00007f94af4ff000)
libz.so.1 => /lib64/libz.so.1 (0x00007f94af2e8000)
```

Если сравнить выходы `rabin2 -l` и `ldd` видно, что `rabin2` перечисляет меньше библиотек, чем `ldd`. Причина в том, что `rabin2` не следует и не показывает зависимостей библиотек. Отображаются только прямые двоичные зависимости.

Строки

Параметр `-Z` используется для перечисления читаемых строк, найденных в разделе ```.rodata''` двоичных файлов ELF и в разделе `.text` файлов PE. Пример:

```
$ rabin2 -z /bin/ls | head
[Strings]
nth paddr      vaddr      len size section type  string
-----
000 0x000160f8 0x000160f8 11 12 (.rodata) ascii dev_ino_pop
001 0x00016188 0x00016188 10 11 (.rodata) ascii sort_files
002 0x00016193 0x00016193 6 7 (.rodata) ascii posix-
003 0x0001619a 0x0001619a 4 5 (.rodata) ascii main
004 0x00016250 0x00016250 10 11 (.rodata) ascii ?pcdb-lswd
005 0x00016260 0x00016260 65 66 (.rodata) ascii # Configuration file for dircolors, a utility to help you set the
006 0x000162a2 0x000162a2 72 73 (.rodata) ascii # LS_COLORS environment variable used by GNU ls with the --
color option.
007 0x000162eb 0x000162eb 56 57 (.rodata) ascii # Copyright (C) 1996-2018 Free Software Foundation, Inc.
008 0x00016324 0x00016324 70 71 (.rodata) ascii # Copying and distribution of this file, with or without modificat
009 0x0001636b 0x0001636b 76 77 (.rodata) ascii # are permitted provided the copyright notice and this notice are
```

С параметром **-zr** эта информация представлена в виде списка команд radare2. Его можно использовать в сеансе radare2 для автоматического создания пространства флагов под названием «строки», предварительно заполненного флагами для всех строк, найденных rabin2. Кроме того, этот сценарий будет помечать соответствующие диапазоны байтов как строки, а не код.

```
$ rabin2 -zr /bin/ls | head
fs stringsf str.dev_ino_pop 12 @ 0x000160f8
Cs 12 @ 0x000160f8
f str.sort_files 11 @ 0x00016188
Cs 11 @ 0x00016188
f str.posix 7 @ 0x00016193
Cs 7 @ 0x00016193
f str.main 5 @ 0x0001619a
Cs 5 @ 0x0001619a
f str.pcdb_lswd 11 @ 0x00016250
Cs 11 @ 0x00016250
```

Секции программ

Rabin2, вызываемый с параметром **-S**, дает полную информацию о разделах исполняемого файла. Для каждой секции отображаются индекс, смещение, размер, выравнивание, тип и разрешения. Пример показывает следующее:

```
$ rabin2 -S /bin/ls
[Sections]
nth paddr      size vaddr      vsiz perm name
-----
00 0x00000000    0 0x00000000    0 ----
01 0x00000238    28 0x00000238   28 -r-- .interp
02 0x00000254    32 0x00000254   32 -r-- .note.ABI_tag
03 0x00000278   176 0x00000278  176 -r-- .gnu.hash
04 0x00000328 3000 0x00000328 3000 -r-- .dynsym
05 0x00000ee0 1412 0x00000ee0 1412 -r-- .dynstr
06 0x00001464 250 0x00001464 250 -r-- .gnu.version
07 0x00001560 112 0x00001560 112 -r-- .gnu.version_r
08 0x000015d0 4944 0x000015d0 4944 -r-- .rela.dyn
09 0x00002920 2448 0x00002920 2448 -r-- .rela.plt
10 0x000032b0 23 0x000032b0 23 -r-x .init
11 0x000032d0 1648 0x000032d0 1648 -r-x .plt
12 0x00003940 24 0x00003940 24 -r-x .plt.got
13 0x00003960 73931 0x00003960 73931 -r-x .text
14 0x00015a2c 9 0x00015a2c 9 -r-x .fini
15 0x00015a40 20201 0x00015a40 20201 -r-- .rodata
16 0x0001a92c 2164 0x0001a92c 2164 -r-- .eh_frame_hdr
17 0x0001b1a0 11384 0x0001b1a0 11384 -r-- .eh_frame
18 0x0001e390 8 0x0021e390 8 -rw- .init_array
19 0x0001e398 8 0x0021e398 8 -rw- .fini_array
20 0x0001e3a0 2616 0x0021e3a0 2616 -rw- .data.rel.ro
21 0x0001edd8 480 0x0021edd8 480 -rw- .dynamic
22 0x0001efb8 56 0x0021efb8 56 -rw- .got
23 0x0001f000 840 0x0021f000 840 -rw- .got.plt
24 0x0001f360 616 0x0021f360 616 -rw- .data
25 0x0001f5c8 0 0x0021f5e0 4824 -rw- .bss
26 0x0001f5c8 232 0x00000000 232 ---- .shstrtab
```

С опцией **-Sr** rabin2 пометит начало/конец каждого раздела и передаст остальную информацию как комментарий.

```
$ rabin2 -Sr /bin/ls | head
fs sections
"f section. 1 0x00000000"
"f section..interp 1 0x000002a8"
"f section..note.gnu.build_id 1 0x000002c4"
"f section..note.ABI_tag 1 0x000002e8"
```

```
"f section..gnu.hash 1 0x00000308"
"f section..dynsym 1 0x000003b8"
"f section..dynstr 1 0x00000fb8"
"f section..gnu.version 1 0x00001574"
"f section..gnu.version_r 1 0x00001678"
```

Программа Radiff2

Radiff2 - это инструмент, предназначенный для сравнения двоичных файлов, подобно тому, как обычный `diff` сравнивает текстовые файлы.

```
$ radiff2 -h
Usage: radiff2 [-abBcCdjrspOxuUvV] [-A[A]] [-g sym] [-m graph_mode][-t %] [file] [file]
-a [arch]   указать архитектурный плагин для использования (x86, arm, ..)
-A [-A]     запустить ааа или аааа после загрузки каждого бинарика (смотри -C)
-b [bits]   указать размер регистров для архитектуры (16, 32, 64, ..)
-B         вывод в двоичном diff (GDIFF)
-c         количество изменений
-C         выполнить сравнения графов управления (колонки: смещение-А, оценка сходства, смещение-
В) (смотри -A)
-d         использовать delta diffing
-D         выводить дизассемблирование вместо шеснадцатеричного кода
-e [k=v]    установить значение переменной среды для всех экземплярах RCore
-g [sym|off1,off2]  сравнение графа заданного символа или двух смещений
-G [cmd]    запустить команду r2 на каждом созданном экземпляре RCore
-i         импортировать diff-ы целевых файлов (смотри -u, -U и -z)
-j         выводить результат в формте json
-n         печатать только адреса (diff.bare=1)
-m [aditsj]  выбрать режим вывода графа
-o         производить сравнение между байтами оп-кодов
-p         использовать физическую адресацию (io.va=0)
-q         тихий режим (обесцетить вывод и его объем)
-r         вывести результат в виде команд radare
-s         вычислить расстояние редактирование (без подстановки, алгоритм сравнения Eugene W. Myer O(ND))
-ss        вычислить расстояние редактирование Левенштейна (замена разрешена, O(N^2))
-S [name]   сортировать код diff (имя, длина имени, адрес, размер, тип, расстояние) (только для -C или -
g)
-t [0-100]  установить порог для кода diff (по умолчанию 70%)
-x         показать две колонки различий в шеснадцатеричном виде
-X         показать две колонки различий hexII
-u         унифицированный формат вывода (---++)
-U         унифицированный вывод, используя системный 'diff'
-v         показать версию
-V         больше информации (сейчас только для -s)
-z         сравнение экстрагированных строк
-Z         сравнение кода в виде сигнатур

Форматы вывода графа управления: (-m [mode])
<blank/a> Ascii-art
s      команды r2
d      Graphviz dot
g      Graph Modelling Language (gml)
j      json
J      json c disarm
k      SDB key-value
t      Tiny ascii art
i      Интерактивный ascii-art
```

Бинарное сравнение

Этот раздел основан на <http://radare.today> статье ``двоичное сравнение''

Без параметров `radiff2` по умолчанию показывает, какие байты изменены и соответствующие им смещения:

```
$ radiff2 genuine cracked      # оригиналный взломанный
0x000081e0 85c00f94c0 => 9090909090 0x000081e0
0x0007c805 85c00f84c0 => 9090909090 0x0007c805

$ rasm2 -d 85c00f94c0
test eax, eax
sete al
```

Заметим, два перехода (`jumps`) заменены на пор-ы.

Для массовой обработки может потребоваться более высокоуровневый обзор различий. Вот почему `radare2` может вычислять расстояние и процент сходства между двумя файлами с помощью опции `-s`:

```
$ radiff2 -s /bin/true /bin/false
similarity: 0.97
distance: 743
```

Если нужны более конкретные данные, можно подсчитать различия с помощью параметра **-C**:

```
$ radiff2 -c genuine cracked
2
```

Если нет уверенности, похожи ли двоичные файлы, с флагом **-C** можно проверить, есть ли соответствующие функции. В этом режиме, он даст вам три столбца для всех функций: «Смещение первого файла», «Процент совпадения» и «Смещение второго файла».

```
$ radiff2 -C /bin/false /bin/true
entry0 0x4013e8 | MATCH (0.904762) | 0x4013e2 entry0
sym.imp._libc_start_main 0x401190 | MATCH (1.000000) | 0x401190 sym.imp._libc_start_main
fcn.00401196 0x401196 | MATCH (1.000000) | 0x401196 fcn.00401196
fcn.0040103c 0x40103c | MATCH (1.000000) | 0x40103c fcn.0040103c
fcn.00401046 0x401046 | MATCH (1.000000) | 0x401046 fcn.00401046
fcn.000045e0 24 0x45e0 | UNMATCH (0.916667) | 0x45f0 24 fcn.000045f0
...
...
```

Можно попросить radiff2 сначала выполнить анализ - флаг **-A** запустит **aaa** на двоичных файлах. И мы можем указать архитектуру двоичных файлов для этого анализа, используя

```
$ radiff2 -AC -a x86 /bin/true /bin/false | grep UNMATCH
[x] Анализировать все флаги, начиная с sym. и entry0 (aa)
[x] Анализировать len байт инструкций на ссылки (aar)
[x] Анализировать вызовы функций (aac)
[ ] [*] Использование -AA или aaa запускают дополнительные экспериментальные алгоритмы анализа.
[x] Создание имен функций для fcn.* и sym.func.* (aan)
[x] Анализировать все флаги, начиная с sym. и entry0 (aa)
[x] Анализировать len байт инструкций на ссылки (aar)
[x] Анализировать вызовы функций (aac)
[ ] [*] Использование -AA или aaa запускают дополнительные экспериментальные алгоритмы анализа.
[x] Создание имен функций для fcn.* и sym.func.* (aan)
    sub.fileno_500 86 0x4500 | UNMATCH (0.965116) | 0x4510 86 sub.fileno_510
    sub._freading_4c0 59 0x44c0 | UNMATCH (0.949153) | 0x44d0 59 sub._freading_4d0
    sub.fileno_440 120 0x4440 | UNMATCH (0.200000) | 0x4450 120 sub.fileno_450
    sub.setlocale_fa0 64 0x3fa0 | UNMATCH (0.104651) | 0x3fb0 64 sub.setlocale_fb0
    fcn.00003a50 120 0x3a50 | UNMATCH (0.125000) | 0x3a60 120 fcn.00003a60
```

Теперь классная функция - radare2 поддерживает сравнения графов, как в DarunGrim, флаг **-g**. Можно присвоить имя символа, указав два смещения, если функция, которую вы хотите сравнить, называется по-разному в сравниваемых файлах. Например, **radiff2 -md -g main /bin/true /bin/false | xdot** - покажет различия в функции **main()** программ Unix **true** и **false**. Можно сравнить его с **radiff2 -md -g main /bin/false /bin/true | xdot** - (обратите внимание на порядок аргументов), чтобы получить две версии. Вот, что из этого получилось:

Части желтого цвета указывают, что некоторые смещения не совпадают. Серый кусок означает идеальное соответствие. Оранжевый подчеркивает сильную разницу. Если присмотреться, то увидим, что в левой части картинки есть **mov eax, 0x1; pop rbx; pop rbp; ret**, а в правой части - **xor edx, edx; pop rbx; pop rbp; ret**.

Двоичное сравнение является важной функцией для обратного проектирования. Его можно использовать для анализа обновлений безопасности, зараженных двоичных файлов, изменений прошивки и многое другое...

Мы показали только функции анализа кода, но radare2 поддерживает дополнительные типы сравнения между двумя двоичными файлами: на уровне байтов, дельтифицированные сходства и многое другое в будущем.

Есть планы по внедрению большего количества видов алгоритмов в r2: добавить поддержку дифференциации ASCII-аптраффов и лучшую интеграцию с остальной частью инструментария.

Утилита Rasm2

Программа **rasm2** является ассемблером/дизассемблером. Изначально инструмент **rasm** разработан для внесения исправлений (patching) в двоичный файл. Основная функция заключается в получении кодов байтов, соответствующих заданной машинной инструкции (оп-кода).

```
$ rasm2 -h
Usage: rasm2 [-ACdDeLBvw] [-a arch] [-b bits] [-o addr] [-s syntax]
               [-f file] [-F fil:ter] [-i skip] [-l len] 'code'|hex|-
-a [arch]      Задать архитектуру для процедур ассемблирования/дисассемблирования (смотрите -L)
-A             Показать результаты анализа заданных шеснадцатеричных кодов
-b [bits]       Задать разрядность регистров процессора (8, 16, 32, 64) (RASM2_BITS)
-B             Вывод/ввод в двоичном виде (-l обязателен для двоичного вывода)
-c [cpu]        Задать конкретный CPU (зависит от архитектуры)
-C             Вывод в формате C
```



Рис. 24: /bin/true против /bin/false

-d, -D Дизассемблировать из шестнадцатеричных кодов байтов (-D показывать оп-коды)
-e Задать прямой порядок (big endian) вместо обратного (little endian) порядок байтов
-E Выводить ESIL-выражения (вход тот же как в -d)
-f [file] Прочитать данные из файла
-F [in:out] Задать входной и/или выходной фильтр (att2intel, x86.pseudo, ...)
-h, -hh Показать данное сообщение, -hh дает больше информации
-i [len]忽орировать/пропустить len байт входных данных
-j Выводить в формате json
-k [kernel] Задать операционную систему (linux, windows, darwin, ...)
-l [len] Длина ввода/вывода
-L Перечислить плагины Asm: (a-ассемблирование, d-дизассемблирование, A-анализ, e-ESIL)
-o [offset] Задать начальное смещение (по умолчанию 0)
-O [file] Имя выходного файла (rasm2 -Bf a.asm -O a)
-p Выполнить SPP над входными данными для ассемблирования
-q "Тихий" режим
-r Представить результат в виде команд radare
-s [syntax] Задать синтаксис (intel, att)
-v Показать информацию о версии
-w Описание семантики оп-кода (что делает инструкция)
Если значение '-l' больше длины вывода, вывод заполняется пор-ами
Если последний аргумент '-' читается из stdin
Переменные среды:
RASM2_NOPLUGINS Не загружать динамические плагины (ускорение загрузки)
RASM2_ARCH То же, что и rasm2 -a
RASM2_BITS То же, что и rasm2 -b
R_DEBUG Отображать ли сообщения об ошибках и сигнал о сбое

Плагины для поддерживаемых архитектур перечисляются, используя флаг `-L`. Используемый плагин указывается именем в флаге `-a`

```
$ rasm2 -L
_dAe 8 16      6502      LGPL3   6502/NES/C64/Tamagotchi/T-1000 CPU
_dAe 8          8051      PD      8051 Intel CPU
_dA_ 16 32     arc       GPL3   Argonaut RISC Core
a___ 16 32 64 arm.as    LGPL3   as ARM Assembler (use ARM_AS environment)
adAe 16 32 64 arm       BSD    Capstone ARM disassembler
_dA_ 16 32 64 arm.gnu   GPL3   Acorn RISC Machine CPU
_d_ 16 32     arm.winedbg LGPL2  WineDBG's ARM disassembler
adAe 8 16      avr       GPL   AVR Atmel
adAe 16 32 64 bf        LGPL3  Brainfuck (by pancake, nibble) v4.0.0
_dA_ 32      chip8     LGPL3  Chip8 disassembler
_dA_ 16      cr16      LGPL3  cr16 disassembly plugin
_dA_ 32      cris       GPL3  Axis Communications 32-bit embedded processor
adA_ 32 64     dalvik    LGPL3  AndroidVM Dalvik
ad_ 16      dcpu16    PD    Mojang's DCPU-16
_dA_ 32 64     ebc       LGPL3  EFI Bytecode
adAe 16      gb        LGPL3  GameBoy(TM) (z80-like)
_dAe 16      h8300     LGPL3  H8/300 disassembly plugin
_dAe 32      hexagon   LGPL3  Qualcomm Hexagon (QDSP6) V6
_d_ 32      hppa      GPL3   HP PA-RISC
_dAe 16 32     i4004     LGPL3  Intel 4004 microprocessor
_dA_ 8       i8080     BSD   Intel 8080 CPU
adA_ 32      java      Apache Java bytecode
_d_ 32      lanai     GPL3  LANAI
_d_ 8       lh5801     LGPL3  SHARP LH5801 disassembler
_d_ 32      lm32      BSD   disassembly plugin for Lattice Micro 32 ISA
_dA_ 16 32     m68k      BSD   Capstone M68K disassembler
_dA_ 32      malbolge  LGPL3  Malbolge Ternary VM
_d_ 16      mcs96     LGPL3  condrets car
adAe 16 32 64 mips      BSD   Capstone MIPS disassembler
adAe 32 64     mips.gnu  GPL3  MIPS CPU
_dA_ 16      msp430    LGPL3  msp430 disassembly plugin
_dA_ 32      nios2     GPL3  NIOS II Embedded Processor
_dAe 8       pic       LGPL3  PIC disassembler
_dAe 32 64     ppc       BSD   Capstone PowerPC disassembler
_dA_ 32 64     ppc.gnu   GPL3  PowerPC
_d_ 32      propeller  LGPL3  propeller disassembly plugin
_dA_ 32 64     riscv     GPL   RISC-V
_dAe 32      rsp       LGPL3  Reality Signal Processor
_dAe 32      sh        GPL3  SuperH-4 CPU
_dA_ 8 16      snes      LGPL3  SuperNES CPU
_dAe 32 64     sparc     BSD   Capstone SPARC disassembler
_dA_ 32 64     sparc.gnu  GPL3  Scalable Processor Architecture
_d_ 16      spc700    LGPL3  spc700, snes' sound-chip
_d_ 32      sysz      BSD   SystemZ CPU disassembler
_dA_ 32      tms320    LGPLv3 TMS320 DSP family (c54x,c55x,c55x+,c64x)
_d_ 32      tricore   GPL3  Siemens TriCore CPU
_dAe 32      v810      LGPL3  v810 disassembly plugin
_dAe 32      v850      LGPL3  v850 disassembly plugin
_dAe 8 32      vax       GPI   VAX
```

adA_	32	wasm	MIT	WebAssembly (by cgvwzq) v0.1.0
dA	32	ws	LGPL3	Whitespace esoteric VM
a___	16 32 64	x86.as	LGPL3	Intel X86 GNU Assembler
_dAe	16 32 64	x86	BSD	Capstone X86 disassembler
a___	16 32 64	x86.nasm	LGPL3	X86 nasm assembler
a___	16 32 64	x86.nz	LGPL3	x86 handmade assembler
dA	16	xap	PD	XAP4 RISC (CSR)
dA	32	xcore	BSD	Capstone XCore disassembler
_dAe	32	xtensa	GPL3	XTensa CPU
adA_	8	z80	GPL	Zilog Z80

Обратите внимание, что «ad» в первой колонке означает, что и ассемблер и дизассемблер поддерживаются pluginом. ``d'' - доступен только дизассемблер, "a" - доступен только ассемблер.

Ассемблирование

Ассемблирование преобразует инструкцию микропроцессора в человекочитаемой форме в виде мнемоники в набор байтов, выполняемые машиной. В radare2 ассемблирование и дизассемблирование реализованы в API `r_asm_*`. Функции API запускаются командами `ra` и `rad` из командной строки radare2, а также с помощью инструмента `rasm2`.

Rasm2 используется для создания и вставки в блоки байтов шестнадцатеричных кодов, представляющих машинную инструкцию. Следующая строка собирает инструкцию `mov` для x86/32.

```
$ rasm2 -a x86 -b 32 'mov eax, 33'
b821000000
```

Помимо указания входных данных в качестве аргумента `rasm2`, мнемоники можно задавать в `rasm2`:

```
$ echo 'push eax;nop;nop' | rasm2 -f -
5090
```

Как видно, `rasm2` может собирать одну или несколько инструкций. В командной строке мнемоники разделяются точкой с запятой ; . Мнемоники также можно загружать это из файла, основывающийся на синтаксисе и директивах `nasm/gas/...` Можно ознакомиться со справочной страницей `rasm2` для получения более подробной информации.

Команды `ra` и `rad` являются подкомандами печати, они отображают результаты ассемблирования и дизассемблирования. Надо сформировать и записать в блок данных инструкцию? - используйте команды `Wa` и `WX` с добавлением строки с мнемониками или байтов.

Ассемблер понимает различные диалекты ассемблера: `x86` (варианты Intel и AT&T), `olly` (синтаксис OllyDBG), `powerpc` (PowerPC), `arm` и `java`. Для синтаксиса Intel `rasm2` имитирует NASM и GAS.

В репозитории исходного кода `rasm2` содержат несколько примеров реализации функций ассемблирования. Ознакомьтесь с ними, посмотрите, как реализуется ассемблирование в двоичный блок данных и в файл.

Создадим двоичный файл с именем `selfstop.rasm` из мнемоник ассемблера:

```
;
; Шелл-код запуска системного выхова, останавливающего программу.
; Код представлен в диалекте rasm для x86.
;
; -- Автор - pancake
;

.arch x86
.equ base 0x8048000
.org 0x8048000 ; смещение, куда мы вставим 5 байт jmp-a

selfstop:
    push 0x8048000
    pusha
    mov eax, 20
    int 0x80

    mov ebx, eax
    mov ecx, 19
    mov eax, 37
    int 0x80
    popa
    ret
;
```

```
; Вставка системного вызова
```

```
;
```

ret

Теперь можно собрать его ``на месте'' (in place):

```
[0x00000000]> e asm.bits = 32
[0x00000000]> wx `!rasm2 -f a.rasm` 
[0x00000000]> pd 20
 0x00000000      6800800408  push 0x8048000 ; 0x08048000
 0x00000005      60          pushad
 0x00000006      b814000000  mov eax, 0x14 ; 0x00000014
 0x0000000b      cd80       int 0x80
    syscall[0x80][0]=?
 0x0000000d      89c3       mov ebx, eax
 0x0000000f      b913000000  mov ecx, 0x13 ; 0x00000013
 0x00000014      b825000000  mov eax, 0x25 ; 0x00000025
 0x00000019      cd80       int 0x80
    syscall[0x80][0]=?
 0x0000001b      61          popad
 0x0000001c      c3          ret
 0x0000001d      c3          ret
```

Визуальный режим

Ассемблирование также доступно в визуальном режиме radare2 по нажатию клавиши A - вставка генерированного кода по адресу смещения. Функция ассемблирования в визуальном режиме не вносит изменения в память, пока не будет нажат enter - это замечательное ее свойство! Таким образом, можно проверить размер кода и то, какие инструкции перекрываются, прежде чем фиксировать изменения.

Дизассемблер

Дизассемблирование --- это обратное действие ассемблированию. Rasm2 принимает шестнадцатеричный код в качестве входных данных (но также может принимать файл в двоичной форме) и показывает человекочитаемую форму инструкций в виде мнемоник.

Для этого в программе rasm2 используется флаг -d:

```
$ rasm2 -a x86 -b 32 -d '90'
nop
```

В Rasm2 есть флаг -D, добавляющий к результату дизассемблирования -d смещения и байты. Флагами управляется также выполнение дизассемблирования блоков данных из заданного места в памяти. Флаги позволяют генерировать варианты дизассемблирования для последующего синтаксического анализа внешними скриптами, фильтроватьгером требуемой информации:

pd N

Дизассемблировать N инструкций

pD N

Дизассемблировать N байт

pda

Дизассемблировать все инструкции (смещающиеся на 1 байт или минимальный размер инструкции выравнивания), что полезно для ROP

pi, pI

То же самое, что pd и pD, но с использованием более простого вывода.

Конфигурирование дизассемблера

Функционирование ассемблера и дизассемблера настраиваются множеством переменных настройки. Конфигурация доступна с помощью команды e. Вот наиболее распространенные из них:

- asm.bytes - показать/скрыть байты
- asm.offset - показать/скрыть смещение

- `asm.lines` - показать/скрыть номера строк
- `asm.ucase` - показать дизассемблированный код в верхнем регистре
- ...

Используйте `e??asm.` для получения более подробной информации.

Программа ragg2

Название `ragg2` расшифровывается как `radare2 egg`, это основной блок для построения перемещаемых фрагментов кода, используемых для внедрения в целевые процессы при эксплуатации.

Программа `ragg2` компилирует программы на простом языке высокого уровня в маленькие двоичные файлы для архитектур x86, x86-64 и ARM.

По умолчанию он будет компилировать свой собственный язык `ragg2`, но также можно скомпилировать и C-код, используя GCC или Clang в шелл-коде в зависимости от расширения файла. Создадим C-файл под названием `a.c`:

```
int main() {
    write(1, "Hello World\n", 13);
    return 0;
}

$ ragg2 -a x86 -b32 a.c
e900000000488d3516000000bf01000000b80400000248c7c20d0000000f0531c0c348656c6c6f20576f726c640a00

$ rasm2 -a x86 -b 32 -D e900000000488d3516000000bf01000000b80400000248c7c20d0000000f0531c0c348656c6c6f20576f726c640a00
0x00000000      5          e900000000  jmp 5
0x00000005      1          48 dec eax
0x00000006      6          8d3516000000 lea esi, [0x16]
0x0000000c      5          bf01000000 mov edi, 1
0x00000011      5          b804000002 mov eax, 0x2000004
0x00000016      1          48 dec eax
0x00000017      6          c7c20d000000 mov edx, 0xd
0x0000001d      2          0f05 syscall
0x0000001f      2          31c0 xor eax, eax
0x00000021      1          c3 ret
0x00000022      1          48 dec eax
0x00000023      2          656c insb byte es:[edi], dx
0x00000025      1          6c insb byte es:[edi], dx
0x00000026      1          6f outsd dx, dword [esi]
0x00000027      3          20576f and byte [edi + 0x6f], dl
0x0000002a      2          726c jb 0x98
0x0000002c      3          640a00 or al, byte fs:[eax]
```

Пример компиляции ragg2

```
$ cat hello.r
exit@syscall(1);

main@global() {
    exit(2);
}

$ ragg2 -a x86 -b 64 hello.r
48c7c00200000050488b3c2448c7c0010000000f054883c408c3
0x00000000      1          48 dec eax
0x00000001      6          c7c002000000 mov eax, 2
0x00000007      1          50 push eax
0x00000008      1          48 dec eax
0x00000009      3          8b3c24 mov edi, dword [esp]
0x0000000c      1          48 dec eax
0x0000000d      6          c7c001000000 mov eax, 1
0x00000013      2          0f05 syscall
0x00000015      1          48 dec eax
0x00000016      3          83c408 add esp, 8
0x00000019      1          c3 ret

$ rasm2 -a x86 -b 64 -D 48c7c00200000050488b3c2448c7c0010000000f054883c408c3
0x00000000      7          48c7c002000000 mov rax, 2
0x00000007      1          50 push rax
0x00000008      4          488b3c24 mov rdi, qword [rsp]
0x0000000c      7          48c7c001000000 mov rax, 1
0x00000013      2          0f05 syscall
0x00000015      4          4883c408 add rsp, 8
0x00000019      1          c3 ret
```

Маленькие двоичные файлы

Файлы создаются, используя флаг `-F` в `ragg2` и `-C` в `rabin2`.

Синтаксис языка

Код `r_egg` компилируется однопроходным компилятором. Необходимо, в частности, определить правильный размер фрейма стека функции во избежание ошибок компиляции.

Компилятор генерирует ассемблерный код для x86-{32,64} и ARM. Также преследуется цель поддержать большее количество платформ. Этот код компилируется с помощью `r_asm` и формирует маленький двоичный файл с `r_bin`.

Можно использовать `r_egg` для создания автономных двоичных файлов, не зависящих от смещения, оп-кодов для инъекций в запущенные процессы и в двоичные файлы на диске.

Сгенерированный код не оптимизирован, но безопасен для выполнения.

Препроцессор

Псевдонимы

Иногда во время компиляции нужно заменить одну сущность в нескольких местах. Псевдонимы переводятся в операторы `'equ'` в сборке языка, т.е. эти переопределения делаются при помощи ключевых слов на уровне ассемблера.

```
AF_INET@alias(2);  
printf@alias(0x8053940);
```

Включения в текст

Используйте `cat(1)` или препроцессор для объединения нескольких скомпилированных файлов.

```
INCDIR@alias("/usr/include/ragg2");  
sys-osx.r@include(INCDIR);
```

Хешбэнг

Можно использовать хэшбэнг, чтобы сделать сгенерированные файлы (`eggs`) исполняемыми.

```
$ head -n1 hello.r  
#!/usr/bin/ragg2 -X  
$ ./hello.r  
Hello World!
```

Main

Кода выполняется последовательно: первая определенная функция будет исполняться первой. Если надо запустить `main()` просто пишите исходный код в следующей форме:

```
#!/usr/bin/ragg2 -X  
main();  
...  
main@global(128,64) {  
...}
```

Определение функции

Если надо разделить свой код на несколько блоков. Блоки кода привязаны к метке, за которой следуют скобки `...`.

Сигнатуры функций

```
name@type(stackframesize,staticframesize) { body },  
name : имя определяемой функции,  
type : см. типы функций ниже,  
stackframesize : получение пространства из стека для хранения локальных переменных,  
staticframesize : получение пространства из стека для хранения статических переменных (strings),  
body : код функции.
```

Типы функций

```
alias Используется для создания псевдонимов,  
data ; тело блока определяется в .data,  
inline ; тело функции встраивается в вызывающий код (inlined) при вызове,  
global ; сделать символ глобальным,  
fastcall ; функция, вызываемая с помощью соглашения о быстром вызове,  
syscall ; определение соглашения о вызове для системного вызова.
```

Системные вызовы

Программа r_egg предлагает синтаксический сахар для определения системных вызовов. Синтаксис выглядит следующим образом:

```
exit@syscall(1);  
@syscall() {  
` : mov eax,.arg````  
: int 0x80  
}  
main@global() {  
exit (0);  
}
```

Библиотеки

На данный момент отсутствует поддержка связывания r_egg программ с системными библиотеками. Но если вы вставляете код в программу (диск/память), нужно определить адрес каждой функции, используя синтаксис @alias.

Основная библиотека

Ведется разработка библиотеки, похожую на библиотеку libc, полностью реализованную на r_egg.

Переменные

```
.arg  
.arg0  
.arg1  
.arg2  
.var0  
.var2  
.fix  
.ret ; eax for x86, r0 for arm  
.bp
```

.pc

.sp

Внимание: Все числа после .var и .arg означают смещение относительно верхней части стека, они не являются частями имен переменных.

Массивы

Поддерживается в виде raw-указателей. TODO: улучшить эту функцию.

Отслеживание

Иногда r_egg-программы ломаются или просто не работают так, как ожидалось. Используйте архитектуру 'trace', чтобы получить трассировку вызова arch-backend:

```
$ ragg2 -a trace -s yourprogram.r
```

Указатели

TODO: Теоретически '*' используется для получения содержимого указателя памяти.

Виртуальные регистры

TODO: a0, a1, a2, a3, sp, fp, bp, pc

Математические операции

Ragg2 поддерживает присвоение локальным переменным результатов математических операций, в том числе следующие операторы:

+ - * / & | ^

Возвращаемые значения

Возвращаемое значение хранится в регистре a0, этот регистр устанавливается в результате вызова функции, или при использовании переменной без присвоения.

```
$ cat test.r
add@global(4) {
    .var0 = .arg0 + .arg1;
    .var0;
}

main@global() {
    add (3,4);
}

$ ragg2 -F -o test test.r
$ ./test
$ echo $?
7
```

Ловушки (traps)

Каждая архитектура имеет отдельную инструкцию для прерывания выполнения программы. Язык REgg использует функцию 'break()' для запуска emit_trap - обратный вызов для выбранной архитектуры. Функция break(); --> компилируется в 'int3' на x86, break; --> компилируется в 'int3' на x86.

Встроенный ассемблер

Строки с префиксом ':' просто встроены в выходной файл.

```
: jmp 0x8048400
: .byte 33,44
```

Метки

Метки определяются, используя ключевое слово `:`, следующим образом:

```
:label_name:,  
/* бесконечный цикл */,  
goto(label_name).
```

Управление исполнением

```
goto (addr) -- ветвление,  
while (cond),  
if (cond),  
if (cond) { body } else { body },  
break () -- выполняет инструкцию trap.
```

Комментарии

Поддерживаемый синтаксис для комментариев:

```
/* многострочный комментарий */,  
// односторонний комментарий,  
# односторонний комментарий.
```

Программа rahash2

Инструмент rahash2 можно использовать для вычисления контрольных сумм файлов, блоков данных дисковых устройств и строк. Реализовано множество различных хэш-алгоритмов. Также можно выполнять некоторые операции кодирования / декодирования, такие как шифрование base64 и хор (гаммирование).

Вот пример использования:

```
$ rahash2 -a md5 -s "hello world"
```

Обратите внимание, что rahash2 также позволяет считывать из `stdin` в потоке, поэтому не нужно выделять 4 ГБ оперативной памяти для вычисления хэша файла размером 4 ГБ.

Блочное хеширование

При проведении сравнения больших объемов файлов полезно вычислять частичные контрольные суммы. При этом файл разделяется на небольшие части, которые легче идентифицировать по содержимому. Аналогично содержимое раздела диска разбивается на блоки, затем хешируются, и по значениям хэшей далее можно проводить сравнение. Если обнаружено два блока с один и тем же хэшем, то весьма вероятно содержание этих блоков будет одно и то же. Очень часто таким образом идентифицируются блоки с нулевыми значениями. Блочное хеширование удобно для определения того, какие блоки изменились. Также инструмент удобно при анализе дампов оперативной памяти с виртуальной машины.

Построение блочного хэша делается при помощи команды, аналогичной

```
$ rahash2 -B 1M -b -a sha256 /bin/ls
```

Хеширование при помощи rabin2

Инструмент rabin2 анализирует двоичные заголовки файлов, но он также может использовать плагины rhash для вычисления контрольной суммы разделов в двоичном файле.

```
$ rabin2 -K md5 -S /bin/ls
```

Получение хэшей в сеансе radare2

Вычисление контрольной суммы текущего блока при запуске radare2 - команда `ph`. Название алгоритма передается в качестве параметра. Пример сеанса:

```
$ radare2 /bin/ls  
[0x08049790]> bf entry0  
[0x08049790]> ph md5  
d2994c75adaa58392f953a448de5fba7
```

Можно использовать все алгоритмы хеширования, поддерживаемые `rahash2`:

```
[0x00000000]> ph?
md5
sha1
sha256
sha384
sha512
md4
xor
xorpair
parity
entropy
hamdist
pcprint
mod255
xxhash
adler32
luhn
crc8smbus
crc15can
crc16
crc16hdlc
crc16usb
crc16citt
crc24
crc32
crc32c
crc32ecma267
crc32bzip2
crc32d
crc32mpeg2
crc32posix
crc32q
crc32jamcrc
crc32xfer
crc64
crc64ecma
crc64we
crc64xz
crc64iso
```

Команда `ph` принимает необязательный числового аргумент для указания длины блока байтов, подлежащего хешированию, параметр изменит размер блока по умолчанию. Например:

```
[0x08049A80]> ph md5 32
9b9012b00ef7a94b5824105b7aaad83b
[0x08049A80]> ph md5 64
a71b087d8166c99869c9781e2edcf183
[0x08049A80]> ph md5 1024
a933cc94cd705f09a41ecc80c0041def
```

Примеры

Инструмент `rahash2` используется для вычисления контрольных сумм, хэшей, включает функции хеширование байтовых потоков, блоков, файлов, текстовых строк.

```
$ rahash2 -h
Usage: rahash2 [-rBhLkv] [-b S] [-a A] [-c H] [-E A] [-s S] [-f O] [-t O] [file] ...
-a algo      список алгоритмов, разделенных запятой (по умолчанию 'sha256')
-b bsize     задать размер блока данных (вместо файла целиком)
-B           показать хэши по блокам
-c hash      сравнить с хэшем, переданным в параметре
-e           переключить порядок байтов (swap endian) (использовать little endian)
-E algo      шифровать, используйте -S для установки ключа и -I для установки вектора инициализации
-D algo      расшифровать, используйте -S для установки ключа и -I для установки вектора инициализации
-f from     начать хеширование по заданному адресу
-i num       повторить хеширование num раз
-I iv        использовать заданный вектор инициализации (IV) (шестнадцатеричный или s:строковый)
-S seed      задать начальное зерно (seed) (шестнадцатеричный или s:строковый), используйте ^ для задания префикса (E), "-" загрузит ключ с stdin, префикс \ указывает на файл
-k           показать хэш, используя алгоритм randomkey из библиотеки openssh
-q           запуск в "тихом" режиме (-qq показывает только результаты)
-L           перечислить все доступные алгоритмы (смотрите -a)
-r           вывести в виде команд radare
-s string   хешировать строку вместо файлов
-t to        остановить хеширование на заданном адресе
-x hexstr   хешировать заданный блок байтов, заданных в шестнадцатеричном виде
-v           показать версию программы и дополнительную информацию
```

Чтобы получить хэш-значение MD5 текстовой строки, используйте параметр **-S** :

```
$ rahash2 -q -a md5 -s 'hello world'  
5eb63bbbe01eede093cb22bb8f5acdc3
```

Можно вычислить хэш-значения для содержимого файлов. Не пытайтесь проводить хэширование очень больших файлов в такой форме, так как rahash2 буферизует весь вход в памяти перед вычислением хэша. Чтобы надо применить все алгоритмы хэширования, известные rahash2, используйте **all** в качестве имени алгоритма:

```
$ rahash2 -a all /bin/ls  
/bin/ls: 0x00000000-0x000268c7 md5: 767f0fff116bc6584dbfc1af6fd48fc7  
/bin/ls: 0x00000000-0x000268c7 sha1: 404303f3960f196f42f8c2c12970ab0d49e28971  
/bin/ls: 0x00000000-0x000268c7 sha256: 74ea05150acf311484bdd19c608aa02e6bf3332a0f0805a4deb278e17396354  
/bin/ls: 0x00000000-0x000268c7 sha384: c6f811287514ceeeaab73b5b2f54545036d6fd3a192ea5d6a1fc494d46151df4117e1c62de0  
/bin/ls: 0x00000000-0x000268c7 sha512: 53e4950a150f06d7922a2ed732060e291bf0e1c2ac20bc72a41b9303e1f2837d50643761030d8  
/bin/ls: 0x00000000-0x000268c7 md4: fdfe7c7118a57c1ff8c88a51b16fc78c  
/bin/ls: 0x00000000-0x000268c7 xor: 42  
/bin/ls: 0x00000000-0x000268c7 xorpair: d391  
/bin/ls: 0x00000000-0x000268c7 parity: 00  
/bin/ls: 0x00000000-0x000268c7 entropy: 5.95471783  
/bin/ls: 0x00000000-0x000268c7 hamdist: 00  
/bin/ls: 0x00000000-0x000268c7 pcprint: 22  
/bin/ls: 0x00000000-0x000268c7 mod255: ef  
/bin/ls: 0x00000000-0x000268c7 xxhash: 76554666  
/bin/ls: 0x00000000-0x000268c7 adler32: 7704fe60  
/bin/ls: 0x00000000-0x000268c7 luhn: 01  
/bin/ls: 0x00000000-0x000268c7 crc8smbus: 8d  
/bin/ls: 0x00000000-0x000268c7 crc15can: 1cd5  
/bin/ls: 0x00000000-0x000268c7 crc16: d940  
/bin/ls: 0x00000000-0x000268c7 crc16hdlc: 7847  
/bin/ls: 0x00000000-0x000268c7 crc16usb: 17bb  
/bin/ls: 0x00000000-0x000268c7 crc16citt: 67f7  
/bin/ls: 0x00000000-0x000268c7 crc24: 3e7053  
/bin/ls: 0x00000000-0x000268c7 crc32: c713f78f  
/bin/ls: 0x00000000-0x000268c7 crc32c: 6cfba67c  
/bin/ls: 0x00000000-0x000268c7 crc32ecma267: b4c809d6  
/bin/ls: 0x00000000-0x000268c7 crc32bzip2: a1884a09  
/bin/ls: 0x00000000-0x000268c7 crc32d: d1a9533c  
/bin/ls: 0x00000000-0x000268c7 crc32mpeg2: 5e77b5f6  
/bin/ls: 0x00000000-0x000268c7 crc32posix: 6ba0dec3  
/bin/ls: 0x00000000-0x000268c7 crc32q: 3166085c  
/bin/ls: 0x00000000-0x000268c7 crc32jamcrc: 38ec0870  
/bin/ls: 0x00000000-0x000268c7 crc32xfer: 7504089d  
/bin/ls: 0x00000000-0x000268c7 crc64: b6471d3093d94241  
/bin/ls: 0x00000000-0x000268c7 crc64ecma: b6471d3093d94241  
/bin/ls: 0x00000000-0x000268c7 crc64we: 8fe37d44a47157bd  
/bin/ls: 0x00000000-0x000268c7 crc64xz: ea83e12c719e0d79  
/bin/ls: 0x00000000-0x000268c7 crc64iso: d243106d9853221c
```

Плагины

Radare2 реализован на основе библиотек, в том числе поддерживающих плагины для расширения их возможностей или добавления поддержки новых архитектур и форматов.

В этом разделе объясняется, что такое плагины, их реализация и использование.

Типы плагинов

```
$ ls libr/*/p | grep : | awk -F / '{ print $2 }'  
anal      # плагины анализа,  
asm       # плагины ассемблерования/дизассемблерования,  
bin       # плагины синтаксического анализа форматов,  
bp        # плагины точек остановов,  
core      # плагины ядра (новые команды),  
crypto    # шифрование/дешифрование/хэши/...,  
debug     # плагины отладки,  
egg       # кодировщики shell-кода, и т.п.,  
fs        # файловые системы и таблицыパーティций,  
io        # плагины ввода-вывода,  
lang      # встроенные языки сценариев,  
parse     # синтаксический анализ результатов дизассемблерования,  
reg       # регистрация логики архитектуры.
```

Получение списка плагинов

Инструменты r2 поддерживают флаг **-L** для вывода всех плагинов во видам реализованных функций.

```
rasm2 -L      # перечень плагинов asm,  
r2 -L        # перечень плагинов io,  
rabin2 -L    # перечень плагинов bin,  
rahash2 -L    # перечень плагинов хэш/шифрование/десифрования.
```

В проекте r2land есть другие плагины, можно получить списки прямо в r2 при помощи группы команд L.

Вот некоторые из команд:

```
L          # перечень плагинов ядра (core),  
iL         # перечень плагинов bin,  
dL         # перечень плагинов debug,  
mL         # перечень плагинов fs,  
ph         # вывести перечень поддерживаемых алгоритмов хеширования.
```

Использование ? в качестве значения в выражении выдает возможные варианты значений для переменных среды.

```
e asm.arch=?  # список плагинов ассемблирования/дизассемблирования  
e anal.arch=? # список плагинов анализа
```

Примечания

Обратите внимание - по ходу развития проекта выявляются некоторые противоречия, они исправляются в новых версиях radare2.

Плагины ввода-вывода

Весь доступ к файлам, сети, отладчику и вводу/выводу другого типа представлен специальным уровнем абстракции, позволяющим radare обрабатывать все данные как файл.

Плагины ввода-вывода представляют операции открытия, чтения, записи и 'system' в виде виртуальных файловых систем. Можно заставить radare обрабатывать что угодно как простой файл. Примерами выступают сокетные соединения, удаленный сеанс radare, файлы, процессы, устройства, сеансы gdb.

Таким образом, когда radare считывает блок байтов, задача плагина ввода-вывода заключается в том, чтобы получить эти байты из своего источника, поместить их во внутренний буфер. Плагин ввода-вывода выбирается по URI открываемого файла. Примеры:

- URI отладки

```
$ r2 dbg:///bin/ls<br />  
$ r2 pid://1927
```

- Удаленные сеансы

```
$ r2 rap://:1234<br />  
$ r2 rap://<host>:1234//bin/ls
```

- Виртуальные буферы

```
$ r2 malloc://512<br />  
сокращение для  
$ r2 -
```

Получение списка плагинов ввода-вывода в radare - radare2 -L:

```
$ r2 -L  
rw_ ar      открытие файлов ar/lib [ar|lib]://[file//path] (LGPL3)  
rw_ bfdbg   отладчик для экзотического BrainFuck (bfdbg://path/to/file) (LGPL3)  
rwd bochs   подключение к отладчику BOCHS (LGPL3)  
r_d debug   встроенный отладчик (dbg://bin/ls dbg://1388 pidof:// waitfor://) (LGPL3) v0.2.0 pancake  
rw_ default  открытие локальных файлов, используя def_mmap:// (LGPL3)  
rwd gdb     подключение к gdbserver, 'qemu -s', gdb://localhost:1234 (LGPL3)  
rw_ gprobe   открытие gprobe-соединения - gprobe:// (LGPL3)  
rw_ gzip    чтение/запись файлов gzip (LGPL3)  
rw_ http    запрос GET HTTP (http://rada.re/) (LGPL3)  
rw_ ihex    интерпретация файла Intel HEX (ihex://eproms.hex) (LGPL)  
r__ mach    отладка архитектуры mach io-плагин (в этой версии не поддерживается) (LGPL)  
rw_ malloc   выделение памяти (malloc://1024 hex://cd8090) (LGPL3)  
rw_ mmap    открытие файлов, используя mmap:// (LGPL3)  
rw_ null    плагин null-plugin (null://23) (LGPL3)  
rw_ procpid доступ в /proc/pid/mem, io-плагин (LGPL3)  
rwd ptrace  использование ptrace и /proc/pid/mem (если доступны) io-плагин (LGPL3)  
rwd qnx    подключение к инстанции QNX pdebug, qnx://host:1234 (LGPL3)  
rw_ r2k     доступ к API ядра, io-плагин (r2k://) (LGPL3)  
rw_ r2pipe  r2pipe io-плагин (MIT)  
rw_ r2web   r2web io-клиент (r2web://cloud.rada.re/cmd/) (LGPL3)  
rw_ rap     сетевой протокол radare (rap://:port rap://host:port/file) (LGPL3)
```

```

rw_ rbuf    RBuffer io-плагин: rbuf:// (LGPL)
rw_ self    чтение памяти из своего адресного пространства, 'self://' (LGPL3)
rw_ shm     ресурсы разделяемой памяти (shm://key) (LGPL3)
rw_ sparse   выделение разреженных буферов (sparse buffer) (sparse://1024 sparse://) (LGPL3)
rw_ tcp      загрузка файлов по TCP (listen или connect) (LGPL3)
rwd windbg  подключение к отладчику KD (windbg://socket) (LGPL3)
rwd winedbg Wine-dbg io- и debug.io-плагины для r2 (MIT)
rw_ zip      открытие файлов zip [apk|ipa|zip|zipall]://[file//path] (BSD)

```

Реализация плагина дизассемблирования

Архитектура Radare2 модульная, поэтому реализация поддержки новой архитектуры очень проста, особенно тем, кто свободно владеет C. По множеству причин реализацию еще проще сделать на основе исходников из дерева репозитория radare2.

Создаем всего один файл C, называемый `asm_mycpu.c` и соответствующий Makefile.

Ключевым моментом плагина RAasm является структура

```

RAsmPlugin r_asm_plugin_mycpu = {
    .name = "mycpu",
    .license = "LGPL3",
    .desc = "MYCPU disassembly plugin",
    .arch = "mycpu",
    .bits = 32,
    .endian = R_SYS_ENDIAN_LITTLE,
    .disassemble = &disassemble
};

```

Здесь `.disassemble` --- это указатель на функцию дизассемблирования, принимающую буфер байтов и его длину:

```
static int disassemble(RAasm *a, RAasmOp *op, const ut8 *buf, int len)
```

Makefile

```

NAME=asm_snes
R2_PLUGIN_PATH=$(shell r2 -H R2_USER_PLUGINS)
LIBEXT=$(shell r2 -H LIBEXT)
CFLAGS=-g -fPIC $(shell pkg-config --cflags r_anal)
LDFLAGS=-shared $(shell pkg-config --libs r_anal)
OBJS=$(NAME).o
LIB=$(NAME).$(LIBEXT)

all: $(LIB)

clean:
    rm -f $(LIB) $(OBJS)

$(LIB): $(OBJS)
    $(CC) $(CFLAGS) $(LDFLAGS) $(OBJS) -o $(LIB)

install:
    cp -f asm_mycpu.$(SO_EXT) $(R2_PLUGIN_PATH)

uninstall:
    rm -f $(R2_PLUGIN_PATH)/asm_mycpu.$(SO_EXT)

asm_mycpu.c
/* radare - LGPL - Copyright 2018 - user */

#include <stdio.h>
#include <string.h>
#include <r_types.h>
#include <r_lib.h>
#include <r_asm.h>

static int disassemble(RAasm *a, RAasmOp *op, const ut8 *buf, int len) {
    struct op_cmd cmd = {

```

```

        .instr = "",
        .operands = ""
    };
    if (len < 2) return -1;
    int ret = decode_opcode (buf, len, &cmd);
    if (ret > 0) {
        sprintf (op->buf_asm, R_ASM_BUFSIZE, "%s %s",
                cmd.instr, cmd.operands);
    }
    return op->size = ret;
}

RAsmPlugin r_asm_plugin_mycpu = {
    .name = "mycpu",
    .license = "LGPL3",
    .desc = "MYCPU disassembly plugin",
    .arch = "mycpu",
    .bits = 32,
    .endian = R_SYS_ENDIAN_LITTLE,
    .disassemble = &disassemble
};

#ifndef R2_PLUGIN_INCORE
R_API RLibStruct radare_plugin = {
    .type = R_LIB_TYPE_ASM,
    .data = &r_asm_plugin_mycpu,
    .version = R2_VERSION
};
#endif

```

После компиляции radare2 упомянет плагин в списке плагинов вывода:

```
_d__ _8_32      mycpu      LGPL3      MYCPU
```

Перемещение плагина в дерево исходников radare2

Добавление новой архитектуры в основную ветвь r2 требует изменения нескольких файлов, что обеспечивает плагину правильную интеграцию аналогично остальным плагинам.

Список файлов, требующих изменения:

- `plugins.def.cfg` : добавить название плагина в виде строки `asm.mycpu`,
- `libr/asm/p/mycpu.mk` : инструкции сборки,
- `libr/asm/p/asm_mycpu.c` : реализация,
- `libr/include/r_asm.h` : добавление определенных структур.

Сравните результат с плагином дизассемблирования для процессора NIOS II по следующим коммитам:

Implement RAsm plugin: <https://github.com/radareorg/radare2/commit/933dc0ef6ddfe44c88bbb261165bf8f8b531476b>

Реализация плагина RAnal: <https://github.com/radareorg/radare2/commit/ad430f0d52fbe933e0830c49ee607e9b0e4ac8f2>

Реализация нового плагина анализа

После реализации плагина дизассемблирования, его вывод бывает далеко не идеальным - нет правильной подсветки, нет ``опорных линий'' и т.д. Radare2 требует от каждого архитектурного плагина предоставлять также аналитическую информацию о каждом оп-коде. На данный момент реализация дизассемблирования и анализа оп-кодов разделена на два модуля - RAsm и RAnal.

Реализуем плагин анализа. Принцип схож с реализацией дизассемблирования - реализовать C-файл и соответствующий Makefile.

Структура плагина RAnal выглядит так -

```
RAnalPlugin r_anal_plugin_v810 = {
    .name = "mycpu",
    .desc = "MYCPU code analysis plugin",
    .license = "LGPL3",
    .arch = "mycpu",
```

```

.bits = 32,
.op = mycpu_op,
.esil = true,
.set_reg_profile = set_reg_profile,
};

```

Аналогично плагину дизассемблирования, есть ключевая функция - `mycpu_op`, сканирующая оп-код и создающая структуру RAnalOp. В этом примере плагины анализа также выполняют описание оп-кода командами ESIL (`uplifting`), так как включен `.esil = true`. Таким образом, `mycpu_op` обязан заполнять соответствующее ESIL-поле структуры RAnalOp для оп-кодов. Вторая важная вещь для кодирования оп-кодов в ESIL и выполнения эмуляции - регистровый профиль, подобно тому, как в отладчике, он задается внутри функции `set_reg_profile`.

Makefile

```

NAME=anal_snes
R2_PLUGIN_PATH=$(shell r2 -H R2_USER_PLUGINS)
LIBEXT=$(shell r2 -H LIBEXT)
CFLAGS=-g -fPIC $(shell pkg-config --cflags r_anal)
LDFLAGS=-shared $(shell pkg-config --libs r_anal)
OBJS=$(NAME).o
LIB=$(NAME).$(LIBEXT)

all: $(LIB)

clean:
    rm -f $(LIB) $(OBJS)

$(LIB): $(OBJS)
    $(CC) $(CFLAGS) $(LDFLAGS) $(OBJS) -o $(LIB)

install:
    cp -f anal_snes.$(SO_EXT) $(R2_PLUGIN_PATH)

uninstall:
    rm -f $(R2_PLUGIN_PATH)/anal_snes.$(SO_EXT)

anal_snes.c:
/* radare - LGPL - Copyright 2015 - condret */

#include <string.h>
#include <r_types.h>
#include <r_lib.h>
#include <r_asm.h>
#include <r_anal.h>
#include "snes_op_table.h"

static int snes_anop(RAnal *anal, RAnalOp *op, ut64 addr, const ut8 *data, int len) {
    memset (op, '\0', sizeof (RAnalOp));
    op->size = snes_op[data[0]].len;
    op->addr = addr;
    op->type = R_ANAL_OP_TYPE_UNK;
    switch (data[0]) {
        case 0xea:
            op->type = R_ANAL_OP_TYPE_NOP;
            break;
    }
    return op->size;
}

struct r_anal_plugin_t r_anal_plugin_snes = {
    .name = "snes",
    .desc = "SNES analysis plugin",
    .license = "LGPL3",
    .arch = R_SYS_ARCH_NONE,
}

```

```

.bits = 16,
.init = NULL,
.fini = NULL,
.op = &snes_anop,
.set_reg_profile = NULL,
.fingerprint_bb = NULL,
.fingerprint_fcn = NULL,
.diff_bb = NULL,
.diff_fcn = NULL,
.diff_eval = NULL
};

#ifndef R2_PLUGIN_INCORE
R_API RLibStruct radare_plugin = {
    .type = R_LIB_TYPE_ANAL,
    .data = &r_anal_plugin_snes,
    .version = R2_VERSION
};
#endif

```

После компиляции radare2 упомянет плагин в списке плагинов вывода:

dA _8_16 snes LGPL3 SuperNES CPU

snes_op_table.h: https://github.com/radareorg/radare2/blob/master/libr/asm/arch/snes/snes_op_table.h

Пример:

- 6502: <https://github.com/radareorg/radare2/commit/64636e9505f9ca8b408958d3c01ac8e3ce254a9b>
- SNES: <https://github.com/radareorg/radare2/commit/60d6e5a1b9d244c7085b22ae8985d00027624b49>

Реализация нового формата двоичного файла

Включение виртуальной адресации

В info надо добавить et->has_va = 1; и ptr->srwx с атрибутом R_BIN_SCN_MAP;

Оформление папки с именем формата файла в libr/bin/format

Makefile:

```

NAME=bin_nes
R2_PLUGIN_PATH=$(shell r2 -H R2_USER_PLUGINS)
LIBEXT=$(shell r2 -H LIBEXT)
CFLAGS=-g -fPIC $(shell pkg-config --cflags r_bin)
LDFLAGS=-shared $(shell pkg-config --libs r_bin)
OBJS=$(NAME).o
LIB=$(NAME).$(LIBEXT)

all: $(LIB)

clean:
    rm -f $(LIB) $(OBJS)

$(LIB): $(OBJS)
    $(CC) $(CFLAGS) $(LDFLAGS) $(OBJS) -o $(LIB)

install:
    cp -f $(NAME).$(SO_EXT) $(R2_PLUGIN_PATH)

uninstall:
    rm -f $(R2_PLUGIN_PATH)/$(NAME).$(SO_EXT)

bin_nes.c:
#include <r_util.h>
#include <r_bin.h>

```

```

static bool load_buffer(RBinFile *bf, void **bin_obj, RBuffer *b, ut64 loadaddr, Sdb *sdb) {
    ut64 size;
    const ut8 *buf = r_buf_data(b, &size);
    r_return_val_if_fail(buf, false);
    *bin_obj = r_bin_internal_nes_load(buf, size);
    return *bin_obj != NULL;
}

static void destroy(RBinFile *bf) {
    r_bin_free_all_nes_obj(bf->o->bin_obj);
    bf->o->bin_obj = NULL;
}

static bool check_buffer(RBuffer *b) {
    if (!buf || length < 4) return false;
    return (!memcmp(buf, "\x4E\x45\x53\x1A", 4));
}

static RBinInfo* info(RBinFile *arch) {
    RBinInfo \*ret = R_NEW0(RBinInfo);
    if (!ret) return NULL;

    if (!arch || !arch->buf) {
        free(ret);
        return NULL;
    }
    ret->file = strdup(arch->file);
    ret->type = strdup("ROM");
    ret->machine = strdup("Nintendo NES");
    ret->os = strdup("nes");
    ret->arch = strdup("6502");
    ret->bits = 8;

    return ret;
}

struct r_bin_plugin_t r_bin_plugin_nes = {
    .name = "nes",
    .desc = "NES",
    .license = "BSD",
    .get_sdb = NULL,
    .load_buffer = &load_buffer,
    .destroy = &destroy,
    .check_buffer = &check_buffer,
    .baddr = NULL,
    .entries = NULL,
    .sections = NULL,
    .info = &info,
};
#endif R2_PLUGIN_INCORE
R_API RLibStruct radare_plugin = {
    .type = R_LIB_TYPE_BIN,
    .data = &r_bin_plugin_nes,
    .version = R2_VERSION
};
#endif

```

Примеры

- XBE - <https://github.com/radareorg/radare2/pull/972>,
- COFF - <https://github.com/radareorg/radare2/pull/645>,
- TE - <https://github.com/radareorg/radare2/pull/61>,

- Zimgz - <https://github.com/radareorg/radare2/commit/d1351cf836df3e2e63043a6dc728e880316f00eb>,
- OMF - <https://github.com/radareorg/radare2/commit/44fd8b2555a0446ea759901a94c06f20566bbc40>.

Реализация плагина отладчика

- Добавление профиля регистров отладчика в shlr/gdb/src/core.c,
- Добавление поддержки профиля и архитектуры регистров в libr/debug/p/debug_native.c и libr/debug/p/debug_gdb.c,
- Добавьте код для профилей в функцию r_debug_gdb_attach(RDebug *dbg, int pid).

Если добавить поддержку gdb, тогда можно будет видеть профиль регистров в активном сеансе gdb, используя команду `maint print registers`.

То ли еще будет!

- Статья по теме: <http://radare.today/posts/extending-r2-with-new-plugins/>

Некоторые коммиты, связанные с «Реализацией новых архитектур»

- Extensa: <https://github.com/radareorg/radare2/commit/6f1655c49160fe9a287020537afe0fb8049085d7>,
- Malbolge: <https://github.com/radareorg/radare2/pull/579>,
- 6502: <https://github.com/radareorg/radare2/pull/656>,
- h8300: <https://github.com/radareorg/radare2/pull/664>,
- GBA: <https://github.com/radareorg/radare2/pull/702>,
- CR16: [&& 726](https://github.com/radareorg/radare2/pull/721),
- XCore: <https://github.com/radareorg/radare2/commit/bb16d1737ca5a471142f16ccfa7d444d2713a54d>,
- SharpLH5801: <https://github.com/neuschaefer/radare2/commit/f4993cca634161ce6f82a64596fce45fe6b818e7>,
- MSP430: <https://github.com/radareorg/radare2/pull/1426>,
- HP-PA-RISC: <https://github.com/radareorg/radare2/commit/f8384feb6ba019b91229adb8fd6e0314b0656f7b>,
- V810: <https://github.com/radareorg/radare2/pull/2899>,
- TMS320: <https://github.com/radareorg/radare2/pull/596>.

Реализация новой псевдоархитектуры

Простой плагин для Z80, который можно использовать в качестве примера:

<https://github.com/radareorg/radare2/commit/8ff6a92f65331cf8ad74cd0f44a60c258b137a06>

Плагины Python

Необходимым условием для реализации модулей расширения на языке Python для radare2 является установка плагина r2lang: `r2pm -i lang-python`. Обратим внимание, что ради читабельности кода в следующих примерах отсутствуют фактическая реализация функций!

Нужно сделать следующее: 1. `import r2lang` и `from r2lang import R` (модуль констант) 2. Далее для плагина RAsm реализуем функцию с двумя подфункциями - `assemble` и `disassemble`, она также должна возвращать структуру, представляющую плагин

```
def mycpu(a):
    def assemble(s):
        return [1, 2, 3, 4]

    def disassemble(memview, addr):
        try:
            opcode = get_opcode(memview) # https://docs.python.org/3/library/stdtypes.html#memview
            opstr = optbl[opcode][1]
            return [4, opstr]
        except:
            return [4, "unknown"]

    return {
        "name" : "mycpu",
        "arch" : "mycpu",
        "bits" : 32,
        "endian" : R.R_SYS_ENDIAN_LITTLE,
        "license" : "GPL",
    }
```

3. Данная конструкция должна содержать указатели на эти две функции -- `assemble` и `disassemble`

```

        "desc" : "MYCPU disasm",
        "assemble" : assemble,
        "disassemble" : disassemble,
    }

```

4. Для плагина RAnal создается функция с двумя подфункциями - `set_reg_profile` и `op`, возвращается структура

```

def mycpu_anal(a):
    def set_reg_profile():
        profile = "=PC pc\n" + \
                  "=SP sp\n" + \
                  "gpr r0 .32 0 0\n" + \
                  "gpr r1 .32 4 0\n" + \
                  "gpr r2 .32 8 0\n" + \
                  "gpr r3 .32 12 0\n" + \
                  "gpr r4 .32 16 0\n" + \
                  "gpr r5 .32 20 0\n" + \
                  "gpr sp .32 24 0\n" + \
                  "gpr pc .32 28 0\n"
        return profile

    def op(memview, pc):
        analop = {
            "type" : R.R_ANAL_OP_TYPE_NULL,
            "cycles" : 0,
            "stackop" : 0,
            "stackptr" : 0,
            "ptr" : -1,
            "jump" : -1,
            "addr" : 0,
            "eob" : False,
            "esil" : "",
        }
        try:
            opcode = get_opcode(memview) # https://docs.python.org/3/library/stdtypes.html#memview
            esilstr = optbl[opcode][2]
            if optbl[opcode][0] == "J": # it's jump
                analop["type"] = R.R_ANAL_OP_TYPE_JMP
                analop["jump"] = decode_jump(opcode, j_mask)
                esilstr = jump_esil(esilstr, opcode, j_mask)

        except:
            result = analop
        # Don't forget to return proper instruction size!
        return [4, result]

```

5. Эта структура должна содержать указатели на две функции `set_reg_profile` и `op`:

```

return {
    "name" : "mycpu",
    "arch" : "mycpu",
    "bits" : 32,
    "license" : "GPL",
    "desc" : "MYCPU anal",
    "esil" : 1,
    "set_reg_profile" : set_reg_profile,
    "op" : op,
}

```

6. (Необязательный шаг) Чтобы добавить дополнительную информацию о размерах операций и выравнивании добавьте подфункцию `archinfo` и поместите ее указатель в структуре:

```

def mycpu_anal(a):
    def set_reg_profile():
        [...]
    def archinfo(query):

```

```

if query == R.R_ANAL_ARCHINFO_MIN_OP_SIZE:
    return 1
if query == R.R_ANAL_ARCHINFO_MAX_OP_SIZE:
    return 8
if query == R.R_ANAL_ARCHINFO_INV_OP_SIZE: # invalid op size
    return 2
return 0
def analop(memview, pc):
    [...]
    return {
        "name" : "mycpu",
        "arch" : "mycpu",
        "bits" : 32,
        "license" : "GPL",
        "desc" : "MYCPU anal",
        "esil" : 1,
        "set_reg_profile" : set_reg_profile,
        "archinfo": archinfo,
        "op" : op,
    }
}

```

7. Зарегистрируйте оба плагина, используя `r2lang.plugin("asm")` и `r2lang.plugin("anal")`, соответственно.

```

print("Registering MYCPU disasm plugin...")
print(r2lang.plugin("asm", mycpu))
print("Registering MYCPU analysis plugin...")
print(r2lang.plugin("anal", mycpu_anal))

```

Можно объединить все в один файл и загрузить его с помощью флага `-i`:

```
r2 -I mycpu.py some_file.bin
```

Можно загрузить плагин из оболочки r2: `#!python mycpu.py`

Смотрите также:

- Python,
- Javascript.

Реализация плагина формата в Python

Обратим внимание, что ради читабельности кода в следующих примерах отсутствуют конкретные реализации функций!

Нужно выполнить: 1. `import r2lang`, 2. Создайте функцию с подфункциями: `-load`, `-load_bytes`, `-destroy`, `-check_bytes`, `-baddr`, `-entries`, `-sections`, `-imports`, `-relocs`, `-binsym`, `-info`,

вернуть структуру плагина (для плагина RAsm):

```

def le_format(a):
    def load(binf):
        return [0]

    def check_bytes(buf):
        try:
            if buf[0] == 77 and buf[1] == 90:
                lx_off, = struct.unpack("<I", buf[0x3c:0x40])
                if buf[lx_off] == 76 and buf[lx_off+1] == 88:
                    return [1]
            return [0]
        except:
            return [0]

```

Проверяйте параметры всех функций и форматы возвращаемых ими данных. Обратите внимание, что функции `entries`, `sections`, `imports`, `relocs` возвращают список словарей специального вида, каждый со своим типом. Другие функции возвращают только список числовых значений, даже если оно всего одно. Есть специальная функция, которая возвращает информацию о файле - `info`:

```

def info(binf):
    return [
        "type" : "le",
        "bclass" : "le",
        "rclass" : "le",
        "os" : "OS/2",
        "subsystem" : "CLI",
        "machine" : "IBM",
        "arch" : "x86",
        "has_va" : 0,
        "bits" : 32,
        "big_endian" : 0,
        "dbg_info" : 0,
    ]
}

```

3. Эта структура должна содержать указатели на наиболее важные функции, такие как `check_bytes`, `load`, `load_bytes`, `entries`, `relocs` и `imports`.

```

return {
    "name" : "le",
    "desc" : "OS/2 LE/LX format",
    "license" : "GPL",
    "load" : load,
    "load_bytes" : load_bytes,
    "destroy" : destroy,
    "check_bytes" : check_bytes,
    "baddr" : baddr,
    "entries" : entries,
    "sections" : sections,
    "imports" : imports,
    "symbols" : symbols,
    "relocs" : relocs,
    "binsym" : binsym,
    "info" : info,
}

```

4. Затем нужно зарегистрировать новый плагин формата файла:

```

print("Registering OS/2 LE/LX plugin...")
print(r2lang.plugin("bin", le_format))

```

Отладка

Часто при проектировании плагинов возникают проблемы, особенно если вы делаете это для первый раз. Поэтому возможность их отладки очень важна. Первым шагом для отладки является задание переменной среды при запуске экземпляра radare2:

```

R_DEBUG=yes r2 /bin/ls
Loading /usr/local/lib/radare2/2.2.0-git//bin_xtr_dyldcache.so
Cannot find symbol 'radare_plugin' in library '/usr/local/lib/radare2/2.2.0-git//bin_xtr_dyldcache.so'
Cannot open /usr/local/lib/radare2/2.2.0-git//2.2.0-git
Loading /home/user/.config/radare2/plugins/asm_mips_ks.so
PLUGIN OK 0x55b205ea6070 fcn 0x7f298de08762
Loading /home/user/.config/radare2/plugins/asm_sparc_ks.so
PLUGIN OK 0x55b205ea6070 fcn 0x7f298de08762
Cannot open /home/user/.config/radare2/plugins/pimp
Cannot open /home/user/.config/radare2/plugins/yara
Loading /home/user/.config/radare2/plugins/asm_arm_ks.so
PLUGIN OK 0x55b205ea6070 fcn 0x7f298de08762
Loading /home/user/.config/radare2/plugins/core_yara.so
Module version mismatch /home/user/.config/radare2/plugins/core_yara.so (2.1.0) vs (2.2.0-git)
Loading /home/user/.config/radare2/plugins/asm_ppc_ks.so
PLUGIN OK 0x55b205ea6070 fcn 0x7f298de08762
Loading /home/user/.config/radare2/plugins/lang_python3.so
PLUGIN OK 0x55b205ea5ed0 fcn 0x7f298de08692
Loading /usr/local/lib/radare2/2.2.0-git/bin_xtr_dyldcache.so
Cannot find symbol 'radare_plugin' in library '/usr/local/lib/radare2/2.2.0-git/bin_xtr_dyldcache.so'
Cannot open /usr/local/lib/radare2/2.2.0-git/2.2.0-git
Cannot open directory '/usr/local/lib/radare2-extras/2.2.0-git'
Cannot open directory '/usr/local/lib/radare2-bindings/2.2.0-git'

```

```
USER CONFIG loaded from /home/user/.config/radare2/radare2rc
-- В визуальном режиме используйте клавишу 'c', для переключения в режим курсора. Используйте TAB для навигации
[0x000005520]>
```

Тестирование плагина

Плагин используется в rasm2 и r2. Проверим, правильно ли он загружается с помощью этой команды:

```
$ rasm2 -L | grep mycpu
_d  mycpu      My CPU disassembler (LGPL3)
```

Откроем пустой файл архитектуры `mycpu` и напишем там какой-нибудь случайный код.

```
$ r2 -
-- I endians swap
[0x00000000] > e asm.arch=mycpu
[0x00000000] > woR
[0x00000000] > pd 10
0x00000000    888e    mov r8, 14
0x00000002    b2a5    ifnot r10, r5
0x00000004    3f67    ret
0x00000006    7ef6    bl r15, r6
0x00000008    2701    xor r0, 1
0x0000000a    9826    mov r2, 6
0x0000000c    478d    xor r8, 13
0x0000000e    6b6b    store r6, 11
0x00000010    1382    add r8, r2
0x00000012    7f15    ret
```

Ура! Заработало... И теперь проверим, как он работает в командной строке!

```
r2 -nqamycpu -cwoR -cpd' 10' -
```

Реализация пакета r2pm для плагина

Напомним, что в radare2 включен собственный менеджер пакетов, можно легко добавлять новые плагины, чтобы они были доступны всем.

Все пакеты расположены в репозитории radare2-pm. Для описания структуры плагина используется очень простой текстовый формат.

```
R2PM_BEGIN
```

```
R2PM_GIT "https://github.com/user/mycpu"
R2PM_DESC "[r2-arch] дизассемблер и анализатор для MYCPU, плагин"

R2PM_INSTALL() {
    ${MAKE} clean
    ${MAKE} all || exit 1
    ${MAKE} install R2PM_PLUGDIR="${R2PM_PLUGDIR}"
}

R2PM_UNINSTALL() {
    rm -f "${R2PM_PLUGDIR}/asm_mycpu.*"
    rm -f "${R2PM_PLUGDIR}/anal_mycpu.*"
}
```

```
R2PM_END
```

Затем добавьте его в каталог /db репозитория radare2-pm и отправьте pull-запрос.

Задачи Crackmes

Задачи Crackmes (из «crack me» challenge) предназначены для решения инженерами, интересующимся обратным инжинирингом, для совершенствования своих навыков. В этом разделе приведены учебные материалы по приемам взлома файлов crackme с помощью r2.

Задачки IOLI CrackMes

Задачки IOLI crackme - хорошая отправная точка для изучения r2. Это набор разобранных примеров, основанных на пособии dustri

Копии задач доступны на локальном зеркале

IOLI 0x00

Первая задачка IOLI, самая простая.

```
$ ./crackme0x00
IOLI Crackme Level 0x00
Password: 1234
Invalid Password!
```

Первое, что проверяется - пароль, он может быть просто открытым текстом внутри файла. Если это так, то ломать дальше ничего не нужно: используем rabin2 с флагом -z для перечисления строк в двоичном файле.

```
$ rabin2 -z ./crackme0x00
[Strings]
nth paddr      vaddr      len size section type  string
-----
0  0x00000568 0x08048568 24  25  .rodata ascii IOLI Crackme Level 0x00\n
1  0x00000581 0x08048581 10  11  .rodata ascii Password:
2  0x0000058f 0x0804858f  6   7   .rodata ascii 250382
3  0x00000596 0x08048596 18  19  .rodata ascii Invalid Password!\n
4  0x000005a9 0x080485a9 15  16  .rodata ascii Password OK :)\n
```

Что представляет собой полученный текст? - это заголовок, отображаемый при запуске приложения.

```
nth paddr      vaddr      len size section type  string
-----
0  0x00000568 0x08048568 24  25  .rodata ascii IOLI Crackme Level 0x00\n
```

Эта строка - текст приглашения для ввода пароля.

```
1  0x00000581 0x08048581 10  11  .rodata ascii Password:
```

Эта - вывод сообщения об ошибке при вводе неверного пароля.

```
3  0x00000596 0x08048596 18  19  .rodata ascii Invalid Password!\n
```

Здесь говорится, что пароль принят.

```
4  0x000005a9 0x080485a9 15  16  .rodata ascii Password OK :)\n
```

А это что? Это строка, но ее не было при запуске приложения.

```
2  0x0000058f 0x0804858f  6   7   .rodata ascii 250382
```

Давайте попробуем ее в качестве пароля.

```
$ ./crackme0x00
IOLI Crackme Level 0x00
Password: 250382
Password OK :)
```

Ну вот, теперь ясно, что 250382 - это и есть пароль, на этом взлом закончен.

IOLI 0x01

Вторая задачка IOLI crackme.

```
$ ./crackme0x01
IOLI Crackme Level 0x01
Password: test
Invalid Password!
```

Посмотрим перечень строк при помощи rabin2.

```
$ rabin2 -z ./crackme0x01
[Strings]
nth paddr      vaddr      len size section type  string
-----
0  0x00000528 0x08048528 24  25  .rodata ascii IOLI Crackme Level 0x01\n
1  0x00000541 0x08048541 10  11  .rodata ascii Password:
2  0x0000054f 0x0804854f 18  19  .rodata ascii Invalid Password!\n
3  0x00000562 0x08048562 15  16  .rodata ascii Password OK :)\n
```

Видно, что взлом будет не на столько прост как в задаче 0x00. Попробуем дизассемблировать, используя r2.

```
$ r2 ./crackme0x01
-- Use `zoom.byte=printable` in zoom mode ('z' in Visual mode) to find strings
[0x08048330]> aa
[0x08048330]> pdf@main
; DATA XREF from entry0 @ 0x8048347
/ 113: int main (int argc, char **argv, char **envp);
|     ; var int32_t var_4h @ ebp-0x4
```

```

; var int32_t var_sp_4h @ esp+0x4
0x080483e4      55          push ebp
0x080483e5      89e5        mov ebp, esp
0x080483e7      83ec18     sub esp, 0x18
0x080483ea      83e4f0     and esp, 0xfffffffff0
0x080483ed      b800000000  mov eax, 0
0x080483f2      83c00f     add eax, 0xf           ; 15
0x080483f5      83c00f     add eax, 0xf           ; 15
0x080483f8      c1e804     shr eax, 4
0x080483fb      c1e004     shl eax, 4
0x080483fe      29c4       sub esp, eax
0x08048400      c70424288504. mov dword [esp], str.IOLI_Crackme_Level_0x01 ; [0x8048528:4]=0x494c4f49 ;
0x08048407      e810ffff    call sym.imp.printf      ; int printf(const char *format)
0x0804840c      c70424418504. mov dword [esp], str.Password; ; [0x8048541:4]=0x73736150 ; "Password: "
0x08048413      e804ffff    call sym.imp.printf      ; int printf(const char *format)
0x08048418      8d45fc     lea eax, [var_4h]
0x0804841b      89442404   mov dword [var_sp_4h], eax
0x0804841f      c704244c8504. mov dword [esp], 0x804854c ; [0x804854c:4]=0x49006425
0x08048426      e8e1feffff  call sym.imp.scanf      ; int scanf(const char *format)
0x0804842b      817dfc9a1400. cmp dword [var_4h], 0x149a
,=< 0x08048432      740e       je 0x8048442
| 0x08048434      c704244f8504. mov dword [esp], str.Invalid_Password ; [0x804854f:4]=0x61766e49 ; "Invalid
| 0x0804843b      e8dcfeffff  call sym.imp.printf      ; int printf(const char *format)
,==< 0x08048440      eb0c       jmp 0x804844e
|`-> 0x08048442      c70424628504. mov dword [esp], str.Password_OK_ ; [0x8048562:4]=0x73736150 ; "Password
| 0x08048449      e8cefeffff  call sym.imp.printf      ; int printf(const char *format)
| ; CODE XREF from main @ 0x8048440
`--> 0x0804844e      b800000000  mov eax, 0
0x08048453      c9          leave
0x08048454      c3          ret

```

Команда «aa» заставляет r2 проанализировать весь двоичный файл, в результате, среди прочего, получим имена символов (идентификаторов сущностей).

Комбинация ``pdf'' - это сокращение от

- Print (распечатать)
- Disassemble (в дизассемблированном виде)
- Function (функцию)

Команда распечатает код основной функции в виде ассемблерных инструкций, т.е. известной всем функции `main()`. В тексте много всего, включая странные имена, стрелки и т.д.

- ``imp.'' - сокращение от ``imports''. Это импортированные символы, такие как `printf()`
- ``str.'' - сокращение от ``strings''. Это, очевидно, строки.

Смотрим внимательно, находим инструкцию `cmp` с константой `0x149a`. `cmp` - инструкция сравнения в архитектуре x86, а `0x` - префикс, задающий 16-ричную систему счисления (hexadecimal, hex) для числа константы.

`0x0804842b 817dfc9a140. cmp dword [ebp + 0xfffffffffc], 0x149a`

Можно использовать команду `radare2 ?` для распечатки `0x149a` в другой системе счисления.

```
[0x08048330]> ? 0x149a
int32 5274
uint32 5274
hex 0x149a
octal 012232
unit 5.2K
segment 0000:049a
string "\x9a\x14"
fvalue: 5274.0
float: 0.000000f
double: 0.000000
binary 0b0001010010011010
trits 0t21020100
```

Теперь понятно, что `0x149a` равно `5274` в десятичной системе счисления. Давайте попробуем это в качестве пароля.

```
$ ./crackme0x01
IOLI Crackme Level 0x01
Password: 5274
Password OK :)
```

Бingo, пароль - `5274`. В этом случае функция пароля в `0x0804842b` сравнивала входные данные со значением `0x149a` (шестнадцатеричный формат). Поскольку вводят числа обычно в десятичной системе счисления, нетрудно догадаться, что

ввод пароля должен быть в десятичном формате или 5274. Поскольку мы хакеры, и любопытство движет нами, давайте посмотрим, что происходит, когда мы вводим в шестнадцатеричном виде.

```
$ ./crackme0x01
IOLI Crackme Level 0x01
Password: 0x149a
Invalid Password!
```

Стоило попытаться, но не сработало. Проблема в том, что `scanf()` принимает 0 в 0x149a как символ нуля, а остальные данные в связи с этим не принимаются как шестнадцатеричное значение.

На этом все с задачей IOLI 0x01.

IOLI 0x02

Третья задача.

```
$ ./crackme0x02
IOLI Crackme Level 0x02
Password: hello
Invalid Password!
```

Посмотрим файл с помощью rabin2.

```
$ rabin2 -z ./crackme0x02
[Strings]
nth paddr      vaddr      len size section type  string
-----
0  0x00000548 0x08048548 24  25    .rodata ascii IOLI Crackme Level 0x02\n
1  0x00000561 0x08048561 10  11    .rodata ascii Password:
2  0x0000056f 0x0804856f 15  16    .rodata ascii Password OK :)\n
3  0x0000057f 0x0804857f 18  19    .rodata ascii Invalid Password!\n
```

Аналогично задаче 0x01, здесь нет строки с явным паролем, проанализируем код с помощью r2.

```
[0x08048330]> aa
[x] Провести анализ всех флагов, начинающихся с sym. и entry0 (aa)
[0x08048330]> pdf@main
; DATA XREF from entry0 @ 0x8048347
/ 144: int main (int argc, char **argv, char **envp);
| ; var int32_t var_ch @ ebp-0xc
| ; var int32_t var_8h @ ebp-0x8
| ; var int32_t var_4h @ ebp-0x4
| ; var int32_t var_sp_4h @ esp+0x4
| 0x080483e4      55          push  ebp
| 0x080483e5      89e5        mov   ebp,  esp
| 0x080483e7      83ec18     sub   esp, 0x18
| 0x080483ea      83e4f0     and   esp, 0xffffffff
| 0x080483ed      b800000000  mov   eax, 0
| 0x080483f2      83c00f     add   eax, 0xf      ; 15
| 0x080483f5      83c00f     add   eax, 0xf      ; 15
| 0x080483f8      c1e804     shr   eax, 4
| 0x080483fb      c1e004     shl   eax, 4
| 0x080483fe      29c4        sub   esp, eax
| 0x08048400      c70424488504. mov   dword [esp], str.IOLI_Crackme_Level_0x02 ; [0x8048548:4]=0x494c4f49 ;
| 0x08048407      e810ffff    call  sym.imp.printf      ; int printf(const char *format)
| 0x0804840c      c70424618504. mov   dword [esp], str.Password: ; [0x8048561:4]=0x73736150 ; "Password: "
| 0x08048413      e804ffff    call  sym.imp.printf      ; int printf(const char *format)
| 0x08048418      8d45fc     lea   eax, [var_4h]
| 0x0804841b      89442404    mov   dword [var_sp_4h], eax
| 0x0804841f      c704246c8504. mov   dword [esp], 0x804856c ; [0x804856c:4]=0x50006425
| 0x08048426      e8e1ffff    call  sym.imp.scanf     ; int scanf(const char *format)
| 0x0804842b      c745f85a0000. mov   dword [var_8h], 0x5a ; 'Z' ; 90
| 0x08048432      c745f4ec0100. mov   dword [var_ch], 0x1ec ; 492
| 0x08048439      8b55f4     mov   edx, dword [var_ch]
| 0x0804843c      8d45f8     lea   eax, [var_8h]
| 0x0804843f      0110        add   dword [eax], edx
| 0x08048441      8b45f8     mov   eax, dword [var_8h]
| 0x08048444      0faf45f8    imul  eax, dword [var_8h]
| 0x08048448      8945f4     mov   dword [var_ch], eax
| 0x0804844b      8b45fc     mov   eax, dword [var_4h]
| 0x0804844e      3b45f4     cmp   eax, dword [var_ch]
,=< 0x08048451      750e        jne   0x8048461
| 0x08048453      c704246f8504. mov   dword [esp], str.Password_OK_: ; [0x804856f:4]=0x73736150 ; "Password_"
| 0x0804845a      e8bdffff    call  sym.imp.printf      ; int printf(const char *format)
,==< 0x0804845f      eb0c        jmp   0x804846d
|`-> 0x08048461      c704247f8504. mov   dword [esp], str.Invalid_Password ; [0x804857f:4]=0x61766e49 ; "Invalid_P
| 0x08048468      e8affeffff  call  sym.imp.printf      ; int printf(const char *format)
| ; CODE XREF from main @ 0x804845f
```

```

| `--> 0x0804846d      b800000000    mov eax, 0
|   0x08048472      c9             leave
\   0x08048473      c3             ret

```

Имея опыт решения crackme0x02, сначала ищем инструкцию `cmp` при помощи простой команды:

```
[0x08048330]> pdf@main | grep cmp
| 0x0804844e      3b45f4        cmp eax, dword [var_ch]
```

К сожалению, переменная, сравниваемая с `eax`, хранится где-то в стеке. Проверить значение этой переменной напрямую невозможно - распространенный случай в реверс-инжениринге. Нужно вычислить значение переменной, анализируя инструкции, стоящие перед `cmp`. Поскольку объем кода относительно невелик, это вполне возможно.

Пример:

```

|      0x080483ed      b800000000    mov eax, 0
|      0x080483f2      83c00f       add eax, 0xf          ; 15
|      0x080483f5      83c00f       add eax, 0xf          ; 15
|      0x080483f8      c1e804       shr eax, 4
|      0x080483fb      c1e004       shl eax, 4
|      0x080483fe      29c4         sub esp, eax

```

Легко получается значение в регистре `eax`. Оно равно `0x16`.

Становится сложновато, когда объем кода растет. Radare2 позволяет выводить дизассемблированный код в формате подобном C, что весьма бывает полезно.

```
[0x08048330]> pdc@main
function main () {
    // 4 функциональных блока

loc_0x80483e4:

    //DATA XREF from entry0 @ 0x8048347
    push ebp
    ebp = esp
    esp -= 0x18
    esp &= 0xffffffff
    eax = 0
    eax += 0xf           //15
    eax += 0xf           //15
    eax >>= 4
    eax <<= 4
    esp -= eax
    dword [esp] = "IOLI Crackme Level 0x02\n" //@[0x8048548:4]=0x494c4f49 ; str.IOLI_Crackme_Level_0x02 ; const char ch

    int printf("IOLI Crackme Level 0x02\n")
    dword [esp] = "Password: " //@[0x8048561:4]=0x73736150 ; str.Password: ; const char *format

    int printf("Password: ")
    eax = var_4h
    dword [var_sp_4h] = eax
    dword [esp] = 0x804856c //@[0x804856c:4]=0x50006425 ; const char *format
    int scanf("%d")           //sym.imp.scanf ()
    dword [var_8h] = 0x5a    //'Z' ; 90
    dword [var_ch] = 0x1ec   //492
    edx = dword [var_ch]
    eax = var_8h           //"Z"
    dword [eax] += edx
    eax = dword [var_8h]
    eax = eax * dword [var_8h]
    dword [var_ch] = eax
    eax = dword [var_4h]
    var = eax - dword [var_ch]
    if (var) goto 0x8048461 //likely
{
    loc_0x8048461:

    //CODE XREF from main @ 0x8048451
    dword [esp] = s"Invalid Password!\n"//@[0x804857f:4]=0x61766e49 ; str.Invalid_Password ; const char *format

    int printf("Invalid ")
do
{
    loc_0x804846d:

    //CODE XREF from main @ 0x804845f
    eax = 0
    leave                  //pstr 0x0804857f) "Invalid Password!\n" ebp ; str.Invalid_Password
```

```

        return
    } while (?);
} while (?);
}
return;
}
}
```

Команда pdc ненадежна, особенно при анализе циклов обработки данных (while, for и т.д.). Поэтому предпочтительно использовать плагин r2dec в репозитории r2 для генерации псевдокода C. Он легко устанавливается:

```
r2pm install r2dec
```

декомпилируем main(), используя команду (типа F5 в IDA):

```
[0x08048330]> pdd@main
/* r2dec pseudo code output */
/* ./crackme0x02 @ 0x80483e4 */
#include <stdint.h>
```

```

int32_t main (void) {
    uint32_t var_ch;
    int32_t var_8h;
    int32_t var_4h;
    int32_t var_sp_4h;
    eax = 0;
    eax += 0xf;
    eax += 0xf;
    eax >>= 4;
    eax <<= 4;
    printf ("IOLI Crackme Level 0x02\n");
    printf ("Password: ");
    eax = &var_4h;
    *((esp + 4)) = eax;
    scanf (0x804856c);
    var_8h = 0x5a;
    var_ch = 0x1ec;
    edx = 0x1ec;
    eax = &var_8h;
    *(eax) += edx;
    eax = var_8h;
    eax *= var_8h;
    var_ch = eax;
    eax = var_4h;
    if (eax == var_ch) {
        printf ("Password OK :)\n");
    } else {
        printf ("Invalid Password!\n");
    }
    eax = 0;
    return eax;
}
```

Теперь это более человекочитабельно. Чтобы посмотреть строку в 0x804856c, можно: * seek (установить смещение), * print string (распечатать строку).

```
[0x08048330]> s 0x804856c
[0x0804856c]> ps
%d
```

Получили строку формата, используемую в `scanf()`. Однако r2dec не распознает второй аргумент (eax) - указатель. Он указывает на `var_4h` и означает, что входные данные будут именно там храниться.

Теперь можно легко выписать псевдокод.

```
var_ch = (var_8h + var_ch)^2;
if (var_ch == our_input)
    printf("Password OK :)\n");
```

Учитывая, что первоначальное значение переменной var_8h равно 0x5a, а var_ch равно 0x1ec, получаем var_ch = 338724 (0x52b24):

```
$ rax2 '=10' '(0x5a+0x1ec)*(0x5a+0x1ec)'  
338724  
  
$ ./crackme0x02  
IOLI Crackme Level 0x02  
Password: 338724  
Password OK :)
```

Вот и все с crackme0x02.

IOLI 0x03

Задача crackme 0x03. Пропустим поиск строк и сразу все проанализируем.

```
[0x08048360]> aaa  
[0x08048360]> pd@sym.main  
/* r2dec pseudo code output */  
/* ./crackme0x03 @ 0x8048498 */  
#include <stdint.h>  
  
int32_t main (void) {  
    int32_t var_ch;  
    int32_t var_8h;  
    int32_t var_4h;  
    int32_t var_sp_4h;  
    eax = 0;  
    eax += 0xf;  
    eax += 0xf;  
    eax >= 4;  
    eax <= 4;  
    printf ("IOLI Crackme Level 0x03\n");  
    printf ("Password: ");  
    eax = &var_4h;  
    scanf (0x8048634, eax);  
    var_8h = 0x5a;  
    var_ch = 0x1ec;  
    edx = 0x1ec;  
    eax = &var_8h;  
    *(eax) += edx;  
    eax = var_8h;  
    eax *= var_8h;  
    var_ch = eax;  
    eax = var_4h;  
    test (eax, eax);  
    eax = 0;  
    return eax;  
}
```

Выглядит просто, за исключением функции `test(eax, eax)`. Необычно вызывать функцию с двумя одинаковыми параметрами, можно предположить, что декомпиляция прошла не совсем так, как требуется. Посмотрим код в виде инструкций процессора.

```
[0x08048360]> pdf@sym.main  
...  
0x080484fc      8945f4      mov dword [var_ch], eax  
0x080484ff      8b45f4      mov eax, dword [var_ch]  
0x08048502      89442404    mov dword [var_sp_4h], eax ; uint32_t arg_ch  
0x08048506      8b45fc      mov eax, dword [var_4h]  
0x08048509      890424      mov dword [esp], eax ; int32_t arg_8h  
0x0804850c      e85dffff    call sym.test  
...
```

В наличии `sym.test`, вызываемый с двумя параметрами. Один параметр - `var_4h` (наш ввод из `scanf()`). Другой - `var_ch`. Значение `var_ch` (как параметр `test()`) можно вычислить, как это было в задаче `crackme_0x02`. Оно равно 0x52b24.

Пробуем!

```
./crackme0x03
IOLI Crackme Level 0x03
Password: 338724
Password OK!!! :)
```

Смотрим на `sym.test`. Это условный переход с двумя ветвями, сравнивающий два параметра, потом производится сдвиг (shift). Догадка состоит в том, сдвиг, скорее всего, является частью некоторой процедуры расшифровки (шифр сдвига, например, шифр Цезаря).

```
/* r2dec pseudo code output */
/* ./crackme0x03 @ 0x804846e */
#include <stdint.h>

int32_t test (int32_t arg_8h, uint32_t arg_ch) {
    eax = arg_8h;
    if (eax != arg_ch) {
        shift ("Lqydolg#Sdvvzrug$");
    } else {
        shift ("Sdvvzrug#RN$$$$#=,");
    }
    return eax;
}
```

Взломаем `shift()` - удовлетворим наше любопытство.

```
[0x08048360]> pdf@sym.shift
; CODE (CALL) XREF 0x08048491 (sym.test)
; CODE (CALL) XREF 0x08048483 (sym.test)
/ function: sym.shift (90)
| 0x08048414  sym.shift:
| 0x08048414      55          push  ebp
| 0x08048415      89e5        mov   ebp, esp
| 0x08048417      81ec98000000  sub   esp, 0x98
| 0x0804841d      c7458400000000  mov   dword [ebp-0x7c], 0x0 ; this seems to be a counter
. ; CODE (JMP) XREF 0x0804844e (sym.shift)
/ loc: loc.08048424 (74)
| . 0x08048424  loc.08048424:
| .--> 0x08048424      8b4508  mov   eax, [ebp+0x8] ; ebp+0x8 = strlen(chain)
| | 0x08048427      890424  mov   [esp], eax
| | 0x0804842a      e811ffff  call  dword imp.strlen
| | ; imp.strlen()
| | 0x0804842f      394584  cmp   [ebp-0x7c], eax
| | ,=< 0x08048432      731c  jae   loc.08048450
| | | 0x08048434      8d4588  lea   eax, [ebp-0x78]
| | | 0x08048437      89c2  mov   edx, eax
| | | 0x08048439      035584  add   edx, [ebp-0x7c]
| | | 0x0804843c      8b4584  mov   eax, [ebp-0x7c]
| | | 0x0804843f      034508  add   eax, [ebp+0x8]
| | | 0x08048442      0fb600  movzx eax, byte [eax]
| | | 0x08048445      2c03  sub   al, 0x3
| | | 0x08048447      8802  mov   [edx], al
| | | 0x08048449      8d4584  lea   eax, [ebp-0x7c]
| | | 0x0804844c      ff00  inc   dword [eax]
| `=< 0x0804844e      ebd4  jmp   loc.08048424
| ; CODE (JMP) XREF 0x08048432 (sym.shift)
/ loc: loc.08048450 (30)
| | 0x08048450  loc.08048450:
|`-> 0x08048450      8d4588  lea   eax, [ebp-0x78]
| | 0x08048453      034584  add   eax, [ebp-0x7c]
| | 0x08048456      c60000  mov   byte [eax], 0x0
| | 0x08048459      8d4588  lea   eax, [ebp-0x78]
| | 0x0804845c      89442404  mov   [esp+0x4], eax
| | 0x08048460      c70424e8850408  mov   dword [esp], 0x80485e8
```

```

| 0x08048467    e8e4fffff    call dword imp.printf
| ; imp.printf()
| 0x0804846c    c9           leave
\ 0x0804846d    c3           ret
; -----

```

Прочитав ассемблерный код, обнаруживаем, что расшифровка на самом деле - ``sub al, 0x3''. Напишем программу python:

```

print(''.join([chr(ord(i)-0x3) for i in 'SdvvzrugRN$$'])))
print(''.join([chr(ord(i)-0x3) for i in 'LqydolgSdvvzrug$'])))

```

Проще запустить этот код расшифровки, то есть отладить его или эмулировать. Попытка использования эмулятора ESIL radare2 провалилась, он завис при выполнении `call dword imp.strlen`. Также не удалось воспользоваться функцией hooking / пропуск инструкции в radare2. Ниже приведен пример эмуляции ESIL.

```

[0x08048414]> s 0x08048445      # the 'sub al, 0x3'
[0x08048445]> aei                 # init VM
[0x08048445]> aeim                # init memory
[0x08048445]> aeip                # init ip
[0x08048445]> aer eax=0x41        # set eax=0x41 -- 'A'
[0x08048445]> aer                 # show current value of regs
oeax = 0x00000000
eax = 0x00000041
ebx = 0x00000000
ecx = 0x00000000
edx = 0x00000000
esi = 0x00000000
edi = 0x00000000
esp = 0x00178000
ebp = 0x00178000
eip = 0x08048445
eflags = 0x00000000
[0x08048445]> V                  # enter Visual mode
# 'p' or 'P' to change visual mode
# I prefer the [xDvc] mode
# use 's' to step in and 'S' to step over
[0x08048442 [xDvc]0 0% 265 ./crackme0x03]> diq;?0;f t.. @ sym.shift+46 # 0x8048442
dead at 0x00000000
- offset - 0 1 2 3 4 5 6 7 8 9 A B C D E F 0123456789ABCDEF
0x00178000 0000 0000 0000 0000 0000 0000 0000 0000 ..... .
0x00178010 0000 0000 0000 0000 0000 0000 0000 0000 ..... .
0x00178020 0000 0000 0000 0000 0000 0000 0000 0000 ..... .
0x00178030 0000 0000 0000 0000 0000 0000 0000 0000 ..... .
oeax 0x00000000     eax 0x00000041     ebx 0x00000000     ecx 0x00000000
edx 0x00000000     esi 0x00000000     edi 0x00000000     esp 0x00178000
ebp 0x00178000     eip 0x08048445     eflags 0x00000000
: 0x08048442       0fb600          movzx eax, byte [eax]
: ;-- eip:
: 0x08048445       2c03           sub al, 3
: 0x08048447       8802           mov byte [edx], al
: 0x08048449       8d4584         lea eax, [var_7ch]
: 0x0804844c       ff00           inc dword [eax]
:=< 0x0804844e     ebd4           jmp 0x8048424
; CODE XREF from sym.shift @ 0x8048432
0x08048450       8d4588         lea eax, [var_78h]

```

Кстати, можно открыть файл и использовать команду ``write data'' для расшифровки данных.

```

r2 -w ./crackme0x03
[0x08048360]> aaa
[0x08048360]> fs strings
[0x08048360]> f
0x080485ec 18 str.Lqydolg_Sdvvzrug
0x080485fe 18 str.Sdvvzrug_RN
0x08048610 25 str.IOLI_Crackme_Level_0x03
0x08048629 11 str.Password:

```

```
[0x08048360]> s str.Lqydolg_Sdvvzrug
[0x080485ec]> wos 0x03 @ str.Lqydolg_Sdvvzrug!0x11
[0x080485ec]> px
- offset - 0 1 2 3 4 5 6 7 8 9 A B C D E F 0123456789ABCDEF
0x080485ec 496e 7661 6c69 6420 5061 7373 776f 7264 Invalid Password
0x080485fc 2100 5364 7676 7a72 7567 2352 4e24 2424 !.Sdvvzrug#RN$$$#
0x0804860c 233d 2c00 494f 4c49 2043 7261 636b 6d65 #=,.IOLI Crackme
0x0804861c 204c 6576 656c 2030 7830 330a 0050 6173 Level 0x03..Pas
0x0804862c 7377 6f72 643a 2000 2564 0000 0000 0000 sword: .%d.....
[0x080485ec]> wos 0x03 @ str.Sdvvzrug_RN!17
[0x080485ec]> px
- offset - 0 1 2 3 4 5 6 7 8 9 A B C D E F 0123456789ABCDEF
0x080485ec 496e 7661 6c69 6420 5061 7373 776f 7264 Invalid Password
0x080485fc 2100 5061 7373 776f 7264 204f 4b21 2121 !.Password OK!!!
0x0804860c 203a 2900 494f 4c49 2043 7261 636b 6d65 :).IOLI Crackme
0x0804861c 204c 6576 656c 2030 7830 330a 0050 6173 Level 0x03..Pas
0x0804862c 7377 6f72 643a 2000 2564 0000 0000 0000 sword: .%d.....
[0x080485ec]>
```

IOLI 0x04

0x04

```
[0x080483d0]> pdd@main
/* r2dec pseudo code output */
/* ./crackme0x04 @ 0x8048509 */
#include <stdint.h>

int32_t main (void) {
    int32_t var_78h;
    int32_t var_4h;
    eax = 0;
    eax += 0xf;
    eax += 0xf;
    eax >>= 4;
    eax <<= 4;
    printf ("IOLI Crackme Level 0x04\n");
    printf ("Password: ");
    eax = &var_78h;
    scanf (0x8048682, eax);
    eax = &var_78h;
    check (eax);
    eax = 0;
    return eax;
}
```

Входим в функцию check.

```
#include <stdint.h>

int32_t check (char * s) {
    char * var_dh;
    uint32_t var_ch;
    uint32_t var_8h;
    int32_t var_4h;
    char * format;
    int32_t var_sp_8h;
    var_8h = 0;
    var_ch = 0;
    do {
        eax = s;
        eax = strlen (eax);
        if (var_ch >= eax) {
```

```

        goto label_0;
    }
    eax = var_ch;
    eax += s;
    eax = *(eax);
    var_dh = al;
    eax = &var_4h;
    eax = &var_dh;
    sscanf (eax, eax, 0x8048638);
    edx = var_4h;
    eax = &var_8h;
    *(eax) += edx;
    if (var_8h == 0xf) {
        printf ("Password OK!\n");
        exit (0);
    }
    eax = &var_ch;
    *(eax)++;
} while (1);
label_0:
    printf ("Password Incorrect!\n");
    return eax;
}

```

Анализируя вручную ассемблерный код и псевдоинструкции, можно просто выписать C-подобный код для описания этой функции:

```

#include <stdint.h>
int32_t check(char *s)
{
    var_ch = 0;
    var_8h = 0;
    for (var_ch = 0; var_ch < strlen(s); ++var_ch)
    {
        var_dh = s[var_ch];
        sscanf(&var_dh, %d, &var_4h);           // read from string[var_ch], store to var_4h
        var_8h += var_4h;
        if(var_8h == 0xf)
            printf("Password OK\n");
    }
    printf("Password Incorrect!\n");
    return 0;
}

```

Короче говоря, вычисляется сумма цифр некоторого числа (сложение цифры за цифрой), например, $96 \Rightarrow 9 + 6 = 15$:

```

./crackme0x04
IOLI Crackme Level 0x04
Password: 12345
Password OK!

```

```

./crackme0x04
IOLI Crackme Level 0x04
Password: 96
Password OK!

```

IOLI 0x05

Смотрим код, в нем используется `scanf()` для получения ввода, затем ввод передается в функцию `check()` в качестве параметра.

```

[0x080483d0]> pdd@main
/* r2dec pseudo code output */
/* ./crackme0x05 @ 0x8048540 */
#include <stdint.h>

```

```

int32_t main (void) {
    int32_t var_78h;
    int32_t var_4h;
    eax = 0;
    eax += 0xf;
    eax += 0xf;
    eax >>= 4;
    eax <<= 4;
    printf ("IOLI Crackme Level 0x05\n");
    printf ("Password: ");
    eax = &var_78h;
    scanf (0x80486b2, eax);           // 0x80486b2 is %s
    eax = &var_78h;
    check (eax);
    eax = 0;
    return eax;
}

```

Уже встречавшаяся функция check():

```

/* r2dec pseudo code output */
/* ./crackme0x05 @ 0x80484c8 */
#include <stdint.h>

int32_t check (char * s) {
    char * var_dh;
    uint32_t var_ch;
    uint32_t var_8h;
    int32_t var_4h;
    char * format;
    int32_t var_sp_8h;
    var_8h = 0;
    var_ch = 0;
    do {
        eax = s;
        eax = strlen (eax);
        if (var_ch >= eax) {
            goto label_0;
        }
        eax = var_ch;
        eax += s;
        eax = *(eax);
        var_dh = al;
        eax = &var_4h;
        eax = &var_dh;
        sscanf (eax, eax, 0x8048668);      // 0x8048668 is %d
        edx = var_4h;
        eax = &var_8h;
        *(eax) += edx;
        if (var_8h == 0x10) {
            eax = s;
            parell (eax);
        }
        eax = &var_ch;
        *(eax)++;
    } while (1);
label_0:
    printf ("Password Incorrect!\n");
    return eax;
}

```

Точно так же можно написать наш собственный С-подобный псевдокод.

```
#include <stdint.h>
int32_t check(char *s)
```

```

{
    var_ch = 0;
    var_8h = 0;
    for (var_ch = 0; var_ch < strlen(s); ++var_ch)
    {
        var_dh = s[var_ch];
        sscanf(&var_dh, %d, &var_4h);           // read from string[var_ch], store to var_4h
        var_8h += var_4h;
        if(var_8h == 0x10)
            parell(s);
    }
    printf("Password Incorrect!\n");
    return 0;
}

```

Условие в if - var_8h == 0x10. Кроме того, новый вызов функции - parell(s) заменяет теперь printf("пароль OK") из предыдущей задачи. Следующим шагом является взлом sym.parell.

```

[0x08048484]> s sym.parell
[0x08048484]> pdd@sym.parell
/* r2dec pseudo code output */
/* ./crackme0x05 @ 0x8048484 */
#include <stdint.h>

uint32_t parell (char * s) {
    int32_t var_4h;
    char * format;
    int32_t var_8h;
    eax = &var_4h;
    eax = s;
    sscanf (eax, eax, 0x8048668);
    eax = var_4h;
    eax &= 1;
    if (eax == 0) {
        printf ("Password OK!\n");
        exit (0);
    }
    return eax;
}

```

С декомпилированным кодом все в порядке, за исключением функции `sscanf()`. Его можно легко исправить, посмотрев ассемблерный код.

```

/ 68: sym.parell (int32_t arg_8h);
    ; var int32_t var_4h @ ebp-0x4
    ; arg int32_t arg_8h @ ebp+0x8
    ; var int32_t var_sp_4h @ esp+0x4
    ; var int32_t var_8h @ esp+0x8
    0x08048484      55          push ebp
    0x08048485      89e5        mov ebp, esp
    0x08048487      83ec18     sub esp, 0x18
    0x0804848a      8d45fc     lea eax, [var_4h]
    0x0804848d      89442408   mov dword [var_8h], eax
    0x08048491      c74424046886. mov dword [var_sp_4h], 0x8048668 ; [0x8048668:4]=0
    0x08048499      b84508     mov eax, dword [arg_8h]
    0x0804849c      890424     mov dword [esp], eax
    0x0804849f      e800ffff    call sym.imp.sscanf           ; int sscanf(const char *
...

```

Инструкция `mov dword [esp], eax` является ближайшей к `sscanf` (она эквивалентна инструкции `push`). Он сохраняет строку 's' на вершине стека (arg1). `mov dword [var_sp_4h], 0x8048668` засыпает '%d' как arg2 в стек. Переменная `var_8h` (`esp + 0x8`) хранит адрес `var_4h`, т.е. `arg3`.

Наконец, исправленный псевдокод -

```
uint32_t parell (char * s) {
```

```

sscanf (s, %d, &var_4h);
if ((var_4h & 1) == 0) {
    printf ("Password OK!\n");
    exit(0);
}
return 0;
}

```

Выявлены два ограничения:

- Сумма цифр должна быть равна 16 (0x10),
- Должно быть нечетным числом (1 & number == 0).

Пароль теперь у нас под рукой.

```

./crackme0x05
IOLI Crackme Level 0x05
Password: 88
Password OK!

```

```

./crackme0x05
IOLI Crackme Level 0x05
Password: 12346
Password OK!

```

Также можно использовать angr для решения этой задачи, так как заданы два ограничения на пароль.

IOLI 0x06

Долгий процесс анализа этого двоичного файла (представление результата далее в тексте не полное):

```

rabin2 -z ./crackme0x06
[Строки]
nth paddr      vaddr      len size section type  string
-----
0  0x00000738  0x08048738  4   5   .rodata ascii LOLO
1  0x00000740  0x08048740  13  14  .rodata ascii Password OK!\n
2  0x0000074e  0x0804874e  20  21  .rodata ascii Password Incorrect!\n
3  0x00000763  0x08048763  24  25  .rodata ascii IOLI Crackme Level 0x06\n
4  0x0000077c  0x0804877c  10  11  .rodata ascii Password:

```

```

rabin2 -I ./crackme0x06
arch      x86
baddr     0x8048000
bintype   elf
bits      32
compiler  GCC: (GNU) 3.4.6 (Gentoo 3.4.6-r2, ssp-3.4.6-1.0, pie-8.7.10)
crypto    false
endian   little
havecode  true
lang      c
machine   Intel 80386
maxopsz  16
minopsz  1
os        linux
static    false
va        true

```

Проанализировав ответ, декомпилируем main

```

[0x08048400]> pd @main
/* r2dec pseudo code output */
/* ./crackme0x06 @ 0x8048607 */
#include <stdint.h>

int32_t main (int32_t arg_10h) {

```

```

int32_t var_78h;
int32_t var_4h;
// выравнивание стека
eax = 0;
eax += 0xf;
eax += 0xf;
eax >= 4;
eax <= 4;

// основная логика
printf ("IOLI Crackme Level 0x06\n");
printf ("Password: ");
eax = &var_78h;
scanf (0x8048787, eax);
eax = arg_10h;
eax = &var_78h;
check (eax, arg_10h);
eax = 0;
return eax;
}

```

Функции main передаются три аргумента argc, argv, envp, и эта программа компилирована компилятором GCC, поэтому стек должен быть таким:

```

[esp + 0x10] - envp
[esp + 0x0c] - argv
[esp + 0x08] - argc
[esp + 0x04] - return address

```

Входим в функцию check() и декомпилируем ее. Функция отличается от той, что была в задаче 0x05, но они по-прежнему имеют схожие структуры кода.

```

int32_t check (char * s, int32_t arg_ch) {
    char * var_dh;
    uint32_t var_ch;
    uint32_t var_8h;
    int32_t var_4h;
    char * format;
    int32_t var_sp_8h;
    var_8h = 0;
    var_ch = 0;
    do {
        eax = s;
        eax = strlen (eax);
        if (var_ch >= eax) {
            goto label_0;
        }
        eax = var_ch;
        eax += s;
        eax = *(eax);
        var_dh = al;
        eax = &var_4h;
        eax = &var_dh;
        sscanf (eax, eax, 0x804873d);
        edx = var_4h;
        eax = &var_8h;
        *(eax) += edx;
        if (var_8h == 0x10) {
            eax = arg_ch;
            eax = s;
            parell (eax, arg_ch);
        }
        eax = &var_ch;
        *(eax)++;
    } while (1);
}

```

```

label_0:
    printf ("Password Incorrect!\n");
    return eax;
}

```

Надо исправить `sscanf` и `parell`, обе они были сгенерированы неправильно:

```

int32_t check (char * s, void* envp)
{
    var_ch = 0;
    var_8h = 0;
    for (var_ch = 0; var_ch < strlen(s); ++var_ch)
    {
        var_dh = s[var_ch];
        sscanf(&var_dh, %d, &var_4h);           // read from string[var_ch], store to var_4h
        var_8h += var_4h;
        if(var_8h == 0x10)
            parell(s, envp);
    }
    printf("Password Incorrect!\n");
    return 0;
}

```

Больше никакой информации, надо снова взламывать `parell()`.

```

#include <stdint.h>

uint32_t parell (char * s, char * arg_ch) {
    sscanf (s, %d, &var_4h);

    if (dummy (var_4h, arg_ch) == 0)
        return 0;

    for (var_bp_8h = 0; var_bp_8h <= 9; ++var_bp_8h){
        if (var_4h & 1 == 0){
            printf("Password OK!\n");
            exit(0);
        }
    }

    return 0;
}

```

Появилось новое условие в `parell()` -- `dummy (var_4h, arg_ch) == 0`. Взламываем `dummy!`

```

[0x080484b4]> pdd@sym.dummy
/* r2dec pseudo code output */
/* ./crackme0x06 @ 0x80484b4 */
#include <stdint.h>

```

```

int32_t dummy (char ** s1) {
    int32_t var_8h;
    int32_t var_4h;
    char * s2;
    size_t * n;
    var_4h = 0;
    do {
        eax = 0;
        edx = eax*4;
        eax = s1;
        if ((*((edx + eax))) == 0) {
            goto label_0;
        }
        eax = var_4h;
        ecx = eax*4;
        edx = s1;
    }
}

```

```

    eax = &var_4h;
    *(eax)++;
    eax = *((ecx + edx));
    eax = strncmp(eax, 3, "LOLO");
} while (eax != 0);
var_8h = 1;
goto label_1;
label_0:
var_8h = 0;
label_1:
eax = 0;
return eax;
}

```

Выглядит как цикл обработки строки, можем ``украсить'' его.

```

[0x080484b4]> pdd@sym.dummy
/* r2dec pseudo code output */
/* ./crackme0x06 @ 0x80484b4 */
#include <stdint.h>

int32_t dummy (char ** s1) {
    for (var_4h = 0; strncmp(s1[var_4h], "LOLO", 3) != 0; var_4h++){
        if (s1[i] == NULL)
            return 0;
    }
    return 1;
}

```

Теперь имеем три ограничения в crackme_0x06:

- сумма цифр,
- нечетное число,
- должна быть переменная среди, имя которой начинается с ``LOL''.

```

$ ./crackme0x06
IOLI Crackme Level 0x06
Password: 12346
Password Incorrect!
$ export LOLAA=help
$ ./cracke0x06
IOLI Crackme Level 0x06
Password: 12346
Password OK!

```

IOLI 0x07

Странная строка ``wtf?''.

```

$ rabin2 -z ./crackme0x07
[Строки]
nth paddr      vaddr      len size section type  string
-----
0 0x000007a8 0x080487a8 4   5     .rodata ascii LOLO
1 0x000007ad 0x080487ad 20  21    .rodata ascii Password Incorrect!\n
2 0x000007c5 0x080487c5 13  14    .rodata ascii Password OK!\n
3 0x000007d3 0x080487d3 5   6     .rodata ascii wtf?\n
4 0x000007d9 0x080487d9 24  25    .rodata ascii IOLI Crackme Level 0x07\n
5 0x000007f2 0x080487f2 10  11    .rodata ascii Password:

```

Опять же, нет строки пароля или сравнений в main(). Приведем здесь уже упрощенный псевдокод. Тип переменной var_78h вероятно char *pointer (string).

```

#include <stdint.h>
int32_t main (int32_t arg_10h) {
    printf ("IOLI Crackme Level 0x07\n");
}

```

```

printf ("Password: ");
scanf ("%s, &var_78h);
return fcn_080485b9 (&var_78h, arg_10h);
}

```

Из-за потери информации ни аа, ниааа не определили названий функций. Можно дважды проверить ``flagspace''. Radare2 использует fcn_080485b9 в качестве имен функций - распространенный случай в реверс-инженеринге, когда нет никакой информации о символах в двоичном файле.

```

[0x080487fd]> fs symbols
[0x080487fd]> f
0x08048400 33 entry0
0x0804867d 92 main
0x080487a4 4 obj._IO_stdin_used

```

Декомпилируем fcn_080485b9():

```

[0x080485b9]> pdfc
; CALL XREF from main @ 0x80486d4
/ 118: fcn.080485b9 (char *s, int32_t arg_ch);
    ; var char *var_dh @ ebp-0xd
    ; var signed int var_ch { >= 0xfffffffffffffff } @ ebp-0xc
    ; var uint32_t var_8h @ ebp-0x8
    ; var int32_t var_bp_4h @ ebp-0x4
    ; arg char *s @ ebp+0x8
    ; arg int32_t arg_ch @ ebp+0xc
    ; var char *format @ esp+0x4
    ; var int32_t var_sp_8h @ esp+0x8
0x080485b9      55          push ebp
0x080485ba      89e5        mov ebp, esp
0x080485bc      83ec28     sub esp, 0x28
0x080485bf      c745f8000000. mov dword [var_8h], 0
0x080485c6      c745f4000000. mov dword [var_ch], 0
; CODE XREF from fcn.080485b9 @ 0x8048628
.-> 0x080485cd      8b4508     mov eax, dword [s]
.: 0x080485d0      890424     mov dword [esp], eax      ; const char *s
.: 0x080485d3      e8d0fdffff  call sym.imp.strlen      ; size_t strlen(const
.: 0x080485d8      3945f4     cmp dword [var_ch], eax
,==< 0x080485db      734d      jae 0x804862a
|.: 0x080485dd      8b45f4     mov eax, dword [var_ch]
|.: 0x080485e0      034508     add eax, dword [s]
|.: 0x080485e3      0fb600    movzx eax, byte [eax]
|.: 0x080485e6      8845f3     mov byte [var_dh], al
|.: 0x080485e9      8d45fc     lea eax, [var_bp_4h]
|.: 0x080485ec      89442408   mov dword [var_sp_8h], eax ; ...
|.: 0x080485f0      c7442404c287. mov dword [format], 0x80487c2 ; [0x80487c2:4]=0x50
|.: ;-- eip:
|.: 0x080485f8      8d45f3     lea eax, [var_dh]
|.: 0x080485fb      890424     mov dword [esp], eax      ; const char *s
|.: 0x080485fe      e8c5fdffff  call sym.imp.sscanf      ; int sscanf(const char
|.: 0x08048603      8b55fc     mov edx, dword [var_bp_4h]
|.: 0x08048606      8d45f8     lea eax, [var_8h]
|.: 0x08048609      0110      add dword [eax], edx
|.: 0x0804860b      837df810   cmp dword [var_8h], 0x10
,==< 0x0804860f      7512      jne 0x8048623
||.: 0x08048611      8b450c     mov eax, dword [arg_ch]
||.: 0x08048614      89442404   mov dword [format], eax      ; char *arg_ch
||.: 0x08048618      8b4508     mov eax, dword [s]
||.: 0x0804861b      890424     mov dword [esp], eax      ; char *
||.: 0x0804861e      e81fffffff  call fcn.08048542
||.: ; CODE XREF from fcn.080485b9 @ 0x804860f
`---> 0x08048623      8d45f4     lea eax, [var_ch]
|.: 0x08048626      ff00      inc dword [eax]
|`=< 0x08048628      eba3      jmp 0x80485cd
|.; CODE XREF from fcn.080485b9 @ 0x80485db

```

```
\     `--> 0x0804862a      e8f5feffff    call fcn.08048524
```

Уже знакомы со структурой этого кода из предыдущих задач (функция проверки). Отсутствие информации о символах не создает нам больших трудностей. При желании можно использовать команду `a fn` для переименования имен функций.

```
int32_t fcn_080485b9 (char * s, void* envp)
{
    var_ch = 0;
    var_8h = 0;
    for (var_ch = 0; var_ch < strlen(s); ++var_ch)
    {
        var_dh = s[var_ch];
        sscanf(&var_dh, %d, &var_4h);           // read from string[var_ch], store to var_4h
        var_8h += var_4h;
        if(var_8h == 0x10)
            fcn_08048542(s, envp);
    }
    return fcn_08048524();
}
```

Большая часть crackme 0x07 схожа с 0x06. Задачу можно решить с помощью того же пароля и переменных среды:

```
$ export LOLAA=help
$ ./crackme0x07
IOLI Crackme Level 0x07
Password: 12346
Password OK!
```

Но ... где находится `wtf?`. Обычное дело в реверс-инжениринге искать перекрестные ссылки (xref) на строки (данные, функции и т.д.). Соответствующие команды в Radare2 находятся в группе имен ``ax``:

```
[0x08048400]> f
0x080487a8 5 str.LOL0
0x080487ad 21 str.Password_Incorrect
0x080487c5 14 str.Password_OK
0x080487d3 6 str.wtf
0x080487d9 25 str.IOLI_Crackme_Level_0x07
0x080487f2 11 str.Password:
[0x08048400]> axt 0x080487d3
(nofunc) 0x804865c [DATA] mov dword [esp], str.wtf
[0x08048400]> axF str.wtf
Ищем ссылки на флаги, соответствующие 'str.wtf'...
[0x001eff28-0x001f0000] (nofunc) 0x804865c [DATA] mov dword [esp], str.wtf
Macro 'findstref' removed.
```

[DATA] `mov dword [esp], str.wtf` по адресу `0x804865c` - инструкция `fcn.080485b9`. Но анализ на ПК игнорирует оставшиеся инструкции и отображает только неполный текст дизассемблирования. Диапазон адресов `fcn.080485b9` должен быть `0x080485b9 ~ 0x0804867c`. Сбросим размер блока и распечатаем текст вместе с опкодом.

```
[0x08040000]> s 0x080485b9
[0x080485b9]> b 230
[0x08048400]> pd
...
0x0804862f    8b450c      mov eax, dword [ebp + 0xc]
0x08048632    89442404    mov dword [esp + 4], eax
0x08048636    8b45fc      mov eax, dword [ebp - 4]
0x08048639    890424    mov dword [esp], eax      ; char **s1
0x0804863c    e873feffff  call fcn.080484b4
0x08048641    85c0        test eax, eax
,=< 0x08048643    7436        je 0x804867b
| 0x08048645    c745f4000000. mov dword [ebp - 0xc], 0
| ; CODE XREF from fcn.080485b9 @ +0xc0
.--> 0x0804864c    837df409    cmp dword [ebp - 0xc], 9
,==< 0x08048650    7f29        jg 0x804867b
| :| 0x08048652    8b45fc    mov eax, dword [ebp - 4]
| :| 0x08048655    83e001    and eax, 1
| :| 0x08048658    85c0        test eax, eax
,==< 0x0804865a    7518        jne 0x8048674
||:| 0x0804865c    c70424d38704. mov dword [esp], str.wtf      ; [0x80487d3:4]=0x3f667477 ; "wtf?\n" ; const
||:| 0x08048663    e850fdffff  call sym.imp.printf      ; int printf(const char *format)
||:| 0x08048668    c70424000000. mov dword [esp], 0      ; int status
||:| 0x0804866f    e874fdffff  call sym.imp.exit      ; void exit(int status)
```

```
|.:| ; CODE XREF from fcn.080485b9 @ +0xa1
`--> 0x08048674    8d45f4      lea eax, [ebp - 0xc]
|.:| 0x08048677    ff00      inc dword [eax]
`--< 0x08048679    ebd1      jmp 0x804864c
|.:| ; CODE XREFS from fcn.080485b9 @ +0x8a, +0x97
`-> 0x0804867b    c9        leave
          0x0804867c    c3        ret
```

`test eax, ea; je 0x804867b` перейдет к инструкциям `leave; ret`, что пропускает часть str.wtf кода. Только использование `aa` при анализе этого двоичного файла может отобразить всю функцию.

IOLI 0x08

Взломав код задачи, обнаруживаем, что задача похожа на 0x07, используется тот же пароль:

```
$ export LOLAA=help  
$ ./cracke0x08  
IOLI Crackme Level 0x08  
Password: 12346  
Password OK!
```

dustri показал лучший способ анализа crackme0x08. 0x07 --- это урезанная версия 0x08.

```
$ radiff2 -A -C ./crackme0x07 ./crackme0x08
```

...							
fcn.08048360	23	0x8048360		MATCH	(1.000000)		0x8048360
sym.imp._libc_start_main	6	0x8048388		MATCH	(1.000000)		0x8048388
sym.imp.scanf	6	0x8048398		MATCH	(1.000000)		0x8048398
sym.imp.strlen	6	0x80483a8		MATCH	(1.000000)		0x80483a8
sym.imp.printf	6	0x80483b8		MATCH	(1.000000)		0x80483b8
sym.imp.sscanf	6	0x80483c8		MATCH	(1.000000)		0x80483c8
sym.imp.strncmp	6	0x80483d8		MATCH	(1.000000)		0x80483d8
sym.imp.exit	6	0x80483e8		MATCH	(1.000000)		0x80483e8
entry0	33	0x8048400		MATCH	(1.000000)		0x8048400
fcn.08048424	33	0x8048424		MATCH	(1.000000)		0x8048424
fcn.08048450	47	0x8048450		MATCH	(1.000000)		0x8048450
fcn.08048480	50	0x8048480		MATCH	(1.000000)		0x8048480
fcn.080484b4	112	0x80484b4		MATCH	(1.000000)		0x80484b4
fcn.08048524	30	0x8048524		MATCH	(1.000000)		0x8048524
fcn.08048542	119	0x8048542		MATCH	(1.000000)		0x8048542
fcn.080485b9	118	0x80485b9		MATCH	(1.000000)		0x80485b9
main	92	0x804867d		MATCH	(1.000000)		0x804867d
fcn.08048755	4	0x8048755		MATCH	(1.000000)		0x8048755
fcn.08048760	35	0x8048760		MATCH	(1.000000)		0x8048760
fcn.0804878d	17	0x804878d		NEW	(0.000000)		
sym._libc_csu_init	99	0x80486e0		NEW	(0.000000)		
sym._libc_csu_fini	5	0x8048750		NEW	(0.000000)		
sym._fini	26	0x8048784		NEW	(0.000000)		

IOLI 0x09

Подсказка: crackme0x09 скрывает строку формата (%d и %s), и она ничем не отличается от 0x08.

```
$ export LOLA=help  
$ ./crackme0x09  
IOLI Crackme Level 0x09  
Password: 12346  
Password OK!
```

Avatao R3v3rs3 4

После нескольких лет отсутствия военных игр (wargames) в Hacktivity, в этом году наконец нашлось время начать и почти закончить (да, я очень смущен этим незаконченным webhack-ом :)) один из них. В конференции было три разные игры, я выбрал предоставленный avatao. Было восемь испытаний, большинство из которых представляют собой базовые элементы веб-хакерства, один - получение привилегированных прав и бегство из песочницы, одно переполнение буфера с

последующим его использованием по назначению, также было два упражнения по реверс-инженериングу. Задачи находятся по адресу <https://platform.avatao.com>.

.radare2

Решим задачу обратной инженерии, используя radar2, свободный инструмент реверс-инженериинга с открытым исходным кодом. Я впервые узнал о r2 еще в 2011 году, когда участвовал в большом проекте, где пришлось взламывать огромный статический ELF размером 11 МБ. Нужен был инструмент, позволяющий легко вносить исправления в Linux ELF. Тогда я использовал r2 вместе с IDA и то только для небольших задач, мне с первого взгляда понравилась вся концепция. С тех пор radare2 сильно развился, и я решил уделить немного времени решению задач crackme при помощи этого инструмента, также описать результат в виде статьи. Что ж, этот CTF дал мне прекрасную возможность :).

Статья нацелена на то, чтобы показать некоторые особенности r2 в процессе решения задачи crackme. Я объясню каждую использованную команду r2 в виде цитат, подобных этой:

Совет от r2: Всегда используй ? или флаг -h - получишь больше информации!

Если владеете r2 или просто интересуетесь задачками на взлом, пропустите эти разделы! Обращаю внимание, что из-за стиля дальнейшего изложения я собираюсь детально разбирать приемы, которые обычно не делаются во время CTF, так как нет на это достаточно времени (например, пометить **каждую** область памяти в соответствии с ее предназначением). С небольшими исполняемыми файлами crackme можно получить результат, не вдаваясь в такие детали.

Дам несколько советов в изучении radare2, и, раз вы интересуетесь реверс-инженериングом, вас как минимум должен заинтересовать r2 :):

Пакет включает множество дополнительных инструментов и огромное количество функций. Все они очень хорошо документированы. Призываю вас ознакомиться с руководством и использовать встроенную систему справки, добавляя ``?'' в конец каждой команды!

Примеры:

```
[0x00000000]> ?
Usage: [...] [cmd] [~grep] [@[@iter]addr!size][|>pipe] ; ...
Append '?' to any char command to get detailed help
Prefix with number to repeat command N times (f.ex: 3x)
%var =valueAlias for 'env' command
| *off[=[0x]value]      Pointer read/write data/values (see ?v, wx, wv)
| (macro arg0 arg1)    Manage scripting macros
| .[-|(m)|f|!sh|cmd]  Define macro or load r2, cparse or rlang file
| = [cmd]               Run this command via rap://
| /                   Search for bytes, regexps, patterns, ..
| ! [cmd]              Run given command as in system(3)
| # [algo] [len]       Calculate hash checksum of current block
| #!lang [...]         Hashbang to run an rlang script
| a                   Perform analysis of code
| b                   Get or change block size
...
[0x00000000]> a?
Usage: a[abcdefGhoprxstc] [...]
| ab [hexpairs]      analyze bytes
| aa                 analyze all (fcns + bbs) (aa0 to avoid sub renaming)
| ac [cycles]        analyze which op could be executed in [cycles]
| ad                 analyze data trampoline (wip)
| ad [from] [to]     analyze data pointers to (from-to)
| ae [expr]          analyze opcode eval expression (see ao)
| af[rnbcsl?+-*]    analyze Functions
| aF                 same as above, but using anal.depth=1
...

```

Проект находится в стадии активной разработки - нет и дня без фиксации в репозитории GitHub. Как говорится в файле readme, всегда нужно использовать версию из git!

Некоторые рекомендуемые материалы для чтения:

- Шпаргалка от pwntester-a
- Книга Radare2
- Блог Radare2
- Radare2 Wiki

.first_steps

Хорошо, хватит хвалить r2, давайте начнем взламывать (ревер-инженеринг) программы.

Во-первых, вы должны знать своего врага:

```
[0x00 avatao]$ rabin2 -I reverse4
pic      false
canary   true
nx       true
crypto   false
va       true
intrp   /lib64/ld-linux-x86-64.so.2
bintype  elf
class    ELF64
lang     c
arch     x86
bits     64
machine AMD x86-64 architecture
os       linux
subsys   linux
 endian   little
stripped true
static   false
linenum  false
lsyms   false
relocs   false
rpath   NONE
binsz   8620
```

Совет от r2: rabin2 --- это удобный инструмент, поставляемый вместе с radare2. Он используется для извлечения информации (импорты, символы, библиотеки и т. д.) из бинарных исполняемых файлов. Как всегда, пользуйтесь справкой (rabin2 -h)!

Итак, имеем динамически связанный, 64-битный исполняемый файл Linux с вырезанной информацией о символах (stripped) - ничего особенного. Попробуем запустить его:

```
[0x00 avatao]$ ./reverse4
?
Size of data: 2623
pamparam
Wrong!

[0x00 avatao]$ "\x01\x00\x00\x00" | ./reverse4
Size of data: 1
```

Хорошо, программа сначала считывает число в качестве размера со стандартного ввода, затем читает далее, вероятно, ``размер'' байтов/символов, обрабатывает этот ввод и выводит либо ``Wrong!'', либо ничего, либо еще что-то, предположительно наш флаг. Не стоит тратить больше времени на дебильный подбор данных для ввода исполняемого файла, запустим r2, потому что in asm we trust!

```
[0x00 avatao]$ r2 -A reverse4
-- Heisenbug: A bug that disappears or alters its behavior when one attempts to probe or isolate it.
(-- Heisenbug - ошибка, которая исчезает или меняет свое поведение при попытке исследовать или изолировать ее (синдром))
```

Совет от r2: Флаг командной строки -A запускает команду *aaa* при входе в r2, он анализирует весь код на наличие перекрестных ссылок. Вследствие этого мы получим имена функций, строк, информацию о перекрестных ссылках (XREFS) и т.д. Как всегда, нужна помощь - используйте ?.

Хорошей практикой является создание проекта для сохранения и восстановления состояния при повторном входе в r2:

```
[0x00400720]> Ps avatao_reverse4
avatao_reverse4
[0x00400720]>
```

Совет от r2: Сохранение проекта - команда Ps [файл], загрузка - Po [файл]. При помощи флага -p можно загрузить проект при запуске r2.

Посмотрим все найденные строки при помощи r2:

```
[0x00400720]> fs strings
[0x00400720]> f
0x00400e98 7 str.Wrong_
0x00400e9f 27 str.We_are_in_the_outer_space_
0x00400f80 18 str.Size_of_data:_u_n
0x00400f92 23 str.Such_VM_MuCH_reV3rse_
0x00400fa9 16 str.Use_everything_
0x00400fb9 9 str.flag.txt
0x00400fc7 26 str.You_won_The_flag_is:_s_n
```

```
[0x00400fe1 21 str.Your_getting_closer_
[0x00400720]>
```

Совет от r2: r2 ставит флаги на важные/интересные адреса смещений, и организует их в пространства флагов (строки, функции, символы и т.д.). Можно посмотреть список всех пространств, используя *fs*, подключиться - *fs [пространство]* (по умолчанию - *, что означает все пространства флагов). Команда *f* покажет все флаги из выбранного пространства.

Итак, строки выглядят интересно, особенно 0x00400f92. Она намекает, что этот crackme основан на виртуальной машине. Будем иметь это в виду!

Строки служат хорошей отправной точкой, если иметь дело с реальным приложением со множеством разных функций. Но у нас crackme, они обычно маленькие и простые, сосредоточены на решении простых задач. Поэтому просто смотрим на точку или точки входа, можно ли понять что-то оттуда. А сейчас все-таки посмотрим, где эти строки используются:

```
[0x00400720]> axt @@=`f~[0]`
d 0x400cb5 mov edi, str.Size_of_data:_u_n
d 0x400d1d mov esi, str.Such_VM__MuCH_reV3rse_
d 0x400d4d mov edi, str.Use_everything_
d 0x400d85 mov edi, str.flag.txt
d 0x400db4 mov edi, str.You_won__The_flag_is:_s_n
d 0x400dd2 mov edi, str.Your_getting_closer_
```

Совет от r2: можно посмотреть список перекрестных ссылок на адреса, используя команду *axt [адрес]*. Также можно использовать *axf* перечислить ссылки с заданного адреса). Последовательность символов *@@@* задает итератор, он запускает команду, подставляя в качестве параметра каждый элемент списка.

Список аргументов в данном примере берется из команды *f~/0/*. Она порождает список строк из исполняемого файла *f*, при этом фильтрует встроенным гтер-ом ~ только первые столбцы (*/0/*), содержащие адреса строк.

.main

Как было уже не раз сказано, надо сперва смотреть точку входа, так что с этого и начнем:

```
[0x00400720]> s main
[0x00400c63]>
```

Совет от r2: Переход к любому смещению, флагу, выражению и т.д. в исполняемом файле выполняется командой *s* (seek). Можно использовать ссылки, например, §§ (текущее смещение), отменить предыдущее перемещение (*s-*) или заново его повторить (*s+*). Команда позволяет искать строки (*s/ [строка]*) или шестнадцатеричные значения (*s/x 4142*), а также делать много других полезных вещей. Имеет смысл почтить инструкции? - *s?*!

Итак, мы находимся в начале функции main, можно, конечно, использовать команду *p* и ее дизассемблирующие разновидности *pd* и *pdf*, но в r2 можно зажигать по полной: у него есть визуальный режим, отображающий графы управления, похожие на графы IDA, но намного круче, так как они изображены в ASCII :)

Совет от r2: Команды группы *p* используется для отображения объектов. Например, команда *pd* дизассемблирует код с текущей позиции, дизассемблирование текущей функции - *pdf*, *ps* печатает строки, *px* - шестнадцатеричные дампы, *rbe* и *rbd* кодируют и декодируют данные согласно base64. Можно выдавать на экран и байты как есть (*raw*) - *pr*, можно дампы из одних файлов записывать в другие. Есть еще много других функций, см. инструкцию - *?*!

R2 также поддерживает вид миникарты, невероятно полезной для получения общего представления о функции:

Совет от r2: Команда *V* переводит в так называемый визуальный режим, имеющий несколько видов отображения информации. Переключать между ними можно, используя *p* и *P*. Граф отображается, если еще раз нажать *V* в визуальном режиме, либо используя *VV* будучи еще в командной строке.

Нажатие *p* в режиме графа управления переведет систему в режим отображения миникарты. Она отображает базовые функциональные блоки и связи между ними, дизассемблированный код выбранного блока, помеченного @@@@@ на миникарте, отображается тут же рядом. Можно выбирать следующий или предыдущий блок с помощью клавиш *<ТАВ> * и *<SHIFT><ТАВ> *, соответственно. Переход по ветвям графа между истинными и ложными альтернативами очевидно, - *t* и *f*.

В визуальном режиме можно открыть командную строку с помощью лавиши :, клавиша *o* - установка смещения.

Походим по графу - узел за узлом! Первый блок выглядит так:

Видно, что программа считывает слово (два байта) в локальную переменную с именем *local_10_6*, и затем сравнивает ее с 0xbb8. Это 3000 в десятичном виде:

```
[0x00400c63]> ? 0xbb8
3000 0xbb8 05670 2.9K 0000:0bb8 3000 10111000 3000.0 0.000000f 0.000000
```

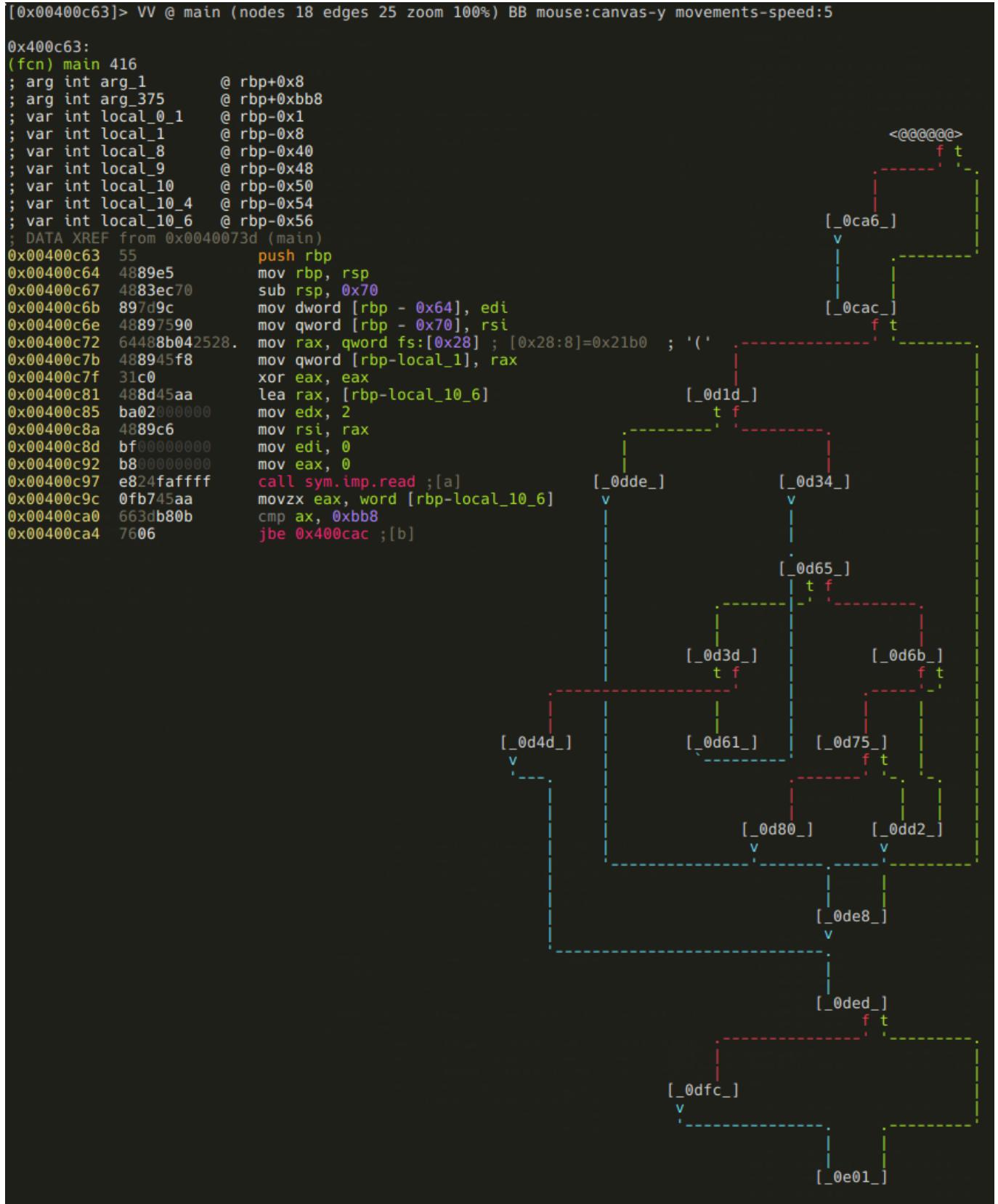


Рис. 25: миникарта функции main

```

[0x00400c63]> VV @ main (nodes 18 edges 25 zoom 100%) BB mouse:canvas-y movements-speed:5

0x400c63:
(fcn) main 416
; arg int arg_1          @ rbp+0x8
; arg int arg_375        @ rbp+0xbb8
; var int local_0_1       @ rbp-0x1
; var int local_1         @ rbp-0x8
; var int local_8         @ rbp-0x40
; var int local_9         @ rbp-0x48
; var int local_10        @ rbp-0x50
; var int local_10_4      @ rbp-0x54
; var int size             @ rbp-0x56
; DATA XREF from 0x0040073d (main)
push rbp
mov rbp, rsp
sub rsp, 0x70
mov dword [rbp - 0x64], edi
mov qword [rbp - 0x70], rsi
mov rax, qword fs:[0x28] ; [0x28:8]=0x21b0 ; '('
mov qword [rbp-local_1], rax
xor eax, eax
lea rax, [rbp-size]
mov edx, 2
mov rsi, rax
mov edi, 0
mov eax, 0
call sym.imp.read ;[a]
movzx eax, word [rbp-size]
cmp ax, 0xbb8
jbe 0x400cac ;[b]

```

Рис. 26: main bb-0c63

Совет от r2: Команда ? вычисляет выражения и печатает результат в различных форматах.

Если значение больше 3000, то оно будет заменено на 3000 принудительно:

В следующем блоке есть несколько интересных моментов:

Печатается сообщение ``Size of data:'', которое выдается при запуске программы. Итак, теперь мы знаем, что локальная переменная *local_10_6* - это размер данных, подаваемых на вход. Назовем переменную подходящим именем. Для этого открываем командную строку r2, находясь в визуальном режиме, с помощью клавиши : и вводим туда команду:

```
:> afvn local_10_6 input_size
```

Совет от r2: Комбинация *af* - семейство команд, используемых в процессе анализа функций. Включает также модификацию аргументов и локальных переменных функции. Эти возможности находятся в разделе *afv*. Можете перечислить аргументы функции (*afa*), локальные переменные (*afv*), даже переименовать их (*afan*, *afvn*). Существует множество других функций - как обычно: ИСПОЛЬЗУЙ ``?'', ЛЮК!

После этого в оперативной памяти выделяется кусок длиной *input_size* байтов и заполняется данными со стандартного ввода. Адрес этого фрагмента памяти хранится в *local_10*, снова используем *afvn*:

```
:> afvn local_10 input_data
```

Почти закончили с этим блоком, осталось всего две вещи. Блок памяти размером 512 (0x200) байт обнуляется по смещению 0x00602120. Беглый взгляд на XREFS на этот адрес показывает, что эта память действительно используется где-то в приложении:

```
:> axt 0x00602120
d 0x400cf8 mov edi, 0x602120
d 0x400d22 mov edi, 0x602120
d 0x400dde mov edi, 0x602120
d 0x400a51 mov qword [rbp - 8], 0x602120
```

Позже это будет важно, поэтому обозначим этот блок памяти:

```
:> f sym.memory 0x200 0x602120
```

```
[0x00400c63]> VV @ main (nodes 18 edges 25 zoom 100%) BB mouse:canvas-y movements-speed:5
0x400ca6:
mov word [rbp-local_10_6], 0xbb8 ; [0xbb8:2]=0x45c7
```



Рис. 27: main bb-0ca6

```
[0x00400c63]> VV @ main (nodes 18 edges 25 zoom 100%) BB mouse:canvas-y movements-speed:5
0x400cac:
movzx eax, word [rbp-local_10_6]
movzx eax, ax
mov esi, eax
mov edi, str.Size_of_data:_u_n ; "Size of data: %u." @ 0x400f80
mov eax, 0
call sym.imp.printf ;[c]
movzx eax, word [rbp-local_10_6]
movzx eax, ax
mov rdi, rax
call sym.imp.malloc ;[d]
mov qword [rbp-local_10], rax
movzx eax, word [rbp-local_10_6]
movzx edx, ax
mov rax, qword [rbp-local_10]
mov rsi, rax
mov edi, 0
mov eax, 0
call sym.imp.read ;[a]
mov edx, 0x200 ; "R.td." @ 0x200
mov esi, 0
mov edi, 0x602120 ; "ela.dyn" 0x00602120 ; "ela.dyn" @ 0x602120
call sym.imp.memset ;[e]
mov rax, qword [rbp-local_10]
mov rdi, rax
call fcn.00400a45 ;[f]
cmp eax, 0x2a ; '*'
jne 0x400de8 ;[g]
```

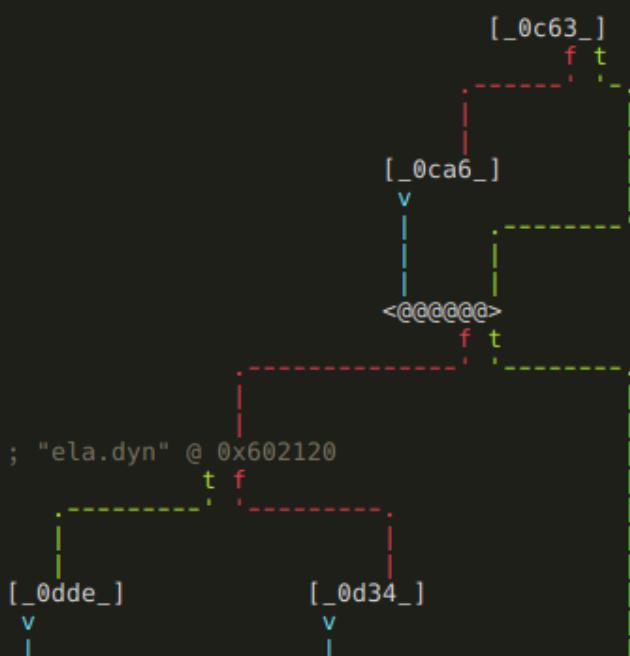


Рис. 28: main bb-0cac

Совет от r2: Флагами (идентификаторами сущностей) можно управлять с помощью команд, начинающихся с *f*. Мы добавили флаг *sym.memory*, обозначающий памяти длиной 0x200 байт по адресу 0x602120. Можно также и удалять флаги (*f-name*), переименовывать их (*fr [текущее имя] [новое имя]*), добавлять комментарии(*fC [имя] [сmt]*), и даже раскрашивать цветом (*fc [имя] [цвет]*).

Пока мы не перешли еще куда-то, нужно объявить этот фрагмент памяти блоком данных, тогда он будет отображаться как шестнадцатеричный дамп при дизассемблировании:

```
:> Cd 0x200 @ sym.memory
```

Совет от r2: Команды группы *C* используются для управления метаданными. Можно задавать комментарии (*CC*) или редактировать их (*CC*), обозначать адреса в памяти блоками данных (*Cd*) или строками (*Cs*) и т.п. Эти команды также выполняются при помощи меню в визуальном режиме, меню вызывается нажатием *d*.

Единственное, что осталось в этом блоке - вызов функции по адресу 0x400a45 с входными данными в качестве ее аргумента. Возвращаемое значение функции сравнивается с ``*'', осуществляется условный переход в зависимости от результата.

Теперь подтверждается предположения, сделанное в самом начале: этот *crackme* основан на виртуальной машине. Учитывая эту информацию, предположим, что эта функция - основной цикл виртуальной машины, а входные данные --- это инструкции, которые будет выполнять виртуальная машина. Оформим догадку, назовем функцию *vmloop*, а ее входные данные переименуем с *input_data* в *bytecode*, а *input_size* в *bytecode_length*. В этом нет крайней необходимости в таком небольшом проекте, но рекомендуется называть сущности в соответствии с их назначением, аналогично тому, как пишется программа.

```
:> af vmloop 0x400a45
:> afvn input_size bytecode_length
:> afvn input_data bytecode
```

Совет от r2: Команда *af* используется для анализа функции, заданной именем, по указанному адресу. Две другие команды уже использовались ранее.

После переименования локальных переменных, пометки области памяти и переименования функции виртуальной машины дизассемблированный код выглядит так:

```
[0x00400c63]> VV @ main (nodes 18 edges 25 zoom 100%) BB mouse:canvas-y movements-speed:5
0x400cac:
movzx eax, word [rbp-bytecode_length]
movzx eax, ax
mov esi, eax
mov edi, str.Size_of_data:_u_n ; "Size of data: %u." @ 0x400f80
mov eax, 0
call sym.imp.printf ;[c]
movzx eax, word [rbp-bytecode_length]
movzx eax, ax
mov rdi, rax
call sym.imp.malloc ;[d]
mov qword [rbp-bytecode], rax
movzx eax, word [rbp-bytecode_length]
movzx edx, ax
mov rax, qword [rbp-bytecode]
mov rsi, rax
mov edi, 0
mov eax, 0
call sym.imp.read ;[a]
mov edx, 0x200 ; "R.td." @ 0x200
mov esi, 0
mov edi, sym.memory ; "ela.dyn" @ 0x602120
call sym.imp.memset ;[e]
mov rax, qword [rbp-bytecode]
mov rdi, rax
call fcn.vmloop ;[f]
cmp eax, 0x2a ; '*'
jne 0x400de8 ;[g]
```

Рис. 29: main bb-0cac_meta

Вернемся к условному переходу. Если *vmloop* возвращает что-то, кроме ``*'', программа просто завершает работу, не давая нам наш флаг. Очевидно, нам этого не надо, поэтому идем по ветке *false*.

```
[0x00400c63]> VV @ main (nodes 18 edges 25 zoom 100%) BB mouse:canvas-y movements-speed:5
0x400d1d:
mov esi, str.Such_VM_MuCH_reV3rse_ ; "Such VM! MuCH reV3rse!" @ 0x400f92 [_0cac_]
mov edi, sym.memory ; "ela.dyn" @ 0x602120
call sym.imp.strptime ;[i]
test eax, eax
jne 0x400dde ;[j]
```

Рис. 30: main bb-0d1d

Теперь видно, что строка в этой 512-байтовой области памяти *sym.memory* сравнивается с ``Such VM! MuCH reV3rse!''. Если они не равны, программа выводит байт-код и выходит:

```
[0x00400c63]> VV @ main (nodes 18 edges 25 zoom 100%) BB mouse:canvas-y movements-speed:5
0x400dde:
mov edi, sym.memory ; "ela.dyn" @ 0x602120
call sym.imp.puts ;[k]
```

Рис. 31: main bb-0dde

Итак, теперь точно знаем, что нужно предоставить на вход байт-код, который будет генерировать эту строку при исполнении. Как видно на миникарте, осталось проанализировать еще несколько ветвей, для достижения результата нужно выполнить больше условий. Исследуем их прежде чем углубляться в *vmloop*!

Посмотрев миникарту всей функции, видно, что на ней распознается какой-то цикл, начинающийся в блоке *[0d34]*, и он включает следующие узлы:

- *[0d34]*
- *[0d65]*
- *[0d3d]*
- *[0d61]*

Вот дизассемблирование этих блоков. Первый помещает 0 в локальную переменную *local_10_4*:

В блоке сравнивается *local_10_4* с числом 8, далее выполняется условный переход на основе полученного результата:

Совершенно очевидно, что *local_10_4* - это переменная цикла, поэтому назовем ее соответственно:

:> afvn local_10_4 i

Следующий блок --- это собственно тело цикла:

Область памяти по адресу 0x6020e0 обрабатывается как массив двойных слов (значения по четыре байта), и проверяется, равно ли *i*-е его значение нулю. Если это не так, цикл просто продолжает выполняться:

Если значение равно нулю, цикл прерывается, перед выходом выполняется этот блок:

Выводится сообщение: ``Use everything!''. Как установлено ранее, мы имеем дело с виртуальной машиной. В этом контексте сообщение, вероятно, означает, что нужно использовать все существующие инструкции. Выполнена ли инструкция или нет, она сохраняется по адресу 0x6020e0, поэтому давайте отметим эту область памяти флагом:

:> f sym.instr_dirty 4*9 0x6020e0

Предполагая, что не выполнен выход из цикла по break, анализируем другие ветвления и переходы:

Этот фрагмент кода кажется немного странным, если не знаком со спецификой архитектуры процессоров x86_64. В этом фрагменте речь идет об относительной RIP-адресации, где адреса задаются как смещения от адреса текущей инструкции,



Рис. 32: main bb-0d34

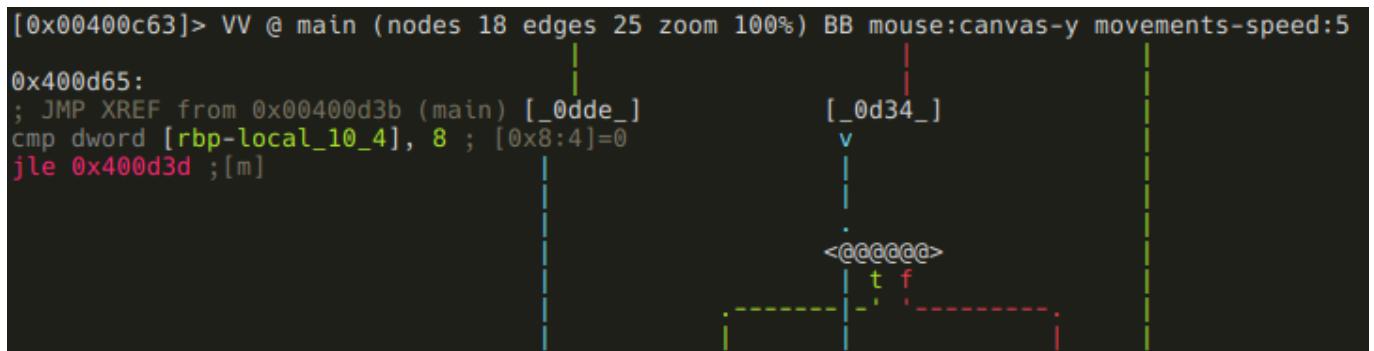


Рис. 33: main bb-0d65

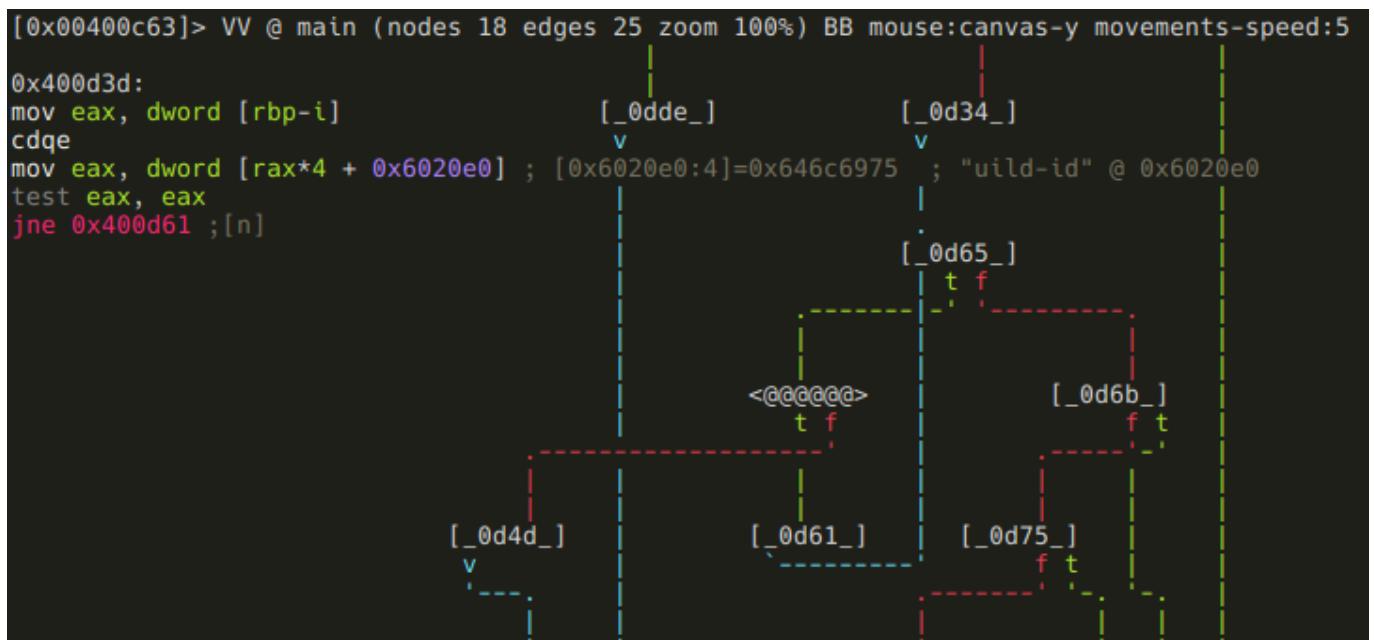


Рис. 34: main bb-0d3d

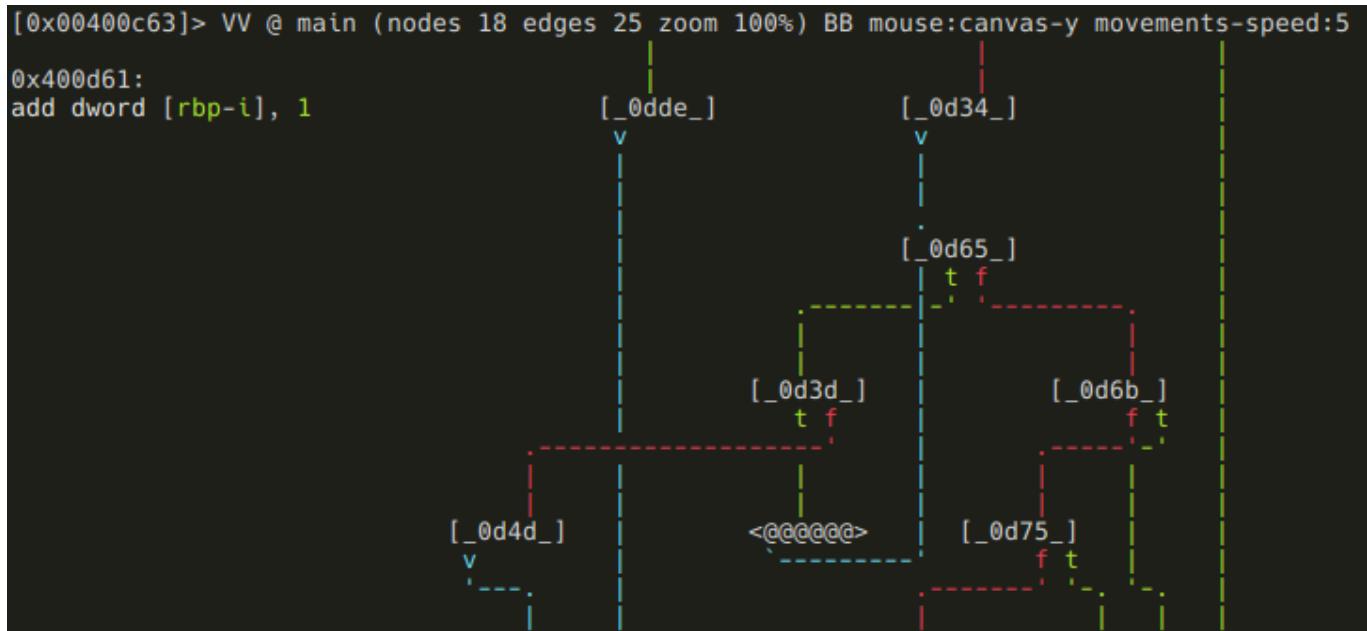


Рис. 35: main bb-0d61

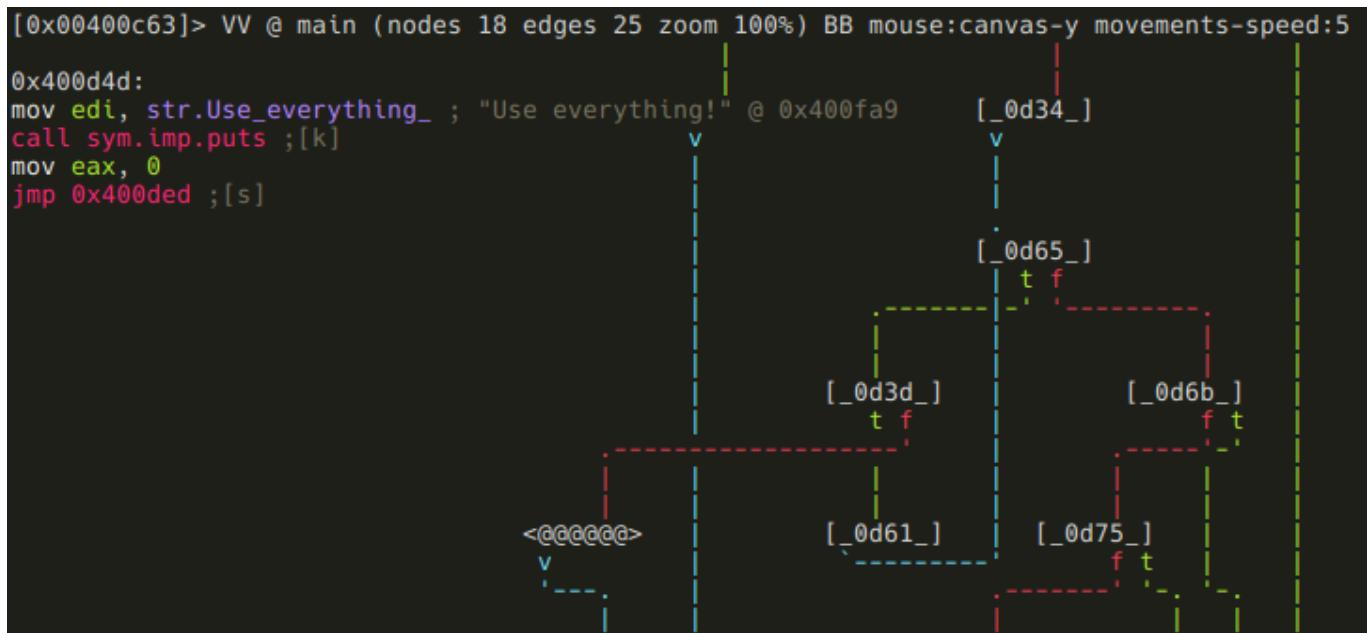


Рис. 36: main bb-0d4d



Рис. 37: main bb-0d6b

что упрощает реализацию режима PIE. В любом случае r2 достаточно умен и отображает фактический адрес (0x602104). Получил адрес, пометь его флагом!

```
:> f sym.good_if_ne_zero 4 0x602104
```

Однако, при использовании относительной адресации RIP, флаги не будут отображаться прямо в результате дизассемблирования, а r2 отобразит их в виде комментариев:

```
[0x00400c63]> VV @ main (nodes 18 edges 25 zoom 100%) BB mouse:canvas-y movements-speed:5  
0x400d6b:  
mov eax, dword [rip + 0x201393] ; [0x602104:4]=0x762e756e LEA sym.good_if_ne_zero ; "nu.v  
test eax, eax  
je 0x400dd2 ;[o]
```

Рис. 38: main bb-0d6b_meta

Если *sym.good_if_ne_zero* равен нулю, получаем сообщение ("Your getting closer!"), и затем исполнение программы останавливается. Если не нуль, то переходим к последней проверке:

```
[0x00400c63]> VV @ main (nodes 18 edges 25 zoom 100%) BB mouse:canvas-y movements-speed:5  
0x400d75:  
mov eax, dword [rip + 0x201375] ; [0x6020f0:4]=0x642e0068 ; "h" @ 0x6020f0_]  
cmp eax, 9  
jg 0x400dd2 ;[o]
```

Рис. 39: main bb-0d75

Здесь программа сравнивает двойное слово по адресу 0x6020f0 (опять же используется RIP-адресация) с 9. Если оно больше 9, получаем то же сообщение "Your getting closer!", если же оно меньше или равно 9, наконец достигаем пункта назначения и получаем флаг:

Как обычно, установим флаг для 0x6020f0:

```
:> f sym.good_if_le_9 4 0x6020f0
```

Функция *main* теперь полностью взломана. Подводя итог, декларируем, программа считывает байт-код со стандартного ввода и передает его виртуальной машине. После выполнения VM состояние программы должно удовлетворять этим ограничениям для получения подтверждения о выполнении всего задания:

- возвращаемое значение *vmloop*-а должно быть "*",
- *sym.memory* должна содержать строку "Such VM! MuCH reV3rse!",
- все девять элементов массива *sym.instr_dirty* не должны быть равны нулю, вероятно, это означает что все инструкции надо использовать хотя бы один раз,
- *sym.good_if_ne_zero* должно стать нулем,
- *sym.good_if_le_9* должно быть меньше или равно 9.

На этом завершается анализ функции *main*, теперь можно перейти к самой виртуальной машине.

.vmloop

```
[offset]> fcn.vmloop
```

Кажется наша публикация разочаровывающе коротка, но не беспокойтесь, у нас есть много чего можно взломать. Все дело в том, что эта функция использует таблицу переходов по адресу 0x00400a74,

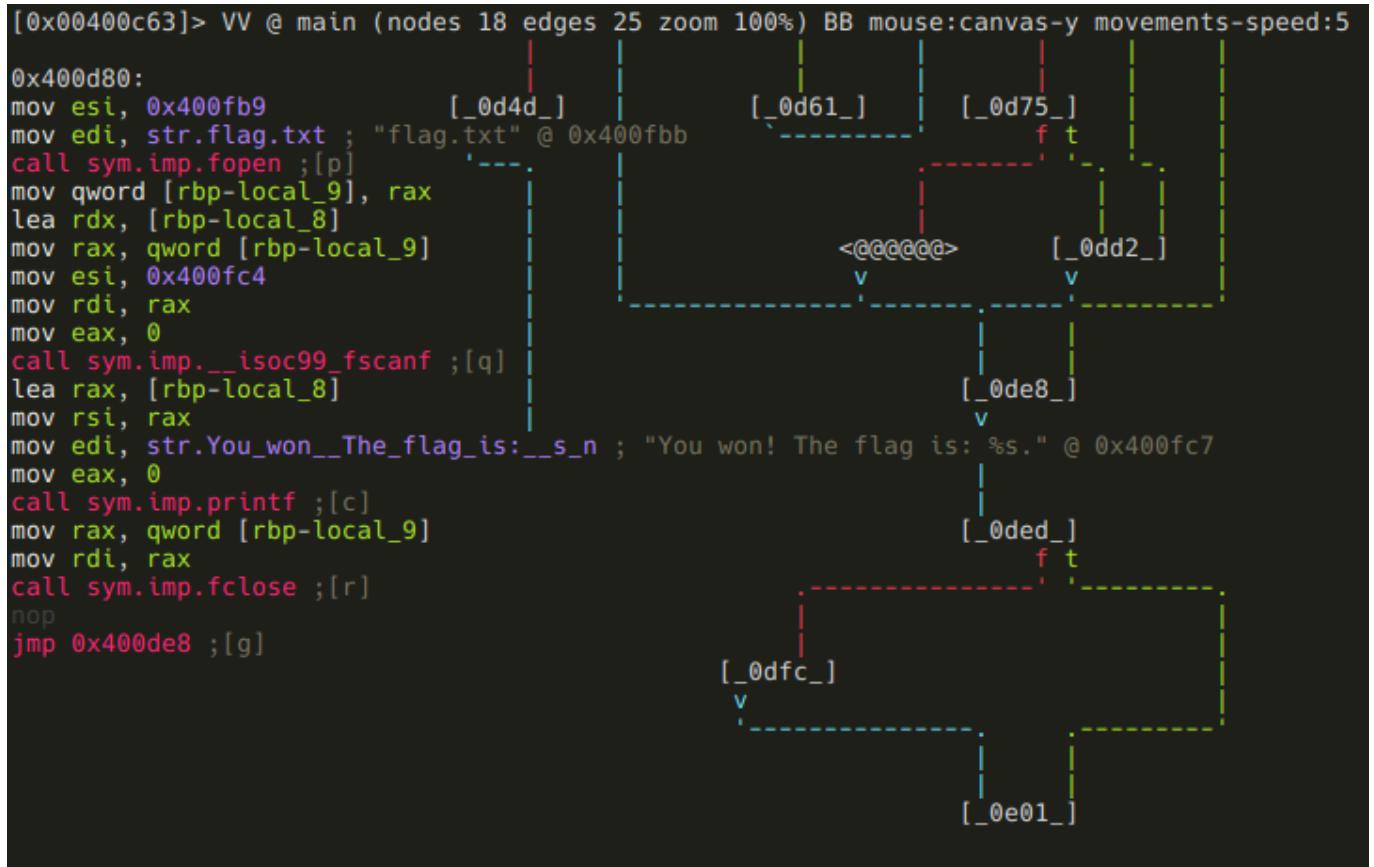


Рис. 40: main bb-0d80

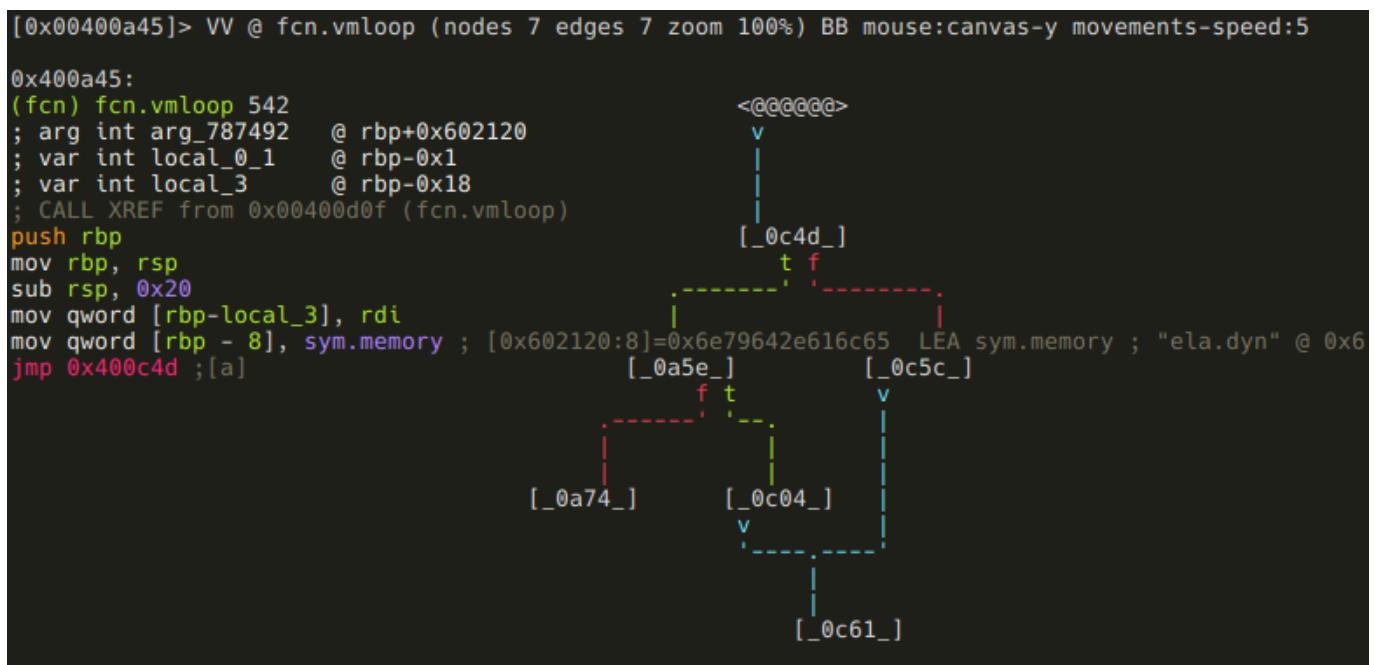


Рис. 41: vmloop bb-0a45



Рис. 42: vmloop bb-0a74

Программа r2 пока не может распознать таблицы переходов (Выпуск 3201), так что анализ этой функции неполон. Граф управления не дает полной информации, поэтому надо либо использовать визуальный режим, либо вносить исправления в блоки. Двоичный код функция всего 542 байта, можно ее взломать и без использования графа. Наша цель - продемонстрировать как можно больше возможностей r2, и имеет смысл показать, как определяются блоки в графе.

Давайте проанализируем то, что у нас уже есть! Значение *rdi* помещается в *local_3*. Поскольку приложение является 64-битным исполняемым файлом Linux, известно, что *rdi* - это первый аргумент функции (как вы, возможно, поняли, автоматический анализ аргументов и локальных переменных не был полностью корректным и полным). Также известно, что первым аргументом *vmloop* является байт-код.

Переименуем *local_3*:

```
:> afvn local_3 bytecode
```

Затем *sym.memory* помещается в другую локальную переменную по адресу *rbp-8*, также нераспознанную r2. Давайте определим ее!

```
:> afv 8 memory qword
```

Совет от r2: Команда *afv [idx] [name] [type]*</g3 используется для определения локальной переменной с указателем фрейма - *idx*, именем *[name]* и типом *[type]*. Можно также удалять локальные переменные с помощью команды *afv- [idx]**.

В следующем блоке программа проверяет один байт байт-кода, и если он равен 0, функция возвращает 1.

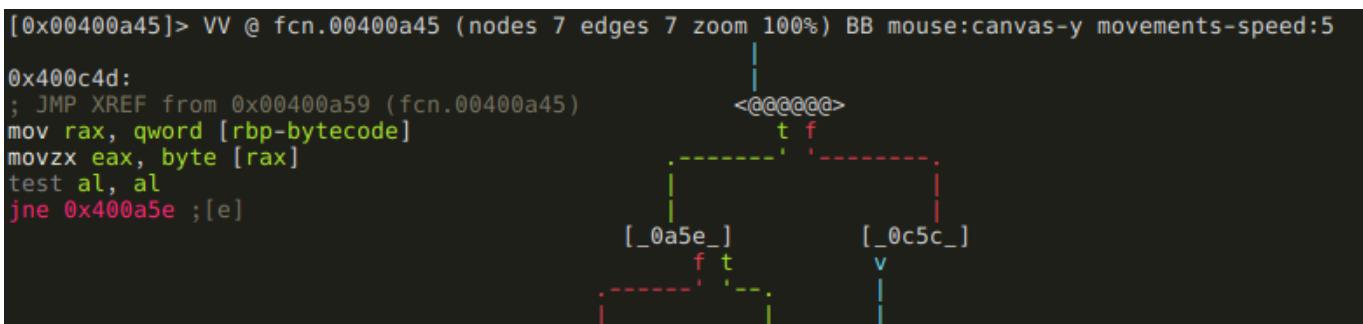


Рис. 43: vmloop bb-0c4d

Если этот байт не равен нулю, программа вычитает из него 0x41 и сравнивает результат с 0x17. Если результат больше 0x17, получаем страшное сообщение "Wrong!", и функция возвращает значение 0. Это означает, что допустимые байт-коды - это ASCII-символы в диапазоне от "A" (0x41) до "X" (0x41 + 0x17). Если байт-код действителен, приходим к фрагменту кода, реализующему таблицу переходов:

Адрес первой ячейки таблицы переходов - 0x400ec0, определим эту область памяти как набор qword-ов:

```
[0x00400a74]> s 0x00400ec0
[0x00400ec0]> Cd 8 @@=?s $$ $$+$+8*0x17 8`
```

Совет от r2: Кроме ?s все части этой команды знакомы, давайте подведем итоги! Cd определяет область памяти как данные, а 8 размер этой области памяти. @@* --- это оператор, выполняющий команду слева для

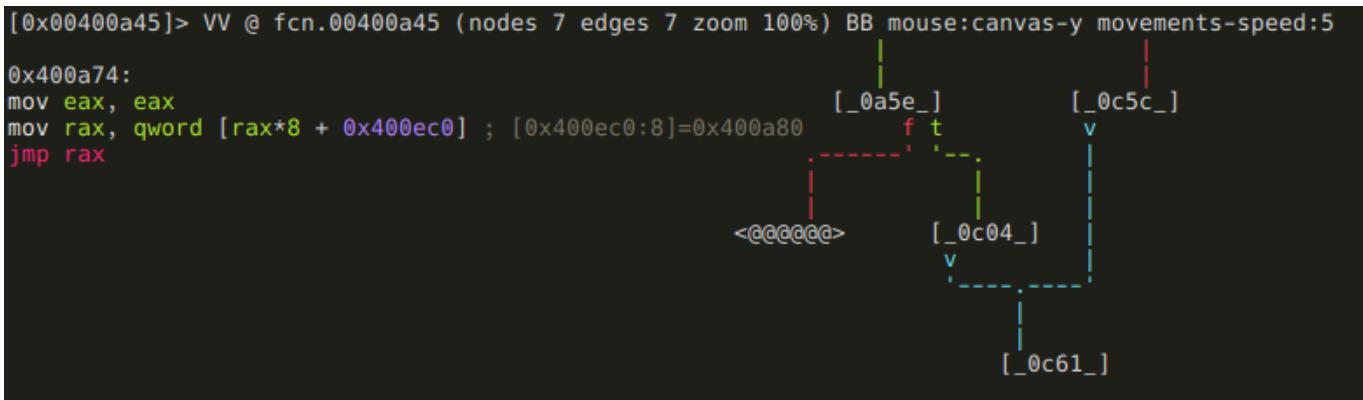


Рис. 44: vmloop bb-0a74

каждого элемента, для которого условие `@@*` истинно. В этом примере он содержит список, генерированный с помощью команды `?s`. Выражение `?s` генерирует список адресов, начиная с текущего смещения (`seek`)

`*seek + 8 * 0x17(*)`

+`80x17*`) с шагом 8.

Вот как выглядит дизассемблирование после добавления метаданных:

```
[0x00400ec0]> pd 0x18
; DATA XREF from 0x00400a76 (unk)
0x00400ec0 .qword 0x0000000000400a80
0x00400ec8 .qword 0x0000000000400c04
0x00400ed0 .qword 0x0000000000400b6d
0x00400ed8 .qword 0x0000000000400b17
0x00400ee0 .qword 0x0000000000400c04
0x00400ee8 .qword 0x0000000000400c04
0x00400ef0 .qword 0x0000000000400c04
0x00400ef8 .qword 0x0000000000400c04
0x00400f00 .qword 0x0000000000400aec
0x00400f08 .qword 0x0000000000400bc1
0x00400f10 .qword 0x0000000000400c04
0x00400f18 .qword 0x0000000000400c04
0x00400f20 .qword 0x0000000000400c04
0x00400f28 .qword 0x0000000000400c04
0x00400f30 .qword 0x0000000000400c04
0x00400f38 .qword 0x0000000000400b42
0x00400f40 .qword 0x0000000000400c04
0x00400f48 .qword 0x0000000000400be5
0x00400f50 .qword 0x0000000000400ab6
0x00400f58 .qword 0x0000000000400c04
0x00400f60 .qword 0x0000000000400c04
0x00400f68 .qword 0x0000000000400c04
0x00400f70 .qword 0x0000000000400c04
0x00400f78 .qword 0x0000000000400b99
```

Теперь видно, что адрес `0x400c04` в таблице активно используется, есть также девять других адресов. Посмотрим сначала `0x400c04`!

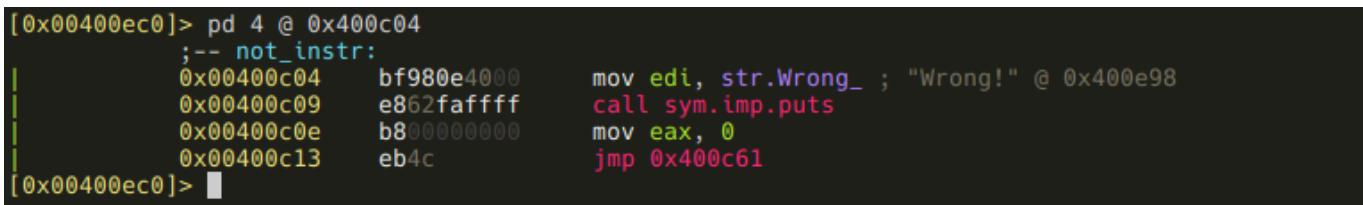


Рис. 45: vmloop bb-0c04

Выводится сообщение ``Wrong!'', и функция просто возвращает 0. Это недопустимые инструкции (они являются допустимым байт-кодами, могут быть, например, параметрами!). Отметим `0x400c04` соответствующим образом:

```
[0x00400ec0]> f not_instr @ 0x0000000000400c04
```

Что касается других адресов, они, кажется, делают что-то существенное, предположим, что адреса соответствуют действительным инструкциям. Пометим их с помощью ASCII-символов:

```
[0x00400ec0]> f instr_A @ 0x0000000000400a80
[0x00400ec0]> f instr_C @ 0x0000000000400b6d
[0x00400ec0]> f instr_D @ 0x0000000000400b17
[0x00400ec0]> f instr_I @ 0x0000000000400aec
[0x00400ec0]> f instr_J @ 0x0000000000400bc1
[0x00400ec0]> f instr_P @ 0x0000000000400b42
[0x00400ec0]> f instr_R @ 0x0000000000400be5
[0x00400ec0]> f instr_S @ 0x0000000000400ab5
[0x00400ec0]> f instr_X @ 0x0000000000400b99
```

Перечисленных адресов нет на графике, определим блоки для них!

Совет от r2: Блоки графа управления задаются с помощью команды `afb+`. Надо указать функцию, соответствующую конкретному блоку, адрес первого байта кода и его размер. Если блок заканчивается переходом (`jmp`), надо указать куда осуществляется переход. Если переход является условным, целевой адрес `false`-ветви также следует указать.

Начальный и конечный адрес каждого блока получается в результате анализа дизассемблированного кода функции `vmloop`.

Ранее мы видели, что сама функция простая, а ее код короткий, простой в изучении особенно с нашими аннотациями. Как обещано, создам недостающие блоки для всех инструкций:

```
[0x00400ec0]> afb+ 0x00400a45 0x00400a80 0x00400ab6-0x00400a80 0x400c15
[0x00400ec0]> afb+ 0x00400a45 0x00400ab6 0x00400aec-0x00400ab6 0x400c15
[0x00400ec0]> afb+ 0x00400a45 0x00400aec 0x00400b17-0x00400aec 0x400c15
[0x00400ec0]> afb+ 0x00400a45 0x00400b17 0x00400b42-0x00400b17 0x400c15
[0x00400ec0]> afb+ 0x00400a45 0x00400b42 0x00400b6d-0x00400b42 0x400c15
[0x00400ec0]> afb+ 0x00400a45 0x00400b6d 0x00400b99-0x00400b6d 0x400c15
[0x00400ec0]> afb+ 0x00400a45 0x00400b99 0x00400bc1-0x00400b99 0x400c15
[0x00400ec0]> afb+ 0x00400a45 0x00400bc1 0x00400be5-0x00400bc1 0x400c15
[0x00400ec0]> afb+ 0x00400a45 0x00400be5 0x00400c04-0x00400be5 0x400c15
```

Из дизассемблированного кода также видно, что кроме блоков инструкций есть еще три блока. Создадим их тоже!

```
[0x00400ec0]> afb+ 0x00400a45 0x00400c15 0x00400c2d-0x00400c15 0x400c3c 0x00400c2d
[0x00400ec0]> afb+ 0x00400a45 0x00400c2d 0x00400c3c-0x00400c2d 0x400c4d 0x00400c3c
[0x00400ec0]> afb+ 0x00400a45 0x00400c3c 0x00400c4d-0x00400c3c 0x400c61
```

Эти блоки начинаются с адресов `0x00400c15` и `0x00400c2d` и заканчиваются условными переходами, надо задать также адрес назначения для `false`-ветви!

И вот график управления во всей его красе после ручной реструктуризации:

Я думаю, это того стоило? :) На самом деле на реструктуризацию не стоило тратить время, так как он не сохраняется при сохранении проекта.

Совет от r2: Можно перемещать выбранный узел в представлении графика с помощью клавиш `HJKL`.

Кстати, вот как выглядит график этой же функции в IDA:

Присматриваясь к дизассемблированному коду `instr_LETTER` в виде блоков графа, распознаются несколько сущностей. Первое: все инструкции начинаются с последовательностей вида

Теперь ясно, что девять dword-ов в `sym.instr_dirty` - это не просто индикаторы того, что инструкция была выполнена, они также используются для подсчета количества вызовов инструкций. Также я должен был понять раньше, что `sym.good_if_le_9` (`0x6020f0`) является частью этого массива из девяти dword-ов ... ну да... Теперь как-то надо жить с этим... Каков смысл условия ```sym.good_if_le_9` должно быть меньше или равно 9'' на самом деле? - `instr_P` не может быть выполнен более девяти раз:

Еще одно некоторое сходство между инструкциями заключается в том, что семь из них вызывает функцию либо с одним, либо с двумя параметрами, причем параметры - это байт-коды, следующий или два последующих. Пример с одним параметром:

И пример с двумя параметрами:

Известно также, что эти блоки помещают в локальную переменную по адресу `0xc` количество байтов, которые они "отъедают" от байт-кода (1 байт инструкции + 1 или 2 байта аргументов = 2 или 3), r2 не распознал эту переменную, давайте обозначим ее вручную!

```
:> afv 0xc instr_ptr_step dword
```

Посмотрим на `instr_J` и увидим, что это исключение из приведенного выше правила, так как она помещает возвращаемое значение вызываемой функции в `instr_ptr_step` вместо константы 2 или 3:

И, продолжая изучать исключения, вот две инструкции, которые не вызывают функции:

```

(0x00400ec0)> pdf @ fcn.vmloop
(fcn) fcn.00400a45 542
; arg int arg_787492    @ rbp+0x602120
; var int local_0_1     @ rbp-0x1
; var qword memory      @ rbp-0x8
; var dword instr_ptr_step @ rbp-0xc
; var int bytecode       @ rbp-0x18
--- fcn.vmloop:
0x00400a45    55          push rbp
0x00400a46    4889e5        mov rbp, rsp
0x00400a49    4883ec20      sub rsp, 0x20
0x00400a4d    48897de8      mov qword [rbp-bytecode], rdi
0x00400a51    48c745f82021. mov qword [rbp-memory], sym.memory
0x00400a59    e9ef010000    jmp 0x400c4d
0x00400a5e    488b45e8      mov rax, qword [rbp-bytecode]
0x00400a60    0fb600        movzx eax, byte [rax]
0x00400a65    0fbec0        movsx eax, al
0x00400a68    83e841        sub eax, 0x41
0x00400a6b    83f817        cmp eax, 0x17
0x00400a6e    0fb790010000. ja 0x400c04
0x00400a74    89c0          mov eax, eax
0x00400a76    488b04c5c00e. mov rax, qword [rax*8 + 0x400ec0]
0x00400a7e    ffe0          jmp rax
--- instr_A:
0x00400a80    8b055162000. mov eax, dword [rip + 0x20165a]
0x00400a86    83c001        add eax, 1
0x00400a89    89055162000. mov dword [rip + 0x201651], eax
0x00400a8f    488b45e8      mov rax, qword [rbp-bytecode]
0x00400a93    488d5002      lea rdx, [rax + 2]
0x00400a97    488b45e8      mov rax, qword [rbp-bytecode]
0x00400a9b    4883c001      add rax, 1
0x00400a9f    4889d6        mov rsi, rdx
0x00400aa2    4889c7        mov rdi, rax
0x00400aa5    e853ffffff
0x00400aaa    c745f4030000. mov dword [rbp-instr_ptr_step], 3
0x00400aab    e95f010000    jmp 0x400c15
--- instr_S:
0x00400ab6    8b0528162000. mov eax, dword [rip + 0x201628]
0x00400abc    83c001        add eax, 1
0x00400abf    89051f162000. mov dword [rip + 0x20161f], eax
0x00400ac5    488b45e8      mov rax, qword [rbp-bytecode]
0x00400ac9    488d5002      lea rdx, [rax + 2]
0x00400acd    488b45e8      mov rax, qword [rbp-bytecode]
0x00400ad1    4883c001      add rax, 1
0x00400ad5    4889d6        mov rsi, rdx
0x00400ad8    4889c7        mov rdi, rax
0x00400adb    e8bfdf0fffff
0x00400ae0    c745f4030000. mov dword [rbp-instr_ptr_step], 3
0x00400ae7    e929010000    jmp 0x400c15
--- instr_I:
0x00400aec    8b05f6152000. mov eax, dword [rip + 0x2015f6]
0x00400af2    83c001        add eax, 1
0x00400afa    8905ed152000. mov dword [rip + 0x2015ed], eax
0x00400afb    488b45e8      mov rax, qword [rbp-bytecode]
0x00400ac5    488d5002      lea rdx, [rax + 2]
0x00400acd    488b45e8      mov rax, qword [rbp-bytecode]
0x00400ad1    4883c001      add rax, 1
0x00400ad5    4889d6        mov rsi, rdx
0x00400ad8    4889c7        mov rdi, rax
0x00400adb    e856feffff
0x00400bb0    c745f4020000. mov dword [rbp-instr_ptr_step], 2
0x00400bb2    e9fe000000    jmp 0x400c15
--- instr_D:
0x00400bb7    8b05cf152000. mov eax, dword [rip + 0x2015cf]
0x00400bbd    83c001        add eax, 1
0x00400bc0    8905c6152000. mov dword [rip + 0x2015c6], eax
0x00400bb6    488b45e8      mov rax, qword [rbp-bytecode]
0x00400bb3    4883c001      add rax, 1
0x00400bb2    4889c7        mov rdi, rax
0x00400bb1    e806feffff
0x00400bb3    c745f4020000. mov dword [rbp-instr_ptr_step], 2
0x00400bbd    e9d3000000    jmp 0x400c15
--- instr_P:
0x00400bb4    8b05a8152000. mov eax, dword [rip + 0x2015a8]
0x00400bb8    83c001        add eax, 1
0x00400bb4    89059f152000. mov dword [rip + 0x20159f], eax
0x00400bb1    488b45e8      mov rax, qword [rbp-bytecode]
0x00400bb5    4883c001      add rax, 1
0x00400bb9    4889c7        mov rdi, rax
0x00400bb5c    e825feffff
0x00400bb1    c745f4020000. mov dword [rbp-instr_ptr_step], 2
0x00400bb8    e9a8000000    jmp 0x400c15
--- instr_C:
0x00400bbd    8b0581152000. mov eax, dword [rip + 0x201581]
0x00400bb3    83c001        add eax, 1
0x00400bb4    89059f152000. mov dword [rip + 0x20159f], eax
0x00400bb7    488b45e8      mov rax, qword [rbp-bytecode]
0x00400bbc    4883c001      add rax, 1
0x00400bb6    4889c7        mov rdi, rax
0x00400bb4    0fb6          movzx eax, byte [rax]
0x00400bb7    0fbec0        movsx eax, al
0x00400bb8    890530152000. mov dword [rip + 0x201530], eax
0x00400bb9    c745f4020000. mov dword [rbp-instr_ptr_step], 2
0x00400bb97    eb7c          jmp 0x400c15
--- instr_X:
0x00400bb9    8b0559152000. mov eax, dword [rip + 0x201559]
0x00400bbf    83c001        add eax, 1
0x00400ba2    890550152000. mov dword [rip + 0x201550], eax
0x00400bb8    488b45e8      mov rax, qword [rbp-bytecode]
0x00400bac    4883c001      add rax, 1
0x00400bb0    4889c7        mov rdi, rax
0x00400bb3    e857feffff
0x00400bb8    c745f4020000. mov dword [rbp-instr_ptr_step], 2
0x00400bbf    eb54          jmp 0x400c15
--- instr_J:
0x00400bc1    8b0535152000. mov eax, dword [rip + 0x201535]
0x00400bc7    83c001        add eax, 1
0x00400bca    89052c152000. mov dword [rip + 0x20152c], eax
0x00400bb0    488b45e8      mov rax, qword [rbp-bytecode]
0x00400bd4    4883c001      add rax, 1
0x00400bd8    4889c7        mov rdi, rax
0x00400bdb    e8d8fdffff
0x00400be0    8945f4          mov dword [rbp-instr_ptr_step], eax
0x00400be3    eb30          jmp 0x400c15
--- instr_R:
0x00400be5    8b0515152000. mov eax, dword [rip + 0x201515]
0x00400beb    83c001        add eax, 1
0x00400bee    89050c152000. mov dword [rip + 0x20150c], eax
0x00400bf4    488b45e8      mov rax, qword [rbp-bytecode]
0x00400bf8    4883c001      add rax, 1
0x00400bfc    0fb600        movzx eax, byte [rax]
0x00400bbf    0fbec0        movsx eax, al
0x00400c02    eb5d          jmp 0x400c61
--- not_instr:
0x00400c04    bf980e4000    mov edi, str.Wrong_
0x00400c09    e852faffff
0x00400c0e    b800000000    call sym.imp.puts
0x00400c13    eb4c          mov eax, 0
0x00400c15    8b45f4        mov eax, dword [rbp-instr_ptr_step]
0x00400c18    4898          cdqe
0x00400c1a    480145e8      add qword [rbp-bytecode], rax
0x00400c1e    488b05631420. mov rax, qword [rip + 0x201463]
0x00400c25    483d20216000. cmp rax, sym.memory
0x00400c2b    720f          jb 0x400c3c
0x00400c2d    488b0541420. mov rax, qword [rip + 0x201454]
0x00400c34    483d20236000. cmp rax, section_end..bss
0x00400c3a    7211          jb 0x400c4d
0x00400c3c    bf9f0e4000    mov edi, str.We_are_in_the_outer_space_
0x00400c41    e82a faffff
0x00400c46    b800000000    call sym.imp.puts
0x00400c4b    eb14          mov eax, 0
0x00400c4d    488b45e8      mov rax, qword [rbp-bytecode]
0x00400c51    0fb600        movzx eax, byte [rax]
0x00400c54    84c0          test al, al
0x00400c56    0f8502feffff
0x00400c5c    b801000000    jne 0x400a5e
0x00400c61    c9            leave
0x00400c62    c3            ret

```

Рис. 46: все о vmloop
200

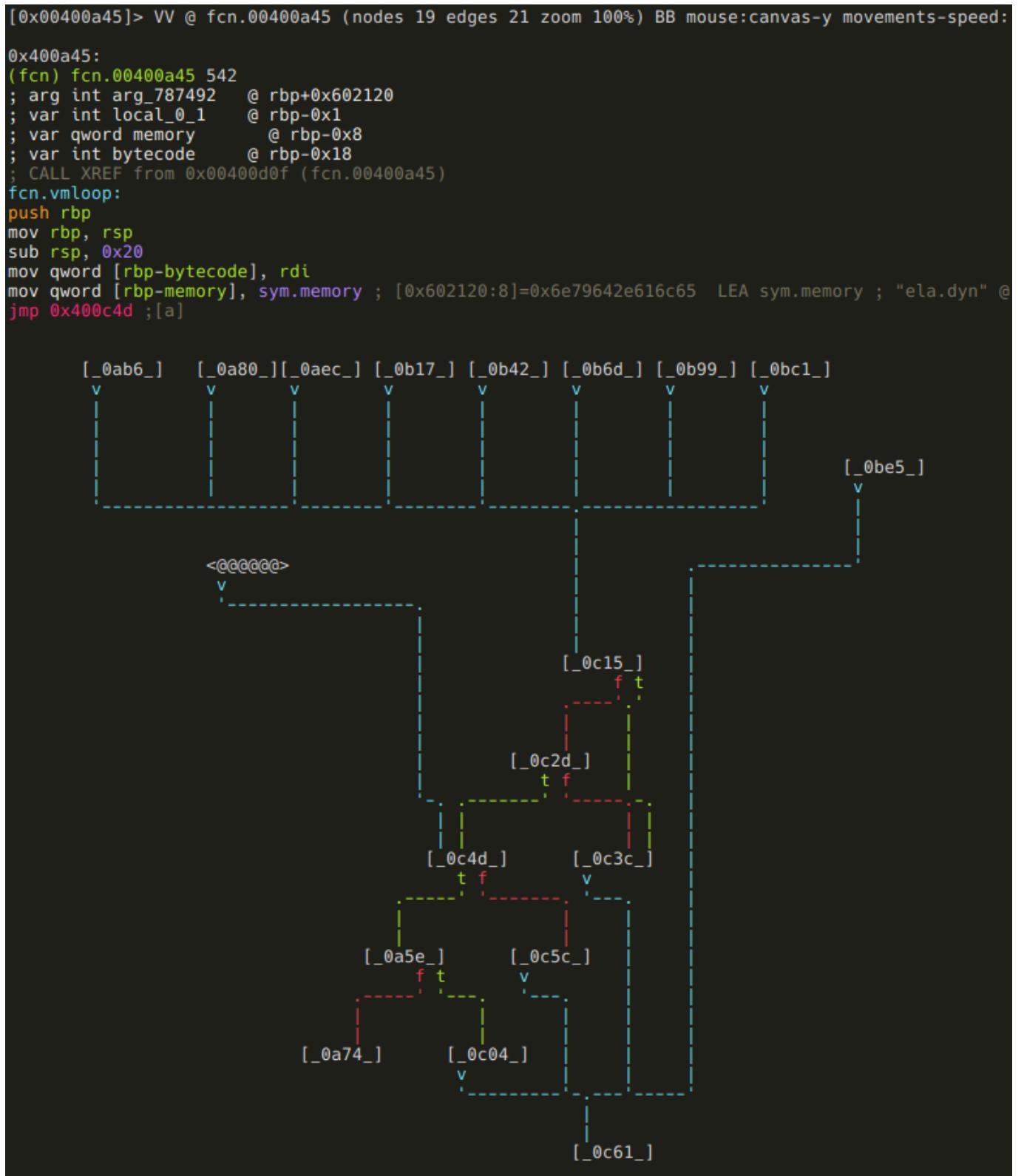


Рис. 47: граф управления vmloop

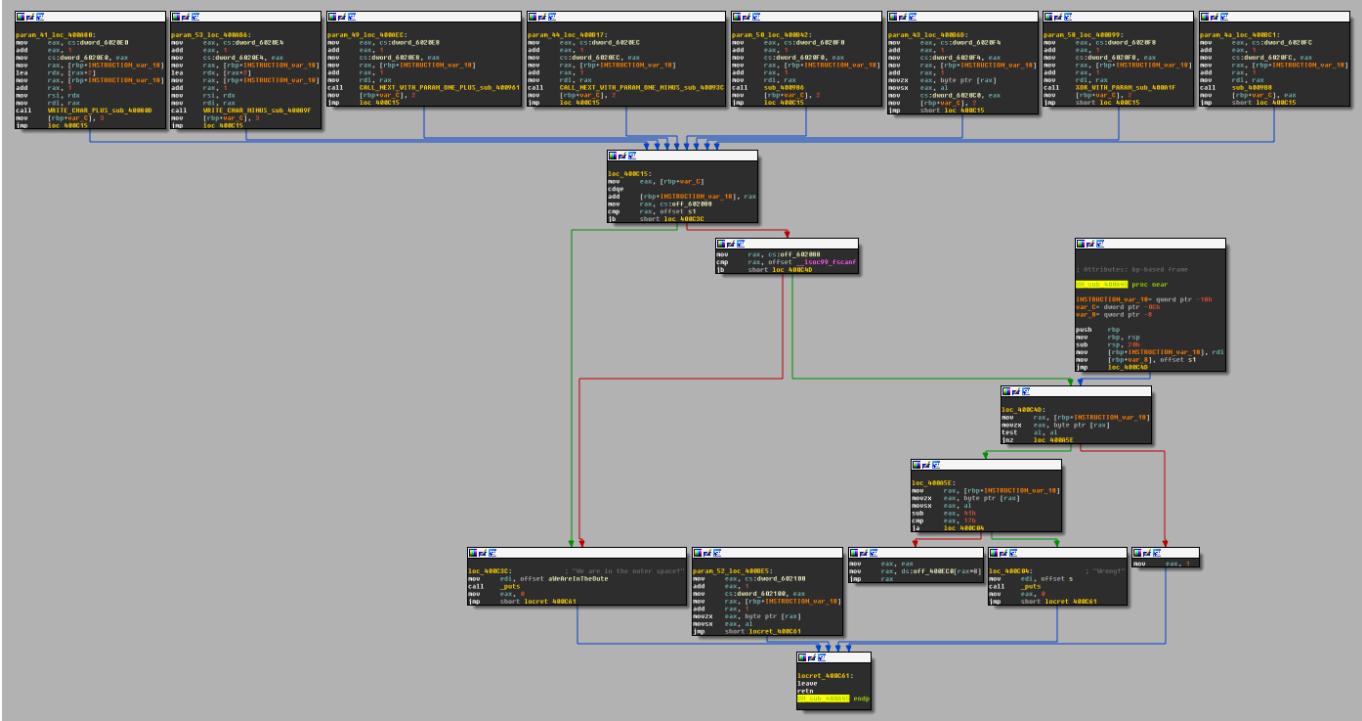


Рис. 48: Граф управления в IDA

```
0x400a80:
instr_A:
mov eax, dword [rip + 0x20165a] ; [0x6020e0:4]=0x646c6975 LEA sym.instr_dirty ; "uild-id" @ 0x6
add eax, 1
mov dword [rip + 0x201651], eax ; [0x6020e0:4]=0x646c6975 LEA sym.instr_dirty ; "uild-id" @ 0x6
```

Рис. 49: vmloop bb-0a80

```
0x400ab6:
instr_S:
mov eax, dword [rip + 0x201628] ; [0x6020e4:4]=0x64692d ; "-id" @ 0x6020e4
add eax, 1
mov dword [rip + 0x20161f], eax ; [0x6020e4:4]=0x64692d ; "-id" @ 0x6020e4
```

Рис. 50: vmloop bb-0ab6

```
0x400b42:
instr_P:
mov eax, dword [rip + 0x2015a8] ; [0x6020f0:4]=0x642e0068 LEA sym.good_if_le_9 ; "h" @ 0x6020f0
add eax, 1
mov dword [rip + 0x20159f], eax ; [0x6020f0:4]=0x642e0068 LEA sym.good_if_le_9 ; "h" @ 0x6020f0
```

Рис. 51: vmloop bb-0b42

```
0x400aec:
instr_I:
mov eax, dword [rip + 0x2015f6] ; [0x6020e8:4]=0x756e672e ; ".gnu.hash" @ 0x6020e8
add eax, 1
mov dword [rip + 0x2015ed], eax ; [0x6020e8:4]=0x756e672e ; ".gnu.hash" @ 0x6020e8
mov rax, qword [rbp-bytecode]
add rax, 1
mov rdi, rax
call fcn.00400961 ;[i]
mov dword [rbp-instr_ptr_step], 2
jmp 0x400c15 ;[d]
```

Рис. 52: vmloop bb-0aec

```
0x400a80:  
instr_A:  
mov eax, dword [rip + 0x20165a] ; [0x6020e0:4]=0x646c6975 LEA sym.instr_dirty ; "uild-id" @ 0x6  
add eax, 1  
mov dword [rip + 0x201651], eax ; [0x6020e0:4]=0x646c6975 LEA sym.instr_dirty ; "uild-id" @ 0x6  
mov rax, qword [rbp-bytes]  
lea rdx, [rax + 2] ; 0x2  
mov rax, qword [rbp-bytes]  
add rax, 1  
mov rsi, rdx  
mov rdi, rax  
call fcn.0040080d ;[c]  
mov dword [rbp-instr_ptr_step], 3  
jmp 0x400c15 ;[d]
```

Рис. 53: vmloop bb-0a80_full

```
0x400bc1:  
instr_J:  
mov eax, dword [rip + 0x201535] ; [0x6020fc:4]=0x74736e79 ; "ynstr" @ 0x6020fc  
add eax, 1  
mov dword [rip + 0x20152c], eax ; [0x6020fc:4]=0x74736e79 ; "ynstr" @ 0x6020fc  
mov rax, qword [rbp-bytes]  
add rax, 1  
mov rdi, rax  
call fcn.004009b8 ;[m]  
mov dword [rbp-instr_ptr_step], eax  
jmp 0x400c15 ;[d]
```

Рис. 54: vmloop bb-0bc1

```
0x400be5:  
instr_R:  
mov eax, dword [rip + 0x201515] ; [0x602100:4]=0x672e0072 ; "r" 0x00602100 ; "r" @ 0x602100  
add eax, 1  
mov dword [rip + 0x20150c], eax ; [0x602100:4]=0x672e0072 ; "r" 0x00602100 ; "r" @ 0x602100  
mov rax, qword [rbp-bytes]  
add rax, 1  
movzx eax, byte [rax]  
movsx eax, al  
jmp 0x400c61 ;[f]
```

Рис. 55: vmloop bb-0be5

Эта инструкция просто помещает следующий байт-код (первый аргумент) в *eax*, и переходит в конец *vmloop*. Итак, добравшись до инструкции *ret* виртуальной машины, теперь мы знаем, что *vmloop* должна возвращать ``*'', поэтому ``R*'' должен быть последними двумя байтами нашего байт-кода.

Следующая инструкция не вызывает функцию:

```
0x400b6d:
instr_C:
mov eax, dword [rip + 0x201581] ; [0x6020f4:4]=0x79736e79 ; "ynsym" @ 0x6020f4
add eax, 1
mov dword [rip + 0x201578], eax ; [0x6020f4:4]=0x79736e79 ; "ynsym" @ 0x6020f4
mov rax, qword [rbp-bytescode]
add rax, 1
movzx eax, byte [rax]
movsx eax, al
mov dword [rip + 0x201530], eax ; [0x6020c0:4]=0x65746e69 LEA sym.written_by_instr_C ; "interp"
mov dword [rbp-instr_ptr_step], 2
jmp 0x400c15 ;[d]
```

Рис. 56: vmloop bb-0b6d

Это инструкция с одним аргументом, и она записывает свой аргумент по адресу 0x6020c0. Установим флаг на этот адрес!
:
: > f sym.written_by_instr_C 4 @ 0x6020c0

О, у меня предчувствие, что инструкции *instr_C* также в исходном коде crackme соответствовал вызов функции, но он был скопимизирован компилятором (*inlined*). Во всяком случае, у нас есть вот эти две инструкции:

- *instr_R(al)*: выходит с *al*,
- *instr_C(al)*: сохраняет *al* в *sym.written_by_instr_C*.

Также известно, что эти инструкции принимают один аргумент

- *instr_I*,
- *instr_D*,
- *instr_P*,
- *instr_X*,
- *instr_J*,

я эти принимают два аргумента

- *instr_A*,
- *instr_S*.

Остается взломать семь функций, которые вызываются инструкциями, потом построить действительную последовательность байт-кода, дающей нам желанный флаг.

instr_A

Функция, которую вызывает эта инструкция, находится по адресу 0x40080d, поэтому давайте изучать ее!

[offset]> 0x40080d

Совет от r2: В визуальном режиме вы можете просто нажать <Enter> на строке перехода (*jmp* и др.) или вызова *call*, и r2 установит смещение (*seek*) на адрес назначения.

Если попробуем перейти на этот адрес в режиме графа управления, получим сообщение: ``Not in a function''. Введите `df` и задайте его прямо здесь. Функция вызывается из блока *r2*, который не был распознан, поэтому *r2* не удалось найти и эту функцию. Повинуемся и набираем старательно *df*! Функция определена, но нам нужно осмысленное название для нее. Наберем *dr*, находясь все еще в визуальном режиме, назовем эту функцию *instr_A*!

Совет от r2: Все эти команды являются частью меню в визуальном режиме, впервые использованного для определения *sym.memory* как блока данных: *Cd*.

Теперь у нас есть наша новая *fcn.instr_A*, взломаем и ее! Из формы миникарты видно, что есть какой-то каскад *if-then-elif* или оператор *switch-case* в теле этой функции. Это одна из причин, почему миникарта так полезна: можно визуально распознать шаблоны в коде, помогающие при анализе (вспомните легко узнаваемый шаблон цикла нескольких абзацев тому назад). Итак, признаем, что миникарта классная и полезная. Покажем все возможности режима графа управления - сделаем все, что нужно в этом режиме. Первые блоки:

Два аргумента функции (*RDI* и *RSI*) сохраняются в локальных переменных, затем первый сравнивается с 0. Если это так, функция завершается (можно видеть это на миникарте), в противном случае эта же проверка выполняется по второму

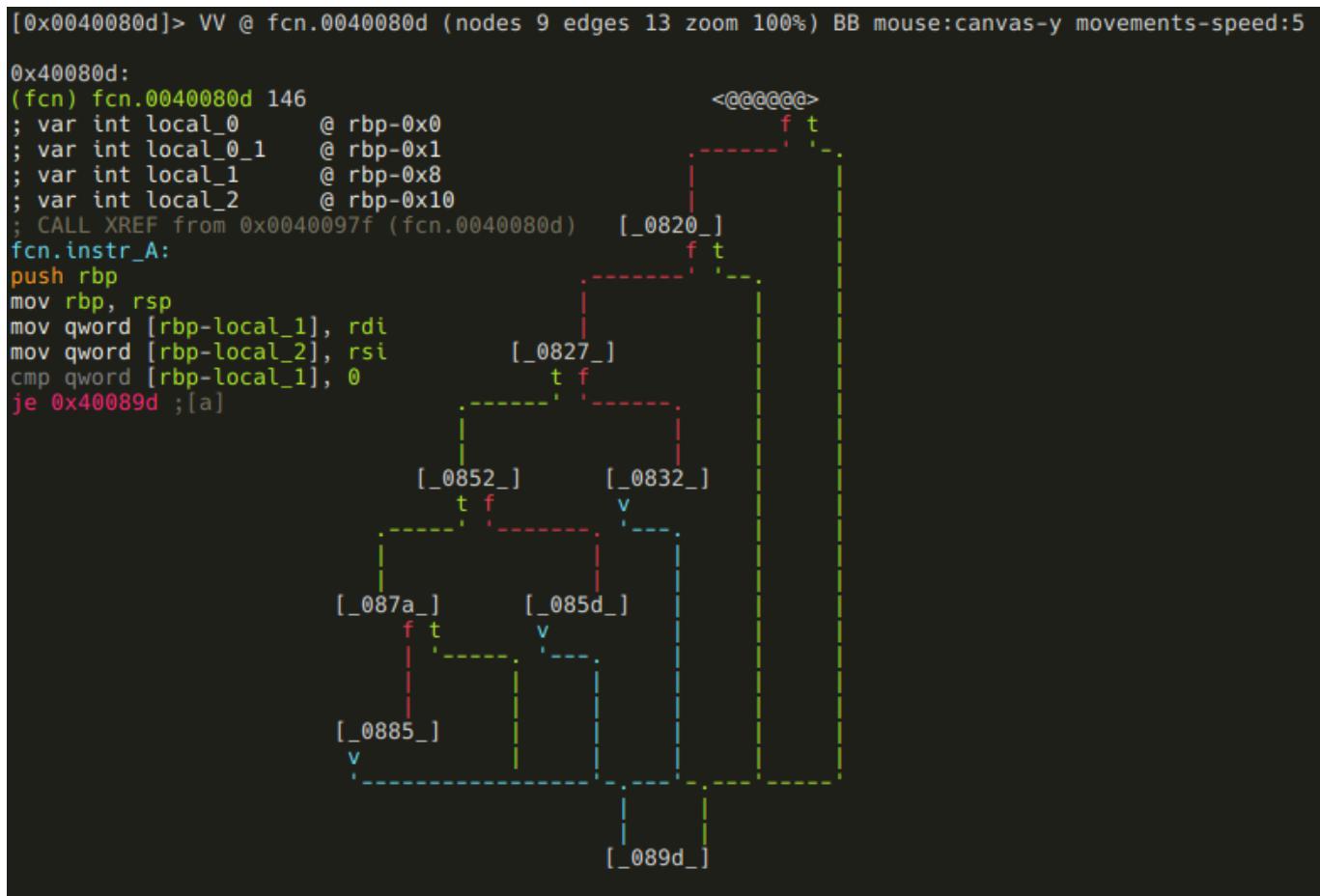


Рис. 57: миникарта инструкции `instr_A`

```
[0x00400080d]> VV @ fcn.00400080d (nodes 9 edges 13 zoom 100%) BB mouse:canvas-y movements-speed:5
```

```
[0x40080d]
(fcn) fcn.00400080d 146
; var int local_0      @ rbp-0x0
; var int local_0_1    @ rbp-0x1
; var int local_1      @ rbp-0x8
; var int local_2      @ rbp-0x10
; CALL XREF from 0x00400097f (fcn.00400080d)
fcn.instr_A:
push rbp
mov rbp, rsp
mov qword [rbp-local_1], rdi
mov qword [rbp-local_2], rsi
cmp qword [rbp-local_1], 0
je 0x40089d ;[a]

          f t
          |
          |
=-- 0x400820
| cmp qword [rbp-local_2], 0
| je 0x40089d ;[a]
=--
```

Рис. 58: instr_A bb-080d

аргументу. Функция завершается, если и второй аргумент равен нулю. Код функции крошечный, но будем придерживаться используемой методологии и переименуем локальные переменные:

```
:> afvn local_1 arg1  
:> afvn local_2 arg2
```

Мы пришли к ранее распознанному оператору switch-case, теперь видно, что значение *arg1* сверяется с ``M'', ``P'' и ``C''.

```
[0x00400080d]> VV @ fcn.00400080d (nodes 9 edges 13 zoom 100%) BB mouse:canvas-y movements-speed:  
|  
|=-----=  
| 0x400827  
| mov rax, qword [rbp-arg1]  
| movzx eax, byte [rax]  
| cmp al, 0x4d ; 'M'  
| jne 0x400852 ;[b]  
|=-----=  
| t f  
|  
|=-----=  
| 0x400852  
| mov rax, qword [rbp-arg1]  
| movzx eax, byte [rax]  
| cmp al, 0x50 ; 'P'  
| jne 0x40087a ;[c]  
|=-----=  
| t f  
|  
|=-----=  
| 0x40087a  
| mov rax, qword [rbp-arg1]  
| movzx eax, byte [rax]  
| cmp al, 0x43 ; 'C'  
| jne 0x40089d ;[a]  
|=-----=  
| f t  
|
```

Рис. 59: значения оператора switch для instr_A

Ветвь ``M'':

Загружается адрес из ячейки 0x602088 и суммируется *arg2* с байтом по этому адресу. Программа r2 показывает нам в комментарии, что 0x602088 изначально содержит адрес *sym.current_memory*, область памяти нужно сконструировать строку ``Such VM! MuCH reV3rse!''. Можно предположить, что надо как-то менять значение, хранящееся по адресу 0x602088, получается, что ветвь «M» может менять байты, отличные от первого. Исходя из этого предположения, пометим 0x602088 как *sym.current_memory_ptr*:

```
:> f sym.current_memory_ptr 8 @ 0x602088
```

Переходим к ветке ``P'':

Фрагмент кода, позволяющий модифицировать *sym.current_memory_ptr*: он прибавляет к нему *arg2*.

Наконец, ветвь ``C'':

Оказывается *instr_C* - это не единственная инструкция, изменяющая *sym.written_by_instr_C*: этот фрагмент кода прибавляет к нему *arg2*.

Мы взломали *instr_A*, подведем итоги! В зависимости от первого аргумента, инструкция исполняет следующее:

```
[0x400832]
mov rax, qword [rip + 0x20184f] ; [0x602088:8]=0x602120 sym.memory ; " !`" @ 0x602088
mov rdx, qword [rip + 0x201848] ; [0x602088:8]=0x602120 sym.memory ; " !`" @ 0x602088
movzx edx, byte [rdx]
mov ecx, edx
mov rdx, qword [rbp-arg2]
movzx edx, byte [rdx]
add edx, ecx
mov byte [rax], dl
jmp 0x40089d ;[a]

V
```

Рис. 60: instr_A switch-M

```
[0x40085d]
mov rdx, qword [rip + 0x201824] ; [0x602088:8]=0x602120 sym.memory LEA sym.current_memory
mov rax, qword [rbp-arg2]
movzx eax, byte [rax]
movzx eax, al
add rax, rdx
mov qword [rip + 0x201810], rax ; [0x602088:8]=0x602120 sym.memory LEA sym.current_memory
jmp 0x40089d ;[a]

V
```

Рис. 61: instr_A switch-P

```
[0x400885]
mov rax, qword [rbp-arg2]
movzx eax, byte [rax]
movzx edx, al
mov eax, dword [rip + 0x20182b] ; [0x6020c0:4]=0x65746e69 LEA sym.written_by_instr_C ; "i
add eax, edx
mov dword [rip + 0x201823], eax ; [0x6020c0:4]=0x65746e69 LEA sym.written_by_instr_C ; "i

V
```

Рис. 62: instr_A switch-C

- *arg1 == "M"*: прибавляет *arg2* к байту по адресу *sym.current_memory_ptr*,
- *arg1 == "P"*: смещает *sym.current_memory_ptr* на *arg2* байтов,
- *arg1 == "C"*: прибавляет *arg2* к значению по адресу *sym.written_by_instr_C*.

instr_S

Эта функция также не распознавалась автоматически, зададим вручную ее аналогично *instr_A*. После этого посмотрим на миникарту, поперемещаем диаграмму, теперь очевидно, что эти две функции очень похожи. Можно также использовать *radiff2* для построения диаграммы различий (diff).

Совет от r2: Программа *radiff2* используется для сравнения двоичных файлов. Есть возможность управлять способом анализа двоичных файлов и требуемым форматом результата. Интересной функцией является порождение графов вида *DarumGrim* с использованием флага *-g*.

Нам надо сравнить две функции из одного и того же двоичного файла, поэтому указываем смещения при помощи флага *-g*, далее применяем *reverse4* для обоих двоичных файлов. Создадим графы сравнения *instr_A* с *instr_S* и сравнения *instr_S* с *instr_A*.

```
[0x00 ~]$ radiff2 -g 0x40080d,0x40089f reverse4 reverse4 | xdot -
```

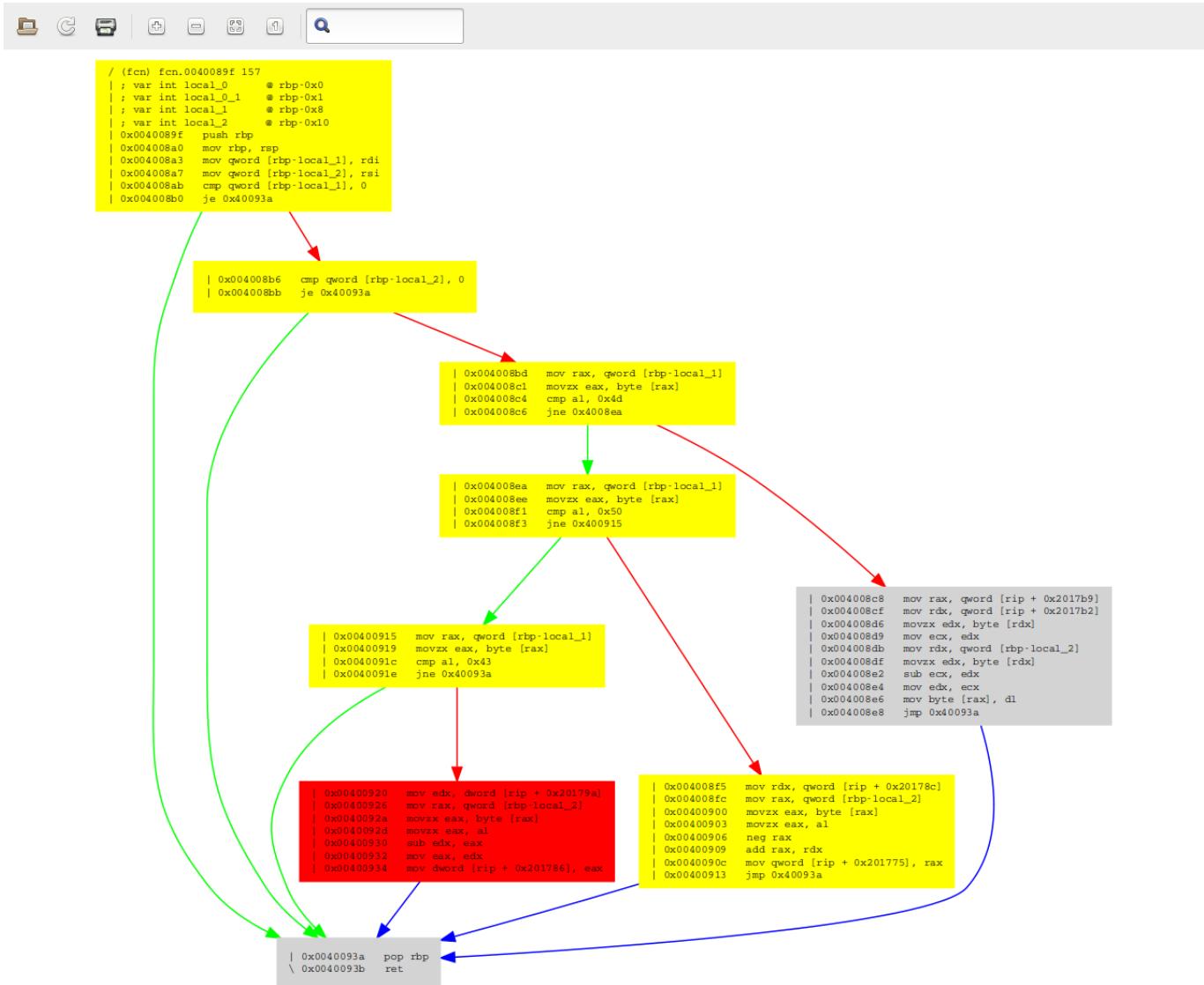


Рис. 63: график сравнения для *instr_S*

```
[0x00 ~]$ radiff2 -g 0x40089f,0x40080d reverse4 reverse4 | xdot -
```

Печальная правда раскрывается после беглого взгляда на эти графы: *radiff2* лжет! Теоретически серые блоки должны быть идентичными, желтые должны отличаться только в некоторых адресах, а красные должны серьезно различаться. Очевидно, что большие серые блоки явно неодинаковы. Определенно надо рыть глубже даже после того, как я закончу эту статью.

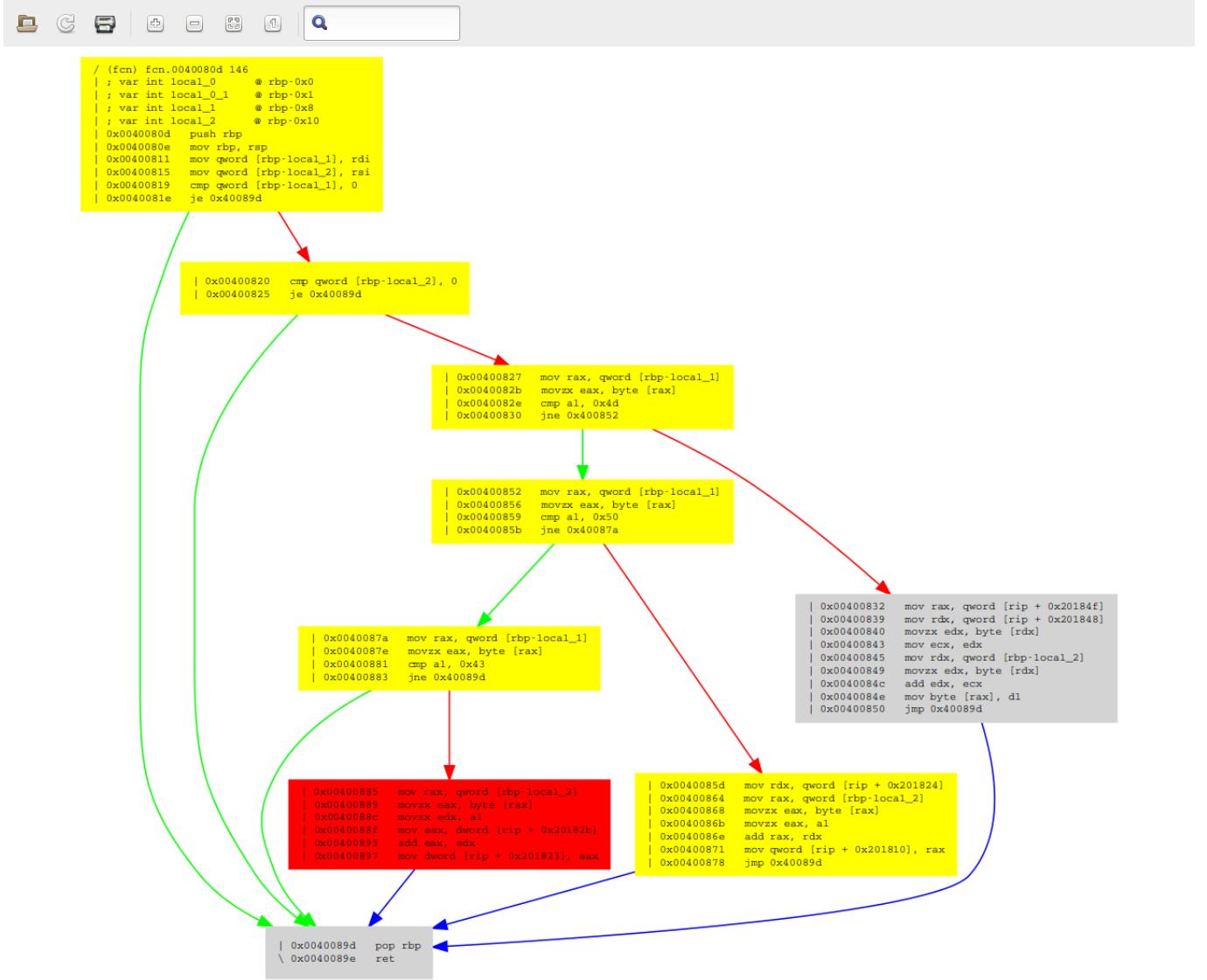


Рис. 64: граф сравнения instr_S

Отойдя от шока, созданного лживым инструментом, легко понимаем, что инструкция *instr_S* по сути является обратной *instr_A*, при этом последняя прибавляет, а первая вычитает. Подведем итог:

- *arg1 == "M"*: вычитает *arg2* из байта по адресу *sym.current_memory_ptr*,
- *arg1 == "P"*: смещает *sym.current_memory_ptr* назад на *arg2* байт,
- *arg1 == "C"*: вычитает *arg2* из значения по адресу *sym.written_by_instr_C*.

instr_I

```
[0x00400961]> VV @ fcn.00400961 (nodes 1 edges 0 zoom 100%) BB mouse:canvas-y movements-speed:5
0x400961:
(fcn) fcn.00400961 37
; var int local_0_1    @ rbp-0x1
; var int local_3      @ rbp-0x18
; CALL XREF from 0x004009b1 (fcn.00400961)
fcn.instr_I:
push rbp                                <@cccccc>
mov rbp, rsp
sub rsp, 0x18
mov qword [rbp-local_3], rdi
mov byte [rbp-local_0_1], 1
lea rdx, [rbp-local_0_1]
mov rax, qword [rbp-local_3]
mov rsi, rdx
mov rdi, rax
call fcn.0040080d ;[a]
leave
ret
```

Рис. 65: instr_I

Инструкция просто вызывает *instr_A(arg1, 1)*. Заметим, что запуск функции выглядит как `call fcn.0040080d` вместо `call fcn.instr_A`. В текущей версии при сохранении и открытии проекта имена функций теряются - еще один момент для исправления в r2!

instr_D

```
[0x0040093c]> VV @ fcn.0040093c (nodes 1 edges 0 zoom 100%) BB mouse:canvas-y movements-speed:5
0x40093c:
(fcn) fcn.0040093c 37
; var int local_0_1    @ rbp-0x1
; var int local_3      @ rbp-0x18
fcn.instr_D:
push rbp                                <@000000>
mov rbp, rsp
sub rsp, 0x18
mov qword [rbp-local_3], rdi
mov byte [rbp-local_0_1], 1
lea rdx, [rbp-local_0_1]
mov rax, qword [rbp-local_3]
mov rsi, rdx
mov rdi, rax
call fcn.0040089f ;[a]
leave
ret
```

Рис. 66: instr_D

Опять же, все просто: она вызывает *instr_S(arg1, 1)*.

instr_P

Время переименовать ее локальные переменные!

```
:> afvn local_0_1 const_M
:> afvn local_0_2 const_P
:> afvn local_3 arg1
```

```
[0x00400986]> VV @ fcn.00400986 (nodes 1 edges 0 zoom 100%) BB mouse:canvas-y movements-speed:5
0x400986:
(fcn) fcn.00400986 50
; arg int arg_9_5      @ rbp+0x4d
; arg int arg_10       @ rbp+0x50
; var int const_M      @ rbp-0x1
; var int const_P      @ rbp-0x2
; var int arg1         @ rbp-0x18           <@000000>
fcn.instr_P:
push rbp
mov rbp, rsp
sub rsp, 0x18
mov qword [rbp-arg1], rdi
mov byte [rbp-const_M], 0x4d ; [0x4d:1]=0 ; 'M'
mov byte [rbp-const_P], 0x50 ; [0x50:1]=64 ; 'P'
mov rax, qword [rip + 0x2016e7] ; [0x602088:8]=0x602120 sym.memory LEA sym.current_memory_ptr ;
mov rdx, qword [rbp-arg1]
movzx edx, byte [rdx]
mov byte [rax], dl
lea rax, [rbp-const_P]
mov rdi, rax
call fcn.00400961 ;[a]
leave
ret
```

Рис. 67: instr_P

Функция также проста, но есть одна странность: `const_M` совсем не используется. Это вероятно какое-то отвлечение внимания хакера?... По сути функция просто сохраняет `arg1` в `sym.current_memory_ptr`, а затем вызывает `instr_I(`P')`. Это означает, что `instr_P` сохраняет один байт и переводит указатель на следующий байт. Эта инструкция идеальна для построения большей части строки ``Such VM! MuCH reV3rse!'', но ее можно использовать только девять раз!

instr_X

Переименовываем локальные переменные ... ну как всегда!

```
:> afvn local_1 arg1
```

```
[0x00400a1f]> VV @ fcn.00400a1f (nodes 1 edges 0 zoom 100%) BB mouse:canvas-y movements-speed:5
0x400a1f:
(fcn) fcn.00400a1f 38
; var int local_0_1    @ rbp-0x1
; var int arg1         @ rbp-0x8
fcn.instr_X:
push rbp
mov rbp, rsp           <@000000>
mov qword [rbp-arg1], rdi
mov rax, qword [rip + 0x20165a] ; [0x602088:8]=0x602120 sym.memory LEA sym.current_memory_ptr ;
mov rdx, qword [rip + 0x201653] ; [0x602088:8]=0x602120 sym.memory LEA sym.current_memory_ptr ;
movzx ecx, byte [rdx]
mov rdx, qword [rbp-arg1]
movzx edx, byte [rdx]
xor edx, ecx
mov byte [rax], dl
pop rbp
ret
```

Рис. 68: instr_X

Функция выполняет операцию XOR со значением по адресу `sym.current_memory_ptr` с `arg1`.

instr_J

Функция не так проста, как предыдущие, но при этом и не сложна. Одержанность переименованием переменных приводит к

```
:> afvn local_3 arg1
```

```
:> afvn local_0_4 arg1_and_0x3f
```

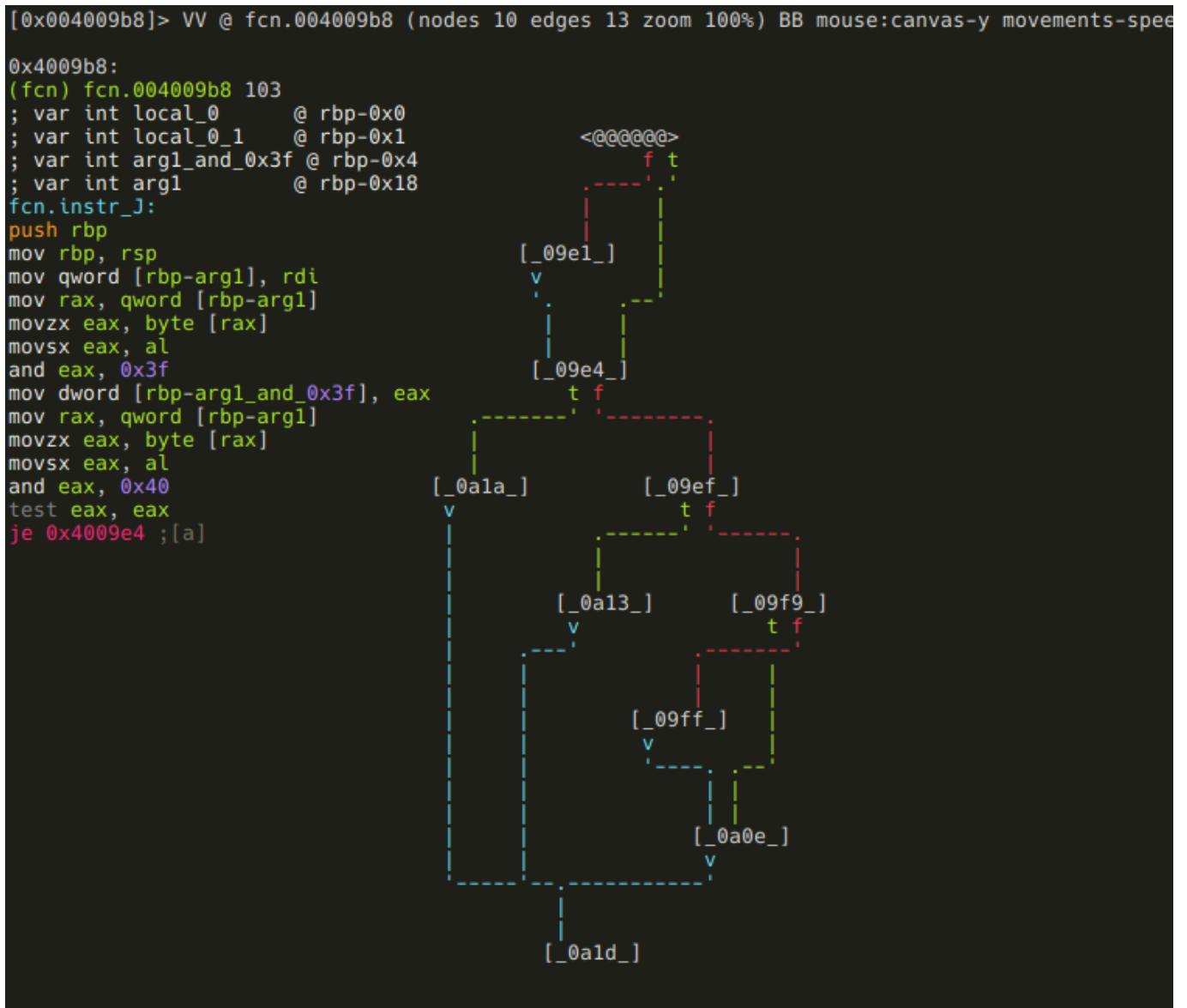


Рис. 69: instr_J

После помещения $arg1 \& 0x3f$ в локальную переменную, $arg1 \& 0x40$ сравнивается с 0. Если результат не равен нулю, биты $arg1_and_0x3f$ инвертируются:

Следующее ветвление: если $arg1 \geq 0$, то функция возвращает $arg1_and_0x3f$,

в противном случае функция разветвляется снова в зависимости от значения `sym.written_by_instr_C`:

Если оно равно нулю, функция возвращает 2,

иначе проверяется, является ли $arg1_and_0x3f$ отрицательным числом,

и если это так, то `sym.good_if_ne_zero` увеличивается на 1:

После всех проверок функция возвращает $arg1_and_0x3f$:

.instructionset

Теперь все инструкции виртуальной машины взломаны, получено полное представление об их работе. Приведем набор инструкций виртуальной машины:

Инструкция	Первый аргумент	Второй аргумент	Функция исструкции
``A''	``M''	arg2	*sym.current_memory_ptr += arg2

Инструкция	Первый аргумент	Второй аргумент	Функция исструкции
``S"	``P"	arg2	sym.current_memory_ptr += arg2
	``C"	arg2	sym.written_by_instr_C += arg2
``M"	arg2	arg2	*sym.current_memory_ptr -= arg2
	``P"	arg2	sym.current_memory_ptr -= arg2
``C"	arg2	arg2	sym.written_by_instr_C -= arg2
	arg1	отсутствует	instr_A(arg1, 1)
``D"	arg1	отсутствует	instr_S(arg1, 1)
``P"	arg1	отсутствует	*sym.current_memory_ptr = arg1; instr_I(`P")
``X"	arg1	отсутствует	*sym.current_memory_ptr ^= arg1
``J"	arg1	отсутствует	arg1_and_0x3f = arg1 & 0x3f;if (arg1 & 0x40 != 0) arg1_and_0x3f *= -1if (arg1 >= 0) return arg1_and_0x3f;else if (*sym.written_by_instr_C != 0) { if (arg1_and_0x3f < 0) ++*sym.good_if_ne_zero; return arg1_and_0x3f;} else return 2;
``C"	arg1	отсутствует	*sym.written_by_instr_C = arg1
``R"	arg1	отсутствует	return(arg1)

.bytecode

Итак, реверс-инжениринг мы сделали, теперь нужно спроектировать программу виртуальной машины с набором инструкций, описанным в предыдущем абзаце. Функциональная спецификация программы:

- программа должна вернуть ``*'',
 - *sym.memory* должна содержать строку ``Such VM! MuCH reV3rse!'' после исполнения,
 - все девять инструкций должны быть использованы хотя бы один раз,
 - *sym.good_if_ne_zero* должно стать нулем,
 - instr P нельзя использовать более девяти раз.

Поскольку этот документ посвящен реверс-инженерингу, оставляю эту задачу на усмотрение читателю :). Однако оставлять вас с пустыми руками не буду, и дам один совет: за исключением "J", все инструкции примитивны, просты в использовании. Проблем с реализацией «Such VM! MuCH reV3rse!» на основе этих инструкций быть не должно. Инструкция «J» сложнее в сравнении с другими. Ее единственная задача состоит в том, чтобы сделать переменную `sym.good_if_ne_zero` большей нуля, это - требование доступа к флагу.

Чтобы увеличить на единицу `sym.good_if_ne_zero` должны выполняться три условия:

- $arg1$ должен быть отрицательным числом, иначе мы выйдем раньше,
 - $sym.written_by_instr_C$ не должен быть 0 в момент вызова «J», это означает, что инструкции «C», «AC» и «SC» следует использовать перед вызовом «J»;
 - $arg1 \text{ and } 0x3f$ должен быть отрицательным в момент проверки.

Так как бит знака 0x3f равен нулю, несмотря ни на значение $arg1$, результат выражения $arg1 \& 0x3f$ всегда будет неотрицательный. Помните, что «J» инвертирует биты $arg1_and_0x3f$ если $arg1 \& 0x40$ не равен нулю. Это означает, что 6-й бит $arg1$ должен быть 1 ($0x40 = 01000000_b$). Также из-за того, что $arg1_and_0x3f$ тоже не может быть 0, по крайней мере один из битов в позициях 0, 1, 2, 3, 4, 5 $arg1$ должен быть 1 ($0x3f = 00111111_b$).

Теперь информации достаточно, можно писать программу. Или же можно просто сделать быстрый и грязный реверс-инженеринг, использованный ранее в CTF:

\x90\x00PSAMuAP\x01AMcAP\x01AMhAP\x01AM AP\x01AMVAP\x01AMMAP\x01AM!AP\x01AM AP\x01AMMAP\x01AMuAP\x01AMCAP\x01AMHAP\x01AM

Имейте в виду, что этот результат получен "на лету", параллельно фазе взлома. Например, есть части, реализованные без перечня всех возможных инструкций. Это означает, что код уродлив и неэффективен.

.outro

Итак, что еще можно сказать? Сложная виртуальная машина, много пришлось взламывать! ;)

То, что начиналось как обзор простого crackme, превратилось в довольно детальное руководство по r2, так что спасибо, если вы его до самого конца прочли. Надеюсь вы остались довольны результатом (я знаю, что это так) и даже чему-то научились. Я также многое узнал о r2 в процессе обзора, и даже сделал несколько патчей, у меня появилось несколько идей о возможных улучшениях.



Рис. 70: instr_J bb-09e1



Рис. 71: instr_J bb-09e4

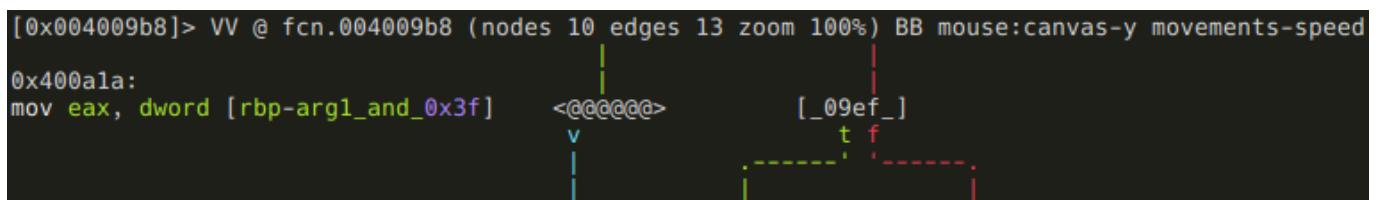


Рис. 72: instr_J bb-0a1a

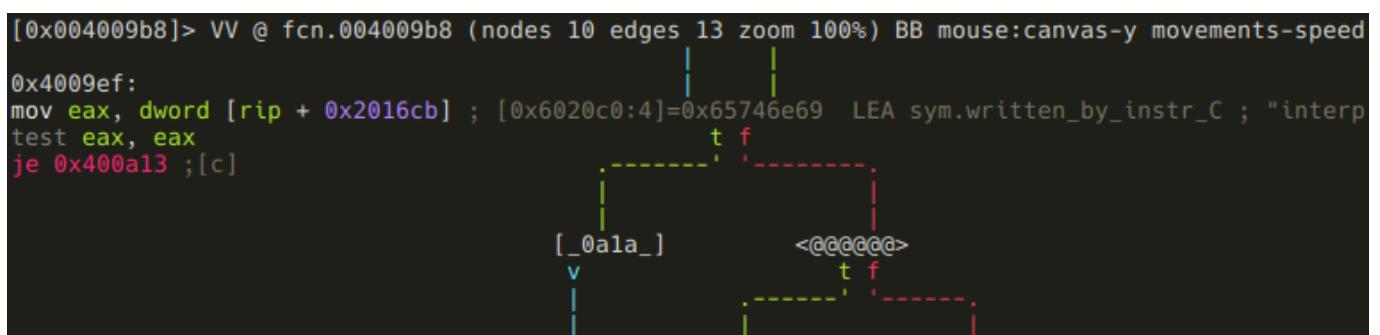


Рис. 73: instr_J bb-09ef

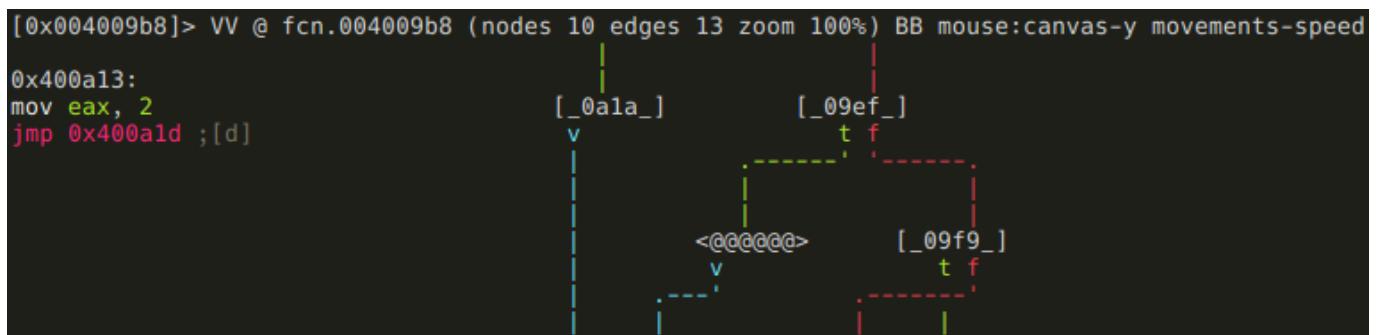


Рис. 74: instr_J bb-0a13

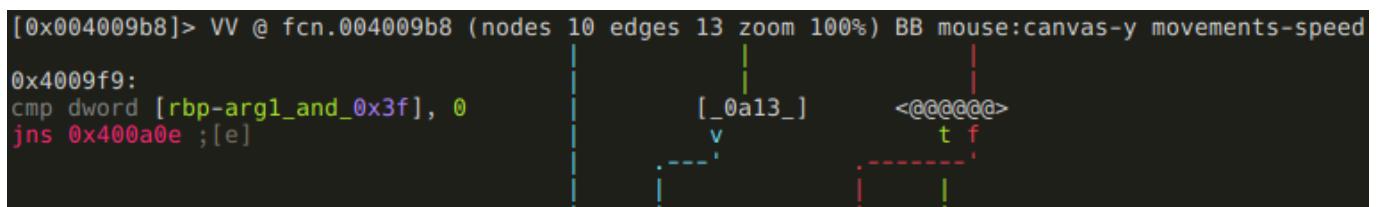


Рис. 75: instr_J bb-09f9

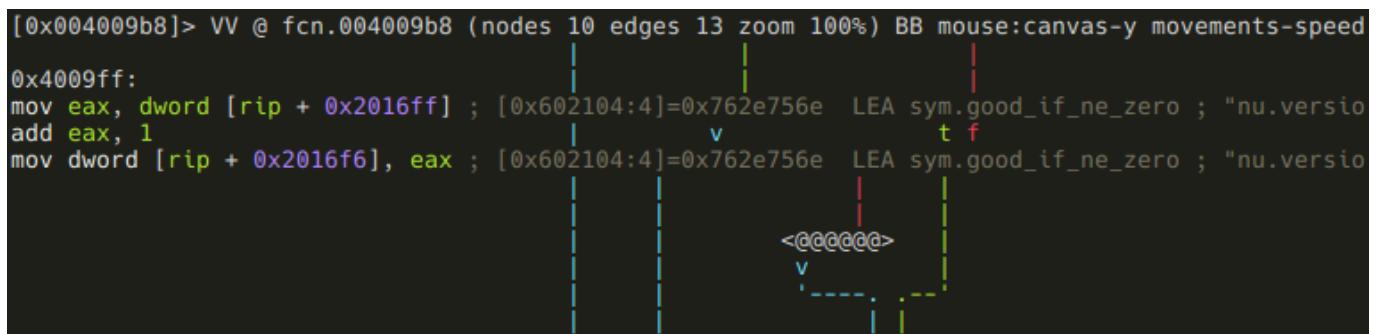


Рис. 76: instr_J bb-09ff



Рис. 77: instr_J bb-0a0e

Справочная карта Radare2

Этот раздел книги основан на справочной карте, разработанной Thanat0s-ом, лицензия - GNU GPL. Первоначальная лицензия была следующая:

Эту карточку можно свободно распространять в соответствии с общей публичной лицензией GNU, Copyright Thanat0s - v0.1.

Руководство по выживанию

Здесь приведены базовые команды, которые необходимо выучить для перемещения по двоичному файлу, получения о нем информации.

Команда	Описание
s (tab)	Установить смещение куда-либо
x [nbytes]	Шестнадцатеричный дамп n байтов, \$b по умолчанию
aa	Анализировать автоматически
pdf@ [funcname](Tab)	Дизассемблировать функцию (main, fcn, и т.д.)
f fcn(Tab)	Перечислить функции
f str(Tab)	Перечислить строки
fr [flagname] [newname]	Переименовать флаг
psz [offset]~grep	Вывод строки и grep-пинг нужной
axF [flag]	Найти перекрестные ссылки на флаг

Флаги

Флаг --- это закладка в коде бинарного файла, но с дополнительной информацией, в частности, размер, теги и ассоциированное с ним пространство флагов. Команда `f` используется для перечисления, задания флагов, а также получения информации о них.

Команда	Описание
f	Перечислить флаги
fd \$\$	Показать информацию о смещении
fj	Показать флаги в формате JSON
fl	Показать длину флага
fx [flagname]	Показать шестнадцатеричный дамп флага
fC [name] [comment]	Показать комментарий о флаге

Пространство флагов

Флаги создаются в пространствах имен, по умолчанию пространство не выбрано, и команда перечисления флагов будет показывать их все. Если требуются флаги из конкретного подмножества, используется команда `fs`, позволяющая задавать ограничения.

Команда	Описание
fs	Показать пространства флагов
fs *	Выбрать все пространства флагов
fs [space]	Выбрать пространство флагов

Информация

Двоичные файлы включают в себя важную информацию, хранящуюся в заголовках. Команда `i` использует API RBin и позволяет получать информацию в rabin2. Вот наиболее часто используемые команды.

Команда	Описание
ii	Информация об импортах
iI	Информация о двоичном коде
ie	Показать точку входа
iS	Показать секции
ir	Показать переопределения

Команда	Описание
iz	Перечислить строки (izz, izzz)

Печать строк

Существуют различные способы представления строк в памяти. Команда `ps` позволяет печатать их в кодировке utf-16, в форматах pascal, C (заканчивающихся нулем), и др.

Команда	Описание
psz [offset]	Печать строки C (с нулем на конце)
psb [offset]	Печать строки в текущем блоке
psx [offset]	Показать строку с использованием escape-последовательностей
psp [offset]	Печать строки pascal
psw [offset]	Печать wide-строки

Визуальный режим

Визуальный режим поддерживает интерактивный режим взаимодействия с radare2. Вход в этот режим --- команда `V` или `V,` после чего большинство операций выполняется клавишами клавиатуры.

Команда	Описание
V	Вход в визуальный режим
p/P	Смена режимов отображения (hex, disasm, debug, words, buf)
c	Включить/выключить курсор (c)
q	Вернуться в командную строку radare2
hjkl	Перемещение области на экране (или HJKL) (влево-вниз-вверх-вправо)
Enter	Перейти по адресу jump-а или call
sS	Step/step over (отладка)
o	Переключиться между asm.pseudo и asm.esil
.	Установить смещение на адрес в программном счетчике
/	В режиме курсора провести поиск в текущем блоке
:cmd	Запустить команду radare
;[-]cmt	Добавить или удалить комментарий
/*+-[]	Изменить размер блока, [] = resize hex.cols
<,>	Поиск с учетом выравнивания размер блока
i/a/A	вставить (i) hex, (a)ссемблировать код, визуальный (A)ссемблер
b	Включить/выключить точку останова
B	Просмотр evals, symbols, flags, classes, ...
d[f?]	Задать функцию, описать данные, код, ...
D	Вход в режим визуального сравнения (установить diff.from/to)
e	Редактировать конфигурационные переменные
f/F	Задать/удалить флаг
gG	Установить смещение на начало или конец файла (0-\$s)
mK/'K	Пометить/перейти на Ключ (любой ключ)
M	Обойти подмонтированные файловые системы
n/N	Установить смещение на следующую/предыдущую функцию/флаг/вхождение поиска (scr.nkey)
C	Переключить отображение в цвете или ч/б
R	Сгенерировать случайную палитру цветов (escr)
tT	Управление панелями на экране. Читайте раздел Панели
v	Меню анализа кода в визуальном режиме
V	Просмотр графа (agv?)
wW	Переместить курсор на следующее/предыдущее слово
uU	Отменить/повторить установку смещения
x	Показать перекрестные ссылки (xrefs) для текущей функции из/в данные/код
yY	Копирование и вставка выделения
z	скрыть/раскрыть комментарии в дизассемблировании

Поиск

Возникает много разных задач, где требуется найти значение внутри двоичного файла или в конкретных областях. Синтаксическая структура `e search.in=?` позволяет указывать, где команда / будет осуществлять поиск заданного значения.

Команда	Описание
/ foo\00	Поиск строки 'foo\0'
/b	Поиск в обратном направлении
//	повторить предыдущий поиск
/w foo	Поиск wide-строки 'f0o\0o\0'
/wi foo	Поиск wide-строки, игнорируя размер букв
!/ ff	Поиск первых несоответствующих вхождений
/i foo	Поиск строки 'foo', игнорируя размер букв
/e /E.F/i	Сопоставить с регулярным выражением
/x a1b2c3	Поиск байтов; пробелы и [0-9a-fA-F] допустимы, аналогично /x A1 B2 C3
/x a1..c3	Поиск байтов, игнорируя некоторые hex-числа (автоматически генерированная маска, в этом примере: ff00ff)
/x a1b2:fff3	Поиск байтов по маске (указываются индивидуальные биты)
/d 101112	Поиск делтифицированной последовательности байтов
!/x 00	Инверсный поиск hex (найти первый байт != 0x00)
/c jmp [esp]	Поиск ассемблерного кода (смотрите search.asmstr)
/a jmp eax	Ассемблировать оп-код и искать его байты
/A	Поиск открытых AES-ключей
/r sym.printf	Анализировать, куда ссылается оп-код
/R	Поиск гаджетов ROP
/P	Показать смещение предыдущей инструкции
/m magicfile	Поиск следующего magic-файла
/p patternsize	Поиск шаблона заданного размера
/z min max	Поиск строки заданного размера
/v[?248] num	Поиск 32-битового значения, учитывая asm.bigendian

Сохранения (сломано)

Эти функции не работают так, как указано в спецификации на момент написания книги (16 ноября 2020). Дополнительная информация - #Ошибка 6945: META - Project files и #Ошибка 17034.

Чтобы записать результаты анализа (пока проблемы не решены) напишите себе скрипт, который будет сохранять имя функции, имена переменных, и т.д., например:

```
vim sample_A.r2
```

```
e scr.utf8 = false
s 0x000403ce0
aaa
s fcn.00403130
afn return_delta_to_heapaddr
afvn iter var_04h
...
```

Использование переменных в выражениях

Команда `?$?` показывает переменные, которые можно использовать в математических операциях в командной строке r2. Например, команда `? $$` вычисляет число, а `?v` - печатает значение в одном из форматов. Все команды r2 принимают эти переменные в качестве параметров.

Команда	Описание
here() \$	текущее невременное виртуальное смещение
\$?	последний результат сравнения
\$alias=value	задать синоним (простейшие макросы)
\$b	размер блока
\$B	базовый адрес (выровненный наименьший адрес отображения)
\$f	адрес перехода jump, если условие ложно (jz 0x10 => следующая инструкция)

Команда	Описание
\$fl	длина флага (размер), ассоциированного с данным адресом (fla; pD \$l @ entry0)
\$F	размер текущей функции
\$FB	начало функции
\$Fb	адрес текущего базового блока
\$Fs	размер текущего базового блока
\$FE	конец функция
\$FS	размер функции
\$Fj	целевой адрес перехода из функции
\$Ff	целевой false-адрес функции
\$FI	инструкции функции
c,r	получить ширину и высоту терминала
\$Cn	получить n-тый вызов функции
\$Dn	получить n-ю ссылку на данные в функции
\$D	текущий адрес базы отображения в режиме отладки ?v \$D @ rsp
\$DD	текущий размер отображения в режиме отладки
\$e	1 если в конце блока, иначе 0
\$j	адрес jump-a (например, jmp 0x10, jz 0x10 => 0x10)
\$Ja	получить n-тый jump функции
\$Xn	получить n-ый xref функции
\$l	длина оп-кода
\$m	ссылка оп-кода на память (например, mov eax,[0x10] => 0x10)
\$M	адрес отображения (наименьший адрес отображения)
\$o	here (текущее смещение на диске, io)
\$p	getpid()
\$P	pid дочернего процесса (только в отладчике)
\$s	размер файла
\$S	смещение секции
\$SS	размер секции
\$v	immediate-значение оп-кода (например, lui a0,0x8010 => 0x8010)
\$w	получить размер слова, 4 если asm.bits=32, 8 если 64, ...
\${ev}	значение конфигурационной переменной
\${reg}	значение поименованного регистра
\${kv}	получить ответ на запрос к sdb
\${flag}	размер флага
RNum	переменные формата \$variables, полезные в математических выражениях

Авторы и активные участники

Этой книги не было бы без помощи большого числа участников, рецензировавших, дописывающих ее и славших сообщения об ошибках и других проблемах в проекте radare2.

Книга radare2

Книга была начата maijin как новая версия оригинальной книги о radare, написанной pancake.

- Книга о radare1 находится здесь <http://www.radare.org/get/radare.pdf> (ссылка не открывается)

Большое спасибо всем, кто участвовал в создании книги на gitbook:

Adrian Studer, Ahmed Mohamed Abd El-MAwgood, Akshay Krishnan R, Andrew Hoog, Anton Kochkov, Antonio Sánchez, Austin Hartzheim, Aswin C (officialcjunior), Bob131, DZ_ruyk, David Tomaschik, Eric, Fangrui Song, Francesco Tamagni, FreeArtMan, Gerardo García Peña, Giuseppe, Grigory Rechistov, Hui Peng, ITAYC0HEN, Itay Cohen, Jeffrey Crowell, John, Judge Dredd (key 6E23685A), Jupiter, Kevin Grandemange, Kevin Laeufer, Luca Di Bartolomeo, Lukas Dresel, Maijin, Michael Scherer, Mike, Nikita Abdullin, Paul, Paweł Łukasik, Peter C, RandomLive, Ren Kimura, Reto Schneider, SchumBlubBlub, SkUaTeR, Solomon, Srimanta Barua, Sushant Dinesh, TDKPS, Thanat0s, Vanellope, Vex Woo, Vorlent, XYlearn, Yuri Slobodyanyuk, ali, aoighost, condret, hdznrrd, izhuer, jvoisin, kij, madblobfish, muzlightbeer, pancake, polym (Tim), puddl3glum, radare, sghtoma, shakreiner, sivaramaaa, taiyu, vane11ope, xarkes.