Лекция 20

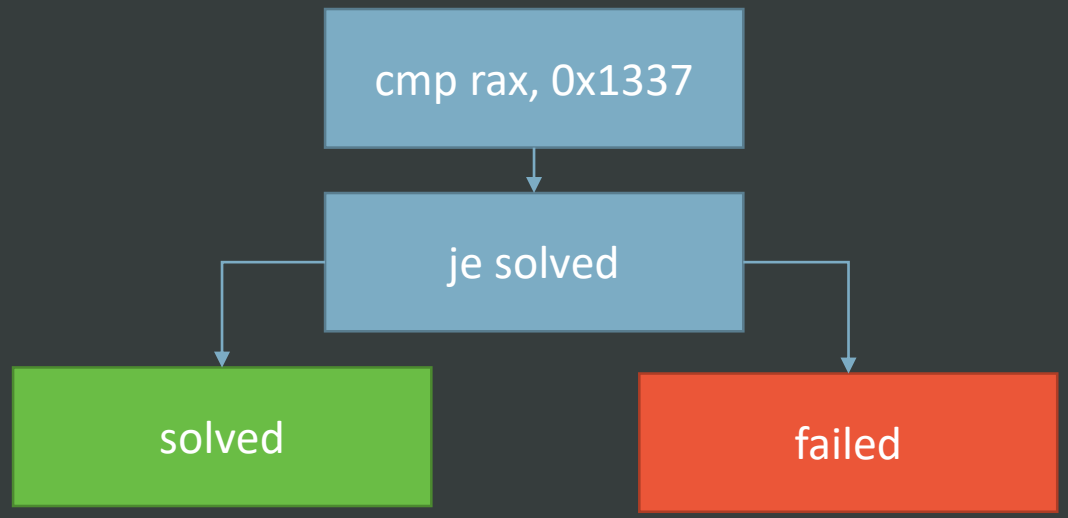
ПРО АВТОМАТИЧЕСКИЙ АНАЛИЗ, АТАКИ ПО СТОРОННИМ КАНАЛАМ И НЕНАТИВНЫЕ ЯЗЫКИ

rev. 5

Символьное выполнение

Символьное выполнение

- Что если исполнять программу на каком-то виртуальном процессоре, у которого в регистрах лежат не конкретные значения, а некоторые переменные?
  - А в процессе выполнения операций и обработки условий из этих переменных можно будет составить уравнения
- Этот подход широко используется в автоматических деобфускаторах и научных работах

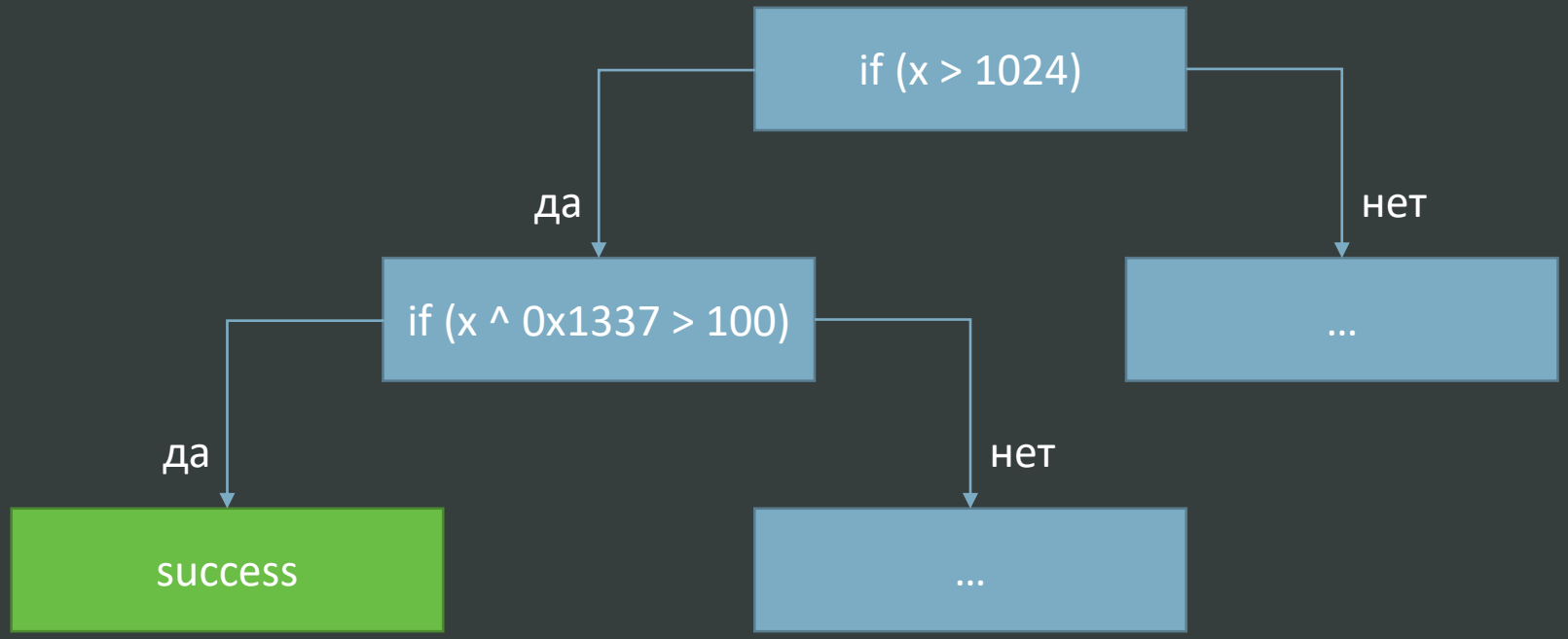



```
RAX = BitVec('RAX', 64)
s = Solver()
s.add(RAX == 0x1337)
```

```
RAX = BitVec('RAX', 64)
s = Solver()
s.add(RAX != 0x1337)
```

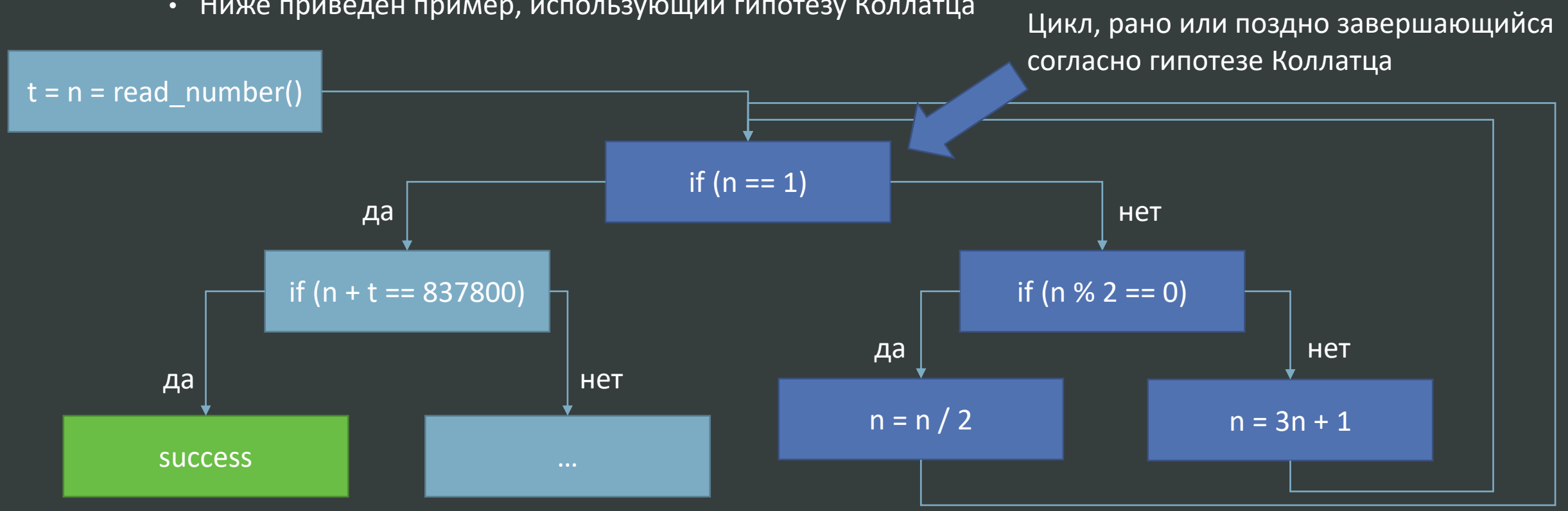
Символьное выполнение

- Этот подход позволяет при желании составить условия для посещения любой ветви программы без полного перебора
  - В противоположность обычному исполнению



Символьное выполнение

- Из-за этого, однако, страдает от проблемы, называемой "Path explosion" – экспоненциального роста числа возможных путей при каждом посещении условия
  - Ниже приведен пример, использующий гипотезу Коллатца



Angr

- Классический инструмент для автоматического анализа, использующий этот подход
- Доступен для загрузки через pip, подробнее прочитать про него можно на сайте https://angr.io/
  - Рекомендуется ставить его по инструкции, через virtualenv, но лично я ставлю его просто через pip install angr
  - В крайнем случае есть и Docker-образ
- Создан при участии CTF-команды Shellphish (UCSB)
- Позволяет очень многое, в том числе решать различные CrackMe в полностью автоматическом режиме
- Использует в качестве SMT-решателя Z3
  - С Z3 мы уже встречались в разделе «криптография»

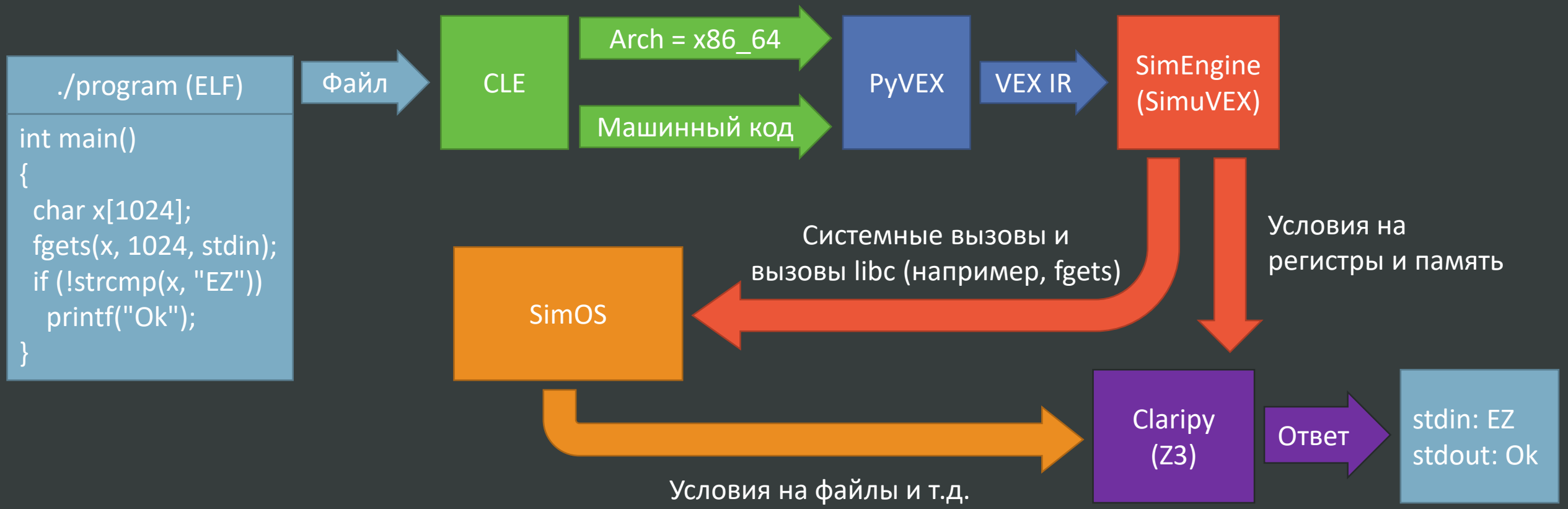
Angr

- Angr поддерживает очень большое число процессорных архитектур:
  - ARM / ARM64
  - x86 / x86_64
  - MIPS / MIPS64
  - Некоторые другие
- Также angr поддерживает следующие ОС:
  - Linux
  - Windows
  - Android, правда, его поддержка пока совсем на начальном этапе

Как работает angr?

- Прежде всего, angr должен загрузить исполняемый файл, это он делает при помощи библиотеки angr CLE
  - Тут происходит обработка ELF, Portable Executable и т.д.
- Потом angr преобразует код в некоторое промежуточное представление (вроде LLVM IR) при помощи VEX (Valgrind)
- Затем это промежуточное представление используется в SimuVEX для символьного исполнения кода
  - В результате этого исполнения появляются различные условия на регистры, например
- Программа может как-то взаимодействовать с окружающим миром, тут на помощь приходит SimOS
  - Системные вызовы и некоторые функции libc обрабатываются как раз здесь
- Затем полученные условия решаются при помощи решателя Claripy (Microsoft Z3)

Как работает angr?



* если учесть самомодифицирующийся код и dlopen, на этой схеме может заметно прибавиться стрелочек

Изучаем angr

- Лучше всего изучать Angr по примерам
  - Примеры доступны по ссылке https://docs.angr.io/examples
  - Еще больше примеров можно найти на GitHub: https://github.com/angr/antr-doc/tree/master/examples
- Также можно почитать документацию на сайте https://docs.angr.io/
- Попробуем разобрать основные моменты в рамках этой презентации
  - Будут, однако, и моменты, про которые в документации не написано

Изучаем angr

- Будем пытаться взломать следующий простой пример:

```c
#include <stdio.h>
int main()
{
    int number;
    printf("Please enter number: ");
    scanf("%d", &number);
    if (number == 1337)
        printf("Good job!\n");
    else
        printf("Wrong!\n");
}
```

Angr. Загружаем исполняемый файл

- Работа с любым исполняемым файлом в angr начинается с его загрузки
  - Для этого используется конструктор angr.Project()

```
# Импортируем модуль angr
import angr
# Загружаем исполняемый файл
p = angr.Project('simplest')
```

- У этого конструктора есть следующий полезный параметр:
  - auto_load_libs (по умолчанию True) – параметр, указывающий на то, стоит ли angr пытаться загрузить реальные библиотеки, которые использует процесс или нет. Если указать False, будут использоваться только функции из состава SimProcedures. Отключение ускоряет анализ и избавляет от некоторых непонятных ошибок, которые могут возникнуть в библиотеке libc (правда, нереализованные функции заменятся пустышками). Почитать мнение авторов angr об опции можно тут: https://twitter.com/angrdothorse/status/1046121168229416960

Angr. Состояния

- При работе angr оперирует состояниями, включающими в себя содержимое регистров, памяти, файловой системы и т.д.
  - Соответственно, в процессе исполнения происходит всего лишь переход из состояния в состояние
- Состояния имеют тип SimState и имеют следующие важные поля:
  - regs – набор регистров (с именами конкретной платформы, например rax)
  - mem – память (массив, к которому можно получать доступ по адресам)
  - posix – объект, позволяющий получить доступ к различным частям эмулируемой системы POSIX, например, файлам и сокетам
    - Особенно часто используется метод posix.dumps(fd), позволяющий представить содержимое файлового дескриптора fd как текст
- Все состояния являются неизменными в процессе выполнения (immutable)
  - При переходе в следующее состояние, старое не изменяется (а создается новое)

Angr. Создаем начальное состояние

- Раздобудем начальное состояние, с которого мы начнем исследование
- Создать его можно при помощи следующих методов p.factory:
  - blank_state() – создает максимально пустое неопределенное состояние
  - entry_state() – создает состояние для запуска программы (т.е. в ее точке входа)
  - full_init_state() – как entry_state, только еще и с учетом конструкторов библиотек
  - call_state(addr, arg1, arg2, …) – создает состояние для вызова функции в программе
- Все эти методы принимают адрес addr, для всех кроме call_state он опционален
- entry_state и full_init_state также могут принимать args и env – список аргументов и переменных окружения

```
# Создаем состояние в начале программы
state = p.factory.entry_state()
```

Angr. Выполняем программу

- Чтобы приступить к выполнению программы и получить что-то поинтереснее чем первоначальное состояние, нам нужен будет SimulationManager
- Его также можно создать при помощи метода p.factory:
  - simulation_manager() или simgr(), в качестве аргумента можно передать состояние, список состояний или ничего (это эквивалентно передаче одного состояния entry_state)

```
# Создаем SimulationManager
sm = p.factory.simulation_manager(state)
```

Angr. Выполняем программу

- В SimulationManager состояния лежат в списках, вот некоторые из них:
  - active – состояния, исполнение которых можно продолжить (сюда, например, кладутся состояния в момент инициализации)
  - deadended – состояния, выполнение которых почему-то нельзя продолжить (не нашлось решение, был вызван exit(), и т.д.)
  - errored – состояния, выполнение которых закончилось ошибкой (например, программа попыталась обратиться к несуществующей памяти)
- Также при помощи функции use_technique можно указать стратегию, которую будет использовать angr при выполнении состояний вместо поиска в ширину, например чтобы попробовать избежать Path explosion
  - Почитать о встроенных стратегиях можно тут: https://docs.angr.io/en/latest/core-concepts/pathgroups.html#exploration-techniques

```
# Указываем стратегию поиска DFS - поиск в глубину
sm.use_technique(angr.exploration_techniques.DFS())
```

Angr. Выполняем программу

- Теперь, когда у нас есть SimulationManager, можно что-нибудь выполнить при помощи следующих функций:
  - step() – позволяет выполнить некоторый «шаг» минимального размера
  - run() – запускает выполнение до тех пор, пока это вообще возможно (таким образом, все состояния из списка active перейдут в deadended)
  - explore(find, avoid) – запускает поиск состояний, которые удовлетворяют условию find и не удовлетворяют (в любой момент времени) условию avoid
    - В find и avoid можно указать адреса или списки адресов, в которые вы хотите чтобы программа попала и не попала (в любой из find и ни в один из avoid)
    - Также в find и avoid можно передать функцию, принимающую состояние в качестве аргумента, и возвращающую True если состояние является желаемым / нежелательным соответственно
    - Оба параметра являются опциональными

```
# Ищем состояния, где нам пишут, что задача решена верно
sm.explore(find=lambda s: b"Good job" in s.posix.dumps(1))
```

Angr. Получаем результат

- Теперь, когда мы нашли нужный путь исполнения, можно посмотреть, что именно мы должны ввести с клавиатуры при помощи posix.dumps(0)
- Полный скрипт решения будет выглядеть следующим образом:

```
import angr
# Загружаем исполняемый файл
p = angr.Project('simplest')
# Создаем состояние в начале программы
state = p.factory.entry_state()
# Создаем SimulationManager
sm = p.factory.simulation_manager(state)
# Ищем состояния, где нам пишут, что задача решена верно
sm.explore(find=lambda s: b"Good job" in s.posix.dumps(1))
# Печатаем искомое содержимое stdin
print(sm.found[0].posix.dumps(0))
```

```
b'0000001337'
```

Время задач

Странная архитектура

Категория: Lesson 20 / Angr + Pin + Non-native

Решивших: 0

Время: 00:00:03

- Доступ к задачам можно получить как всегда на nsuctf.ru
- В этой задаче вам может пригодиться angr: https://angr.io/
- И код с прошлого слайда: https://pastebin.com/QYQRFyma

Angr. Добавляем условия

- До этого мы полагались на то, что необходимые нам данные вводятся в программу с клавиатуры, причем про них ничего не известно
- Чтобы немного разнообразить это поведение – можно использовать модуль claripy
  - По использованию он очень похож на Microsoft Z3, если вы с ним знакомы
- При помощи claripy можно создавать символьные переменные и, соответственно, запихивать их в удобные места программы
  - Будь то параметры командной строки, переменные окружения или значения регистров
- Также на эти переменные можно навешивать условия
  - В общем, все как в Z3Py, только с настоящими программами

Claripy

- Символьные переменные создаются при помощи классов, доступных в модуле claripy:
  - BVV(value, size) и BVS(name, size) позволяют создать битовые константы и вектора
    - size – размер битового вектора в битах, name – имя, value – значение константы
  - FPV(value, type) и FPS(name, type) позволяют создавать дробные константы и переменные
    - Type – claripy.fp.FSORT_DOUBLE для double и claripy.fp.FSORT_FLOAT для float
  - BoolV(value) и BoolS(name) позволяют создавать булевы константы и переменные
- Увы, настоящих строк тут (пока?) нет, так что вместо них приходится использовать BVS(…, длина_строки * 8)
  - Метод BV.get_byte(i) позволяет получить конкретный байт из такой «строки»
  - Метод BV.chop(n) позволяет разбить битовый вектор на куски размера n бит
- Более полная документация доступна в https://api.angr.io/projects/claripy/en/latest/

Angr. Добавляем условия

- Условия добавляются при помощи метода state.add_constraints
  - Как и в Z3 сюда можно просто писать выражение, которое вы хотели бы чтобы имело значение True
  - Условия можно объединять при помощи claripy.Or, claripy.And и т.д. (аналогично Z3Py)
- Условия зачастую можно добавлять как до поиска пути в SimulationManager, так и после (тут все тоже как в Microsoft Z3)
- Чтобы получить реальное значение символьной переменной после решения, можно использовать метод state.solver.eval(var)
  - Чтобы получить в результате строку, можно добавить аргумент cast_to=bytes
  - Существуют дополнительные методы вроде eval_atleast(var, n) и eval_upto(var, n), позволяющие получить множество решений

Claripy. Простой пример

- Рассмотрим простую задачу, использующую argv и решение ее при помощи angr:

```c
#include <stdio.h>
int main(int argc, char** argv)
{
    int number;
    number = atoi(argv[1]);
    if (number == 1337)
        printf("Good job!\n");
    else
        printf("Wrong!\n");
}
```

```python
import angr
import claripy
p = angr.Project('argv')
# Создаем переменную длиной в 5 символов
arg = claripy.BVS('arg', 5 * 8)
# Передаем ее как аргумент (не забываем и argv[0])
state = p.factory.entry_state(args=['./argv', arg])
sm = p.factory.simulation_manager(state)
sm.explore(find=lambda s: b"Good job" in s.posix.dumps(1))
# Печатаем на экран решение для нашей переменной
print(sm.found[0].solver.eval(arg, cast_to=bytes))
```

```
b'+1337'
```

Angr. Добавляем условия

- Интересно, а может ли в качестве решения выступить отрицательное число?
  - Давайте выясним, добавив ограничение на строку ответа

```
import angr
import claripy
p = angr.Project('argv')
# Создаем переменную длиной в 16 символов
arg = claripy.BVS('arg', 16 * 8)
# Передаем ее как аргумент (не забываем и argv[0])
state = p.factory.entry_state(args=['./argv', arg])
# Добавляем условие: первый символ - минус
state.add_constraints(arg.get_byte(0) == b'-')
sm = p.factory.simulation_manager(state)
sm.explore(find=lambda s: b"Good job" in s.posix.dumps(1))
# Печатаем на экран решение для нашей переменной
print(sm.found[0].solver.eval(arg, cast_to=bytes))
```

```
b'-55834573511\x04\x00\x00\x00'
```

Angr. Добавляем условия на результат

- Иногда бывает, что нам нужно не просто попасть в определенную ветвь программы, а получить от нее определенный результат (в рамках одной ветви)
- Допустим, мы хотим получить на экране 1337 в следующем примере:

```
#include <stdio.h>
int function(int var)
{
    return var * var;
}
int main()
{
    int number;
    printf("Please enter some number: ");
    scanf("%d", &number);
    printf("Result: %d\n", function(number));
}
```

Angr. Добавляем условия на результат

```
import angr
import claripy
p = angr.Project('fun')
# Создаем аргумент
arg = claripy.BVS('arg', 64)
# Вызываем функцию function
state = p.factory.call_state(p.loader.find_symbol('function').rebased_addr, arg)
sm = p.factory.simulation_manager(state)
# Запускаем выполнение
sm.run()
# Получаем единственное конечное состояние
state = sm.deadended[0]
# Добавляем условие на вывод (результат в cdecl лежит в RAX)
state.add_constraints(state.regs.rax == 1337)
# Получаем решение для входа
print(state.solver.eval(arg))
```

```
3859546837
```

Angr. Добавляем условия на результат

- Важно помнить, что из-за ограничений реализации стандартной библиотеки в angr подобные условия можно добавить не всегда
  - Например нельзя добавлять условия на строки, созданные в результате работы printf
- Следующую задачу angr не решит:

```c
#include <stdio.h>
int main() {
    int number;
    printf("Please enter number: ");
    scanf("%d", &number);
    char tmps[1024];
    sprintf(tmps, "%d", number);
    if (!strcmp(tmps, "1337"))
        printf("Good job!\n");
    else
        printf("Wrong!\n");
}
```

Angr. Перехват функций

- Иногда может возникнуть необходимость заменить некоторую функцию своей реализацией
  - Как правило, такое случается в статических исполняемых файлах, где имена функций неизвестны (а сами реализации функций очень большие и не поддаются анализу)
  - Также необходимость может возникнуть, если реализация функции в angr вас не устраивает
- Осуществить замену можно при помощи следующих функций объекта Project:
  - hook(addr, hook, length) – позволяет заменить часть кода, начинающуюся в addr длины length на ваш обработчик, который вы передадите в hook (типа Hook, может быть SimProcedure)
    - Если передать length = 0 (по умолчанию), то можно просто посмотреть состояние из своей функции, не заменяя ни одну из инструкций, а просто дополняя их
  - hook_symbol(name, proc) – позволяет заменить функцию по имени name на вашу реализацию proc (может быть только SimProcedure)

Angr. Перехват функций

- Hook – функция, принимающая единственный аргумент state
  - И, возможно, как-то на него влияющая
- SimProcedure – более сложный в устройстве класс, именно его экземплярами являются все встроенные функции, реализованные в SimOS
  - Для наших целей наиболее полезной будет функция run(self, args), в которой можно написать свою реализацию перехватываемой функции, указав аргументы в объявлении функции и вернув из нее результат (очень удобно)
- Встроенные функции из SimOS доступны в массиве angr.SIM_PROCEDURES[библиотека][функция]
  - Например, angr.SIM_PROCEDURES['libc']['memcpy']()

Angr. Беды со scanf()

- Реализация scanf в angr является неполной, поэтому не поддерживает формат, который я использовал при написании большинства задач: "%1023[^\n]"
  - Этот формат позволяет считать строку до ближайшего переноса
- Также в angr отсутствовала реализация gets(), но позже появилась
  - Впрочем, все равно разберем, как ее можно реализовать при помощи хуков в качестве примера
- Поэтому по умолчанию применение angr к моим старым задачам дает неудовлетворительные результаты
  - Angr или падает или выдает ложный ответ
- Исправим это недоразумение, воспользовавшись функцией hook_symbol

```python
import angr, sys
p = angr.Project(sys.argv[1])
# Наша реализация gets()
class mygets(angr.SimProcedure):
    def run(self, str_addr):
        # Читаем из stdin (всего одну строку не более 128 байт)
        data, sz = self.state.posix.get_fd(0).read_data(128)
        # Записываем результат по указателю, переданному в gets (str_addr)
        self.state.memory.store(str_addr, data, size=sz)
        # Возвращаем результат (для gets - переданный указатель str_addr)
        return str_addr
# Наш scanf умеет только читать строки (как gets())
class myscanf(mygets):
    def run(self, fmt, str_addr):
        super().run(str_addr)
        return 1
# Устанавливаем перехват
p.hook_symbol('scanf', myscanf())
p.hook_symbol('__isoc99_scanf', myscanf())
p.hook_symbol('gets', mygets())
state = p.factory.entry_state()
sm = p.factory.simulation_manager(state)
sm.explore(find=lambda s: b"Correct" in s.posix.dumps(1))
print(sm.found[0].posix.dumps(0))
```

Атаки по сторонним каналам

Атаки по сторонним каналам

- Зачастую код является слишком унылым, чтобы пытаться его полноценно реверсить
  - Например, обфусцированным
- Ранее вы знакомились с некоторыми способами атак по сторонним каналам:
  - Поиск данных в оперативной памяти
  - Перехват функций сторонних библиотек
  - Оба эти способа, как можно заметить, не требуют модификации и прочтения кода самой программы
- Сегодня я расскажу вам про еще один способ подобной атаки – тщательное измерение количества выполненных инструкций
  - Этот способ может быть особенно удобен если программа осуществляет сравнение некоторой контролируемой вами строки с правильной и завершается сразу при обнаружении несоответствия

Временная атака

- Программа, которую мы будем атаковать:

```
#include <stdio.h>
int main()
{
    char string[1024];
    char * secret = "S3cr3t";
    printf("Please enter string: ");
    scanf("%s", string);
    for (int i=0;i<=strlen(secret);i++)
        if (string[i] != secret[i])
        {
            printf("Wrong!\n");
            return 1;
        }
    printf("Good job!\n");
}
```

Временная атака

- При подобном сравнении, соответственно, время выполнения будет тем больше, чем больше букв мы угадали
- Итоговая сложность атаки составит $O(K * N)$
  - Против $O(K^N)$ для полного перебора
  - Это вполне реалистичная сложность для небольших строк (вроде флагов или лицензионных ключей)

Intel Pin

- Для реализации такой атаки используется библиотека Intel Pin
- Intel Pin – инструмент для т.н. «динамической бинарной инструментации»
  - Увы, мы не будем изучать, что это, желающие могут почитать тут: https://habr.com/ru/company/dsec/blog/142575/
  - Если коротко, это механизм, позволяющий менять поведение исполняемого файла без его исходного кода и без ручной модификации бинарного кода
- Все использования этого инструмента в реверсе, которые я встречал, были связаны с подсчетом инструкций
  - Абсолютно точный подсчет числа инструкций – нетривиальная задача, и это средство является одним из немногих, позволяющих это делать
- Существуют и решения, основанные на Valgrind
  - Там инструмент callgrind позволяет также получить количество инструкций

Intel Pin. Готовые инструменты

- Pintool – https://github.com/wagiro/pintool
- PinCTF – https://github.com/ChrisTheCoolHut/PinCTF
  - У него больше возможностей, но на вид более запутанный интерфейс (даже какой-то файл конфигурации есть)
  - Зато есть скрипт для автоматической загрузки Pin и сборки нужного примера inscount0
  - Поддерживает многопоточность
- Pintool-n0n3m4 – https://n0n3m4.ru/nsuctf/n0n3m4_pintool_v2.tar.gz
  - Мой форк Pintool, не такой странный, как оригинал (но сделанный за час)
  - Имеет скрипт для загрузки и сборки Pin
  - Единственный из всего списка выключает ASLR, из-за чего, опять же, единственный рабочий на Ubuntu 18.04+ инструмент
  - Поддерживает и Valgrind / callgrind

Проводим временную атаку

- Качаем инструмент Pintool-n0n3m4 и распаковываем его где-нибудь на виртуальной машине с Ubuntu
  - Можно и в WSL 1, но там придется использовать Valgrind
- Запускаем ./getPin.sh, не забыв, конечно, ввести пароль для sudo
- Используем инструмент при помощи ./autosolve.py, имеющего следующие флаги:
  - -l – флаг, позволяющий указать длину пароля, до которой мы перебираем
  - -s – символ, которого наверняка нет во флаге (по умолчанию _)
  - -c – алфавит для перебора (6 – все символы)
  - -e – включает режим перебора длины пароля (на случай, если у вас в программе есть еще и сравнение на strlen)
  - -a – позволяет выбрать разрядность исполняемого файла (32 бита или 64)
  - -v – заменяет Pin на Valgrind / callgrind, для него разрядность можно не выбирать

Проводим временную атаку

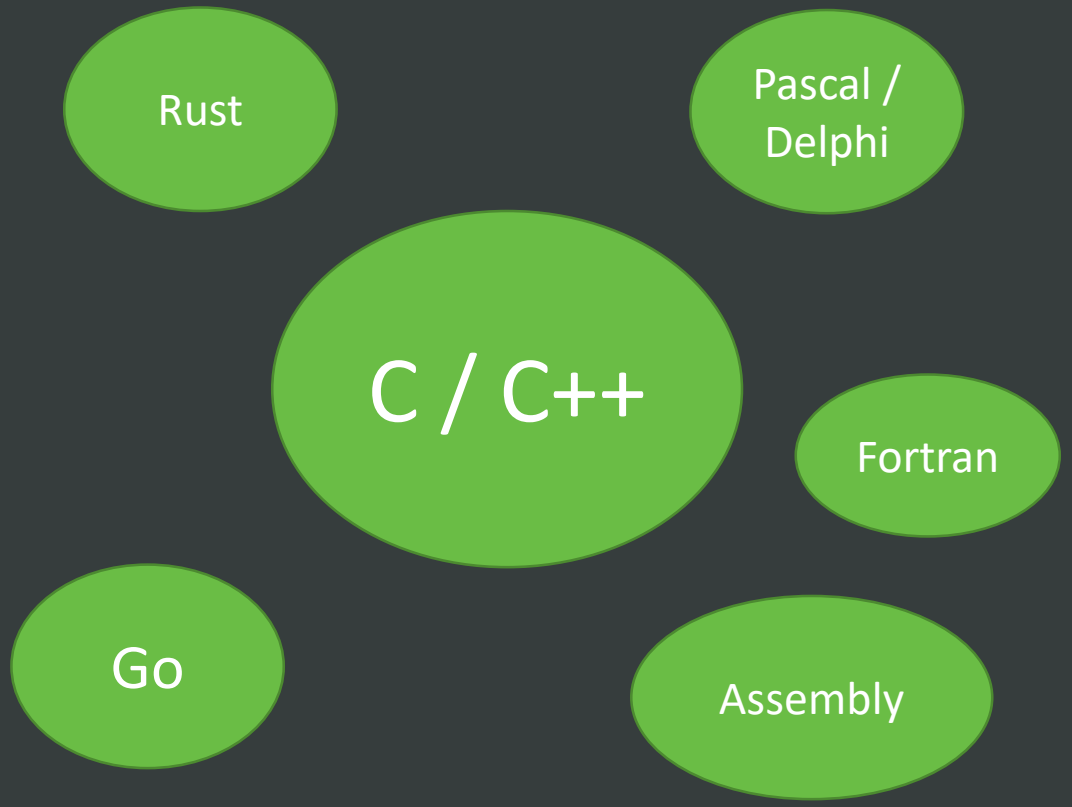
- Для нашего конкретного примера будем считать длину известной, добавив параметр -l 6 и используем полный алфавит, добавив параметр -c 6
- Запасаемся терпением и получаем следующий вывод:

```
n0n3m4@localhost:~$ ./autosolve.py -l 6 -c 6 ./timeattack
S_____ = 185049 difference 38 instructions
S3____ = 185087 difference 38 instructions
S3c___ = 185125 difference 38 instructions
S3cr__ = 185163 difference 38 instructions
S3cr3_ = 185201 difference 38 instructions
S3cr3t = 185264 difference 63 instructions
Solution:  S3cr3t
```

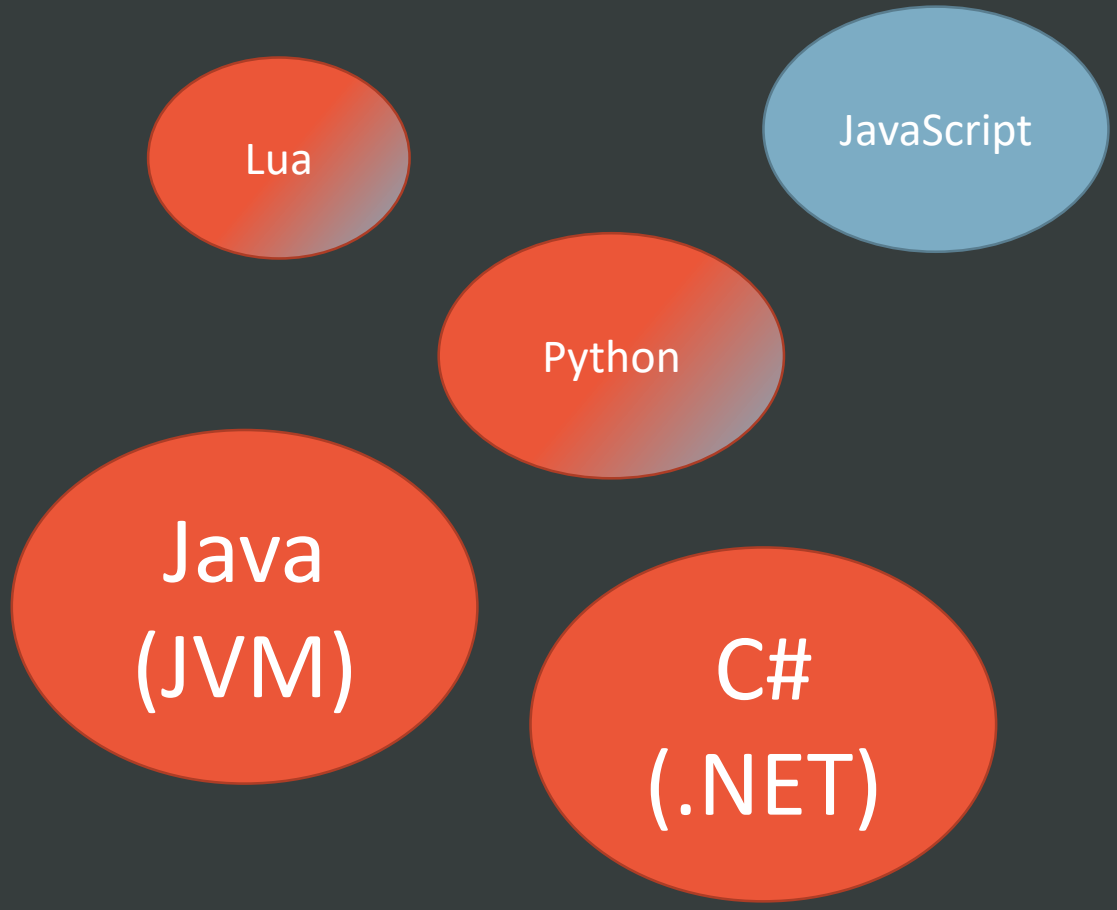
- Стоит ли говорить, что работоспособность атаки не зависит от запутанности кода
  - Правда, с movfuscator у меня почему-то не сработало, valgrind там тоже падает

Ненативные языки

Нативные языки



Языки с байткодом или интерпретируемые



Ненативные языки

- В случае с большинством ненативных языков обратную разработку осуществлять гораздо легче, поскольку в наличии декомпиляторы
  - И эти декомпиляторы в среднем работают гораздо лучше, чем декомпиляторы машинного кода
- В некоторых случаях полученный в результате декомпиляции код даже можно запустить / собрать без изменений
- На этой лекции мы рассмотрим инструменты для следующих языков / платформ:
  - Python
  - C# / .NET
  - Java / JVM
  - Java / Android

Python

- Python имеет байткод, который обычно хранится в формате .pyc
  - .pyc файлы можно запускать как обычные при помощи python file.pyc
  - Нагенерить таких файлов из исходников можно командой python -m compileall .
- Байткод Python зависит от версии
  - Байткод для Python 3.6 не запустится в Python 3.7, например
- Обычно байткод используется при сборке Python-программ при помощи инструментов наподобие pyinstaller или py2exe
  - Логично, ведь исходный код там уже не нужен

Декомпилируем Python

- uncompyle6 – декомпилятор, поддерживающий все версии Python начиная с первой
  - Установить его можно через pip командой "pip install uncompyle6" или из репозитория https://github.com/rocky/python-uncompyle6
  - Для декомпиляции достаточно использовать команду "uncompyle6 file.pyc"
  - У него есть более современная версия decompyle3: https://github.com/rocky/python-decompile3
  - Не поддерживает Python 3.9+ (на момент написания), автор собирал деньги на разработку https://github.com/rocky/python-decompile3/issues/45, но вроде как разработка и так ведется в настоящее время (уже 3+ года как)
- Decompyle++ – другой популярный декомпилятор, поддерживает Python 3.9+
  - Можно установить через snap (не работает в WSL1) командой "sudo snap install pycdc" или собрать из исходников https://github.com/zrax/pycdc
  - Используется аналогично: "pycdc file.pyc"

Декомпилируем Python

- Оригинал:

```
string = input("Hello, please enter key: ")
if string == "S3cr3t":
        print("Correct key, congratulations!")
else:
        print("Wrong key")
```

- Вывод uncompyle6:

```
string = input('Hello, please enter key: ')
if string == 'S3cr3t':
    print('Correct key, congratulations!')
else:
    print('Wrong key')
```

Декомпилируем Python

- В самых тяжелых случаях можно использовать Python-дизассемблеры
- Обычно они ходят парой с декомпиляторами
  - pydisasm (и библиотека xdis) для uncompyle6 (кстати, в отличие от декомпилятора, дизассемблер поддерживает Python 3.9+)
  - pycdas для Decompyle++
- Используются аналогично декомпиляторам, просто выводят байткод

```
1:            0 LOAD_NAME                0 (input)
              2 LOAD_CONST               0 ('Hello, please enter key: ')
              4 CALL_FUNCTION            1
              6 STORE_NAME               1 (string)

2:            8 LOAD_NAME                1 (string)
             10 LOAD_CONST               1 ('S3cr3t')
             12 COMPARE_OP               2 (==)
             14 POP_JUMP_IF_FALSE       26 (to 26)

3:           16 LOAD_NAME                2 (print)
             18 LOAD_CONST               2 ('Correct key, congratulations!')
             20 CALL_FUNCTION            1
             22 POP_TOP
             24 JUMP_FORWARD             8 (to 34)

5:       >>  26 LOAD_NAME                2 (print)
             28 LOAD_CONST               3 ('Wrong key')
             30 CALL_FUNCTION            1
             32 POP_TOP
         >>  34 LOAD_CONST               4 (None)
             36 RETURN_VALUE
```

Время задач

CrackMe.py

Категория: Lesson 20 / Angr + Pin + Non-native

Решивших: 0

Время: 00:00:02

- Доступ к задачам можно получить как всегда на nsuctf.ru
- В этой задаче вам может пригодиться uncompyle6 и pycdc

.NET

- .NET Framework – программная платформа, созданная Microsoft
- Применяется в следующих областях:
  - Очевидно, программы для Windows
  - Программы на Mono / Xamarin для других операционных систем (в т.ч. мобильных)
  - Игровой движок Unity
- Включает следующие языки:
  - C# (как правило, основной язык, на котором пишут под .NET)
  - Visual Basic
  - F# и всякое другое
- Узнать можно по тому, что несмотря на расширение .exe / .dll, IDA такие файлы практически не берет (и предупреждает, что в файле .NET-код)

.NET декомпиляторы

- С декомпиляцией .NET все обычно очень хорошо
  - Мне лично приходилось во времена Windows Mobile успешно пересобирать достаточно внушительную программу из декомпилированного исходного кода
- Среди наиболее известных декомпиляторов присутствуют следующие:
  - dnSpy (https://github.com/0xd4d/dnSpy) – форк проекта ILSpy (https://github.com/icsharpcode/ILSpy), наиболее примечательной функцией является поддержка отладки и модификации .NET байткода
    - Увы, на текущий момент не совсем жив, но есть форк https://github.com/dnSpyEx/dnSpy
  - dotPeek (https://www.jetbrains.com/decompiler/) – декомпилятор от JetBrains, иногда имеет более вменяемый код, чем dnSpy
  - .NET Reflector (https://www.red-gate.com/products/dotnet-development/reflector/) – ранее популярный коммерческий декомпилятор .NET, ныне практически забыт
- При желании также можно почитать байткод

.NET декомпиляторы

```
static void Main(string[] args) {
    Console.Write("Hello, please enter key: ");
    string s = Console.ReadLine();
    if (s == "S3cr3t")
        Console.WriteLine("Correct key, congratulations!");
    else
        Console.WriteLine("Wrong key");
}
```

```
private static void Main(string[] args) {
    Console.Write("Hello, please enter key: ");
    if (Console.ReadLine() == "S3cr3t")
      Console.WriteLine("Correct key, congratulations!");
    else
      Console.WriteLine("Wrong key");
}
```

```
IL_0000: nop
IL_0001: ldstr     "Hello, please enter key: "
IL_0006: call      void [netstandard]System.Console::Write(string)
IL_000B: nop
IL_000C: call      string [netstandard]System.Console::ReadLine()
IL_0011: stloc.0
IL_0012: ldloc.0
IL_0013: ldstr     "S3cr3t"
IL_0018: call      bool [netstandard]System.String::op_Equality(string, string)
IL_001D: stloc.1
IL_001E: ldloc.1
IL_001F: brfalse.s IL_002E
IL_0021: ldstr     "Correct key, congratulations!"
IL_0026: call      void [netstandard]System.Console::WriteLine(string)
IL_002B: nop
IL_002C: br.s      IL_0039
IL_002E: ldstr     "Wrong key"
IL_0033: call      void [netstandard]System.Console::WriteLine(string)
IL_0038: nop
IL_0039: ret
```

Время задач

CrackMe.NET

Категория: Lesson 20 / Angr + Pin + Non-native

Решивших: 0

Время: 00:00:12

- Доступ к задачам можно получить как всегда на nsuctf.ru
- В этой задаче вам может пригодиться dnSpy или ILSpy: https://github.com/0xd4d/dnSpy (https://github.com/dnSpyEx/dnSpy) и https://github.com/icsharpcode/ILSpy

Java

- Java – язык, созданный компанией Sun (ныне Oracle)
  - Значительной частью Java является JVM – виртуальная машина Java, на которой выполняется байткод Java
- Применяется (и применялся) в огромном количестве областей, например:
  - Кросс-платформенная разработка
  - Разработка для ОС Android
- JVM исполняет множество языков, например:
  - Собственно, Java
  - Kotlin – язык, созданный JetBrains, популярен при разработке для Android
  - Clojure – Lisp для JVM, Jython – Python для JVM
- Можно распознать по характерному расширению .jar и .class

Java декомпиляторы

- Среди популярных декомпиляторов можно отметить:
  - JD-GUI (https://github.com/java-decompiler/jd-gui) – старый, но простой и удобный, с GUI
  - CFR (https://github.com/leibnitz27/cfr) – с поддержкой новых версий Java, работает из командной строки
  - Fernflower (https://github.com/fesh0r/fernflower) – от JetBrains, входит в состав IntelliJ IDEA, работает из командной строки
  - Bytecode Viewer (https://bytecodeviewer.com/) – графический интерфейс к множеству различных декомпиляторов, имеет редактор
  - ReCaf (https://github.com/Col-E/Recaf) – редактор JVM-байткода
  - Важно помнить, что из-за большого зоопарка языков, с декомпиляцией JVM байткода других языков кроме Java в общем случае все не очень хорошо
- Осторожно, грабли: чтобы запускать .jar-файлы для командной строки из Windows, нужно использовать java -jar jarfile.jar: иначе вывод не будет показан
  - Это можно исправить в реестре в HKCR\jarfile\shell\open\command, убрав "w" у "javaw"

```java
public static void main(String[] args) throws Exception {
    System.out.println("Hello, please enter key: ");
    BufferedReader br = new BufferedReader(new
InputStreamReader(System.in));
    String s = br.readLine();
    if ("S3cr3t".equals(s))
        System.out.println("Correct key, congratulations!");
    else
        System.out.println("Wrong key");
}
```

```java
public static void main(String[] arrstring) throws Exception {
    System.out.println("Hello, please enter key: ");
    BufferedReader bufferedReader = new BufferedReader(new
InputStreamReader(System.in));
    String string = bufferedReader.readLine();
    if ("S3cr3t".equals(string)) {
        System.out.println("Correct key, congratulations!");
    } else {
        System.out.println("Wrong key");
    }
}
```

```
public static void main(java.lang.String[]) throws java.lang.Exception;
Code:
  0: getstatic     #2   // Field java/lang/System.out:Ljava/io/PrintStream;
  3: ldc           #3   // String Hello, please enter key:
  5: invokevirtual #4   // Method java/io/PrintStream.println:(Ljava/lang/String;)V
  8: new           #5   // class java/io/BufferedReader
  11: dup
  12: new           #6  // class java/io/InputStreamReader
  15: dup
  16: getstatic     #7  // Field java/lang/System.in:Ljava/io/InputStream;
  19: invokespecial #8  // Method java/io/InputStreamReader."<init>":(Ljava/io/InputStream;)V
  22: invokespecial #9  // Method java/io/BufferedReader."<init>":(Ljava/io/Reader;)V
  25: astore_1
  26: aload_1
  27: invokevirtual #10 // Method java/io/BufferedReader.readLine:()Ljava/lang/String;
  30: astore_2
  31: ldc           #11 // String S3cr3t
  33: aload_2
  34: invokevirtual #12 // Method java/lang/String.equals:(Ljava/lang/Object;)Z
  37: ifeq          51
  40: getstatic     #2  // Field java/lang/System.out:Ljava/io/PrintStream;
  43: ldc           #13 // String Correct key, congratulations!
  45: invokevirtual #4  // Method java/io/PrintStream.println:(Ljava/lang/String;)V
  48: goto          59
  51: getstatic     #2  // Field java/lang/System.out:Ljava/io/PrintStream;
  54: ldc           #14 // String Wrong key
  56: invokevirtual #4  // Method java/io/PrintStream.println:(Ljava/lang/String;)V
```

Java для Android

- Java на Android несколько отличается от обычной – байткод подвергается дополнительному преобразованию для виртуальной машины Dalvik
  - Поэтому вместо .jar и .class файлов на Android код хранится в файлах .dex
- Сами .dex-файлы хранятся, в свою очередь, внутри .apk-файлов, где могут также храниться:
  - Ресурсы, необходимые для работы программы
  - Библиотеки с нативным кодом, собранные при помощи Android NDK
  - Кстати, сам .apk-файл – ни что иное как zip-архив (как, впрочем, и .jar файлы), поэтому его можно открыть любым архиватором, например 7-Zip
- В рамках этой презентации мы рассмотрим только реверсить Java-код

Java для Android

- На текущий момент для обратной разработки обычно используют специализированные декомпиляторы для Android
  - Например, JADX (https://github.com/skylot/jadx/)
  - Также хорошо подойдет JEB (но он платный, поэтому не стоит искать его на https://down.52pojie.cn/, раньше не стоило на forum.reverse4you.org, но сейчас его там почему-то нет)
- Можно использовать и декомпиляторы из мира Java, для этого нужно воспользоваться инструментом dex2jar (https://github.com/pxb1988/dex2jar), чтобы преобразовать .dex в .jar
  - Для того чтобы получить из .dex-файлов .jar-файл можно просто перетащить ваш APK-файл на скрипт d2j-dex2jar.bat из его состава
  - Кстати, Bytecode Viewer тоже умеет работать с APK-файлами
- Для модификации APK-файлов лучше использовать apktool и работать на уровне байткода Dalvik

Время задач

CrackMe Mobile

Категория: Lesson 20 / Angr + Pin + Non-native

Решивших: 0

Время: 00:00:01

- Доступ к задачам можно получить как всегда на nsuctf.ru
- В этой задаче вам может пригодиться JADX или Bytecode Viewer: https://github.com/skylot/jadx/ и https://bytecodeviewer.com/

Спасибо за внимание!
Задачи доступны на

nsuctf.ru

- Пожалуйста, используйте имя пользователя формата "Фамилия Имя"
  - e-mail можно забить любой, сервером он не проверяется
- Для вопросов по задачам рекомендую присоединиться к @NSUCTF в Telegram
  - Только, пожалуйста, без спойлеров