Cybersecurity. SAST+DAST

Кибербезопасность. Комплексный анализ ПО.

к.ф.-м.н. Курмангалеев Шамиль

31 мая 2023

ИСП РАН

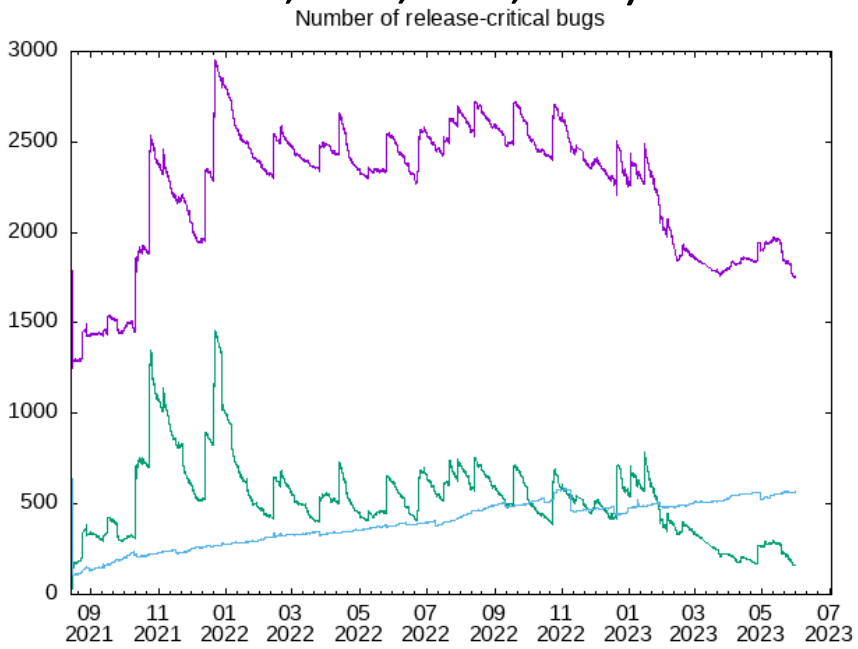
Введение

ИСП РАН

- Ошибки могут проявляться на всех этапах жизненного цикла ПО.
- Те ошибки, что не нашлись при разработке потенциально могут найтись во время эксплуатации ПО.
- Объемы ПО (Debian BookWorm 1,340,877,737 -> SID 1,557,455,011)

Release-critical bugs status*
Wed May 31 00:00:00 UTC 2023
Total number of release-critical bugs: 1758
Number that have a patch: 198
Number that have a fix prepared and waiting to upload: 22
Number that are being ignored: 109
Number concerning the current stable release: 568
Number concerning the next release: 164
Number concerning the previous stable release: 43

*https://bugs.debian.org/release-critical/



Введение

ИСП РАН

- **Ошибки в ПО есть, а «серебряной пули» нет ☹.**

- Как уменьшить количество ошибок в ПО?
- Как понизить опасность ошибок, о которых разработчик не знал и не исправил?
- Какие ошибки исправлять в первую очередь?
- Жизненный цикл разработки доверенного ПО и технологии
  - **Компилятор**
  - **SAST** (Static Analysis Security Testing)
  - **DAST** (Dynamic Analysis Security Testing)
  - Среда выполнения

- Возможности и вызовы при разработке инструментов

Жизненный цикл разработки доверенного ПО

ИСП РАН

- ГОСТ Р 56939-2016 «Защита информации. Разработка безопасного программного обеспечения. Общие требования»
- Microsoft Security Development Lifecycle

Обучение	Требования	Проектирование	Реализация	Проверка	Выпуск	Реагирование
Обучение основам безопасности	Задание требований безопасности Создание контрольных условий качества и панелей ошибок Оценка рисков безопасности и конфиденциальности	Задание требований проектирования Анализ возможных направлений для злоумышленных атак Моделирование рисков	Применение утвержденных инструментов Отказ от небезопасных функций Статический анализ	Динамический анализ Нечеткое тестирование Проверка возможных направлений для злоумышленных атак	Планирование реагирования на инциденты Окончательная проверка безопасности Архивация выпуска	Выполнение плана реагирования на инциденты

Технологии

ИСП РАН

- Компилятор
  - Контроль кода на потенциальные ошибки
    - Легковесный статический анализ и Warnings от компилятора
    - «Безопасные» варианты функций (strncpy…)
  - Встраивание защитных механизмов
    - Проверка времени выполнения FORTIFFY_SOURCE
    - Контроль целостности кадра стека, потока управления…
    - Санитайзеры, в т.ч. оповещение о «неопределенном» поведении
    - Диверсификация кода
    - Обфускация (запутывание)
  - Инструментация для снятия покрытия ПО, встраивание эвристик для фаззера и динамики.

Технологии

ИСП РАН

- Статический анализ
  - Анализ исходного кода, поиск ошибок
  - Управление известными ошибками
  - Поиск первоисточников для заимствованных компонентов в т.ч. в виде бинарного кода
  - Поиск клонов кода
    - Семантические клоны
    - Масштабируемый поиск для дистрибутивов с открытым кодом
  - LOLBas, GTFOBins – Использование стандартных программ для выполнения действий по обходу ограничений политик безопасности
  - Анализ бинарного кода

Технологии

ИСП РАН

- Динамический анализ
  - Фаззинг
    - Классические ошибки (аварийные завершения)
    - Нежелательное поведение (напр. запуск программ)
    - Какой выбрать, как настроить?
      - Дифференциальный фаззинг
    - Типы инструментации
      - Исходный код, бинарный код
  - Поиск ошибок
    - Санитайзеры
    - Valgrind
    - BoundsChecker

Технологии

ИСП РАН

- Динамический анализ
  - Символьное выполнение
    - Полносистемное
    - Режима пользователя
  - Анализ помеченных данных
    - Оценка поверхности атаки
  - Фаззинг обертки
    - Новые unit-тесты
      - Выбор целей
      - Автоматизация

Технологии

ИСП РАН

- Среда выполнения
  - Средства защиты
    - Предотвращение эксплуатации уязвимостей ASRL,DEP, SEHOP …
    - Политики безопасности AppArmor, SELinux, Linux Capabilities…
    - Цепочка доверенной загрузки
    - Доверенное хранилище

Технологии, SAST+DAST

ИСП РАН

- Статический анализ
  - Мы видим всю программу
- Динамический анализ
  - Видим обработку конкретных данных

Компиляторные технологии

Компиляторные технологии

- Базовые понятия
  - Граф потока управления программы
  - Граф вызовов
  - Граф потока данных
  - Передаточные функции

Технологии, SAST+DAST

ИСП РАН

Умное объединение технологий дает ключ к решению или снижению сложности некоторых проблем.

- Статический анализ
  - Значения указателей
    - Данные
    - Код, в т.ч. Виртуальные вызовы
  - Контекст вызова функции
  - Видим все пути одновременно, как отсеять несовместные, а среди совместных найти содержащие ошибку
  - Большое количество предупреждений, для многих не ясно на каких данных можно дойти до начала трассы ошибки
  - Понимаем структуру программы, но часто без внешней подсказки не хватает точности в определении смысловой нагрузки
- Динамический анализ
  - Видим один путь за один раз
  - Имеем конкретные значения переменных и указателей, но не их множество
  - Ищем новые пути наугад, идем куда получилось, а не куда надо

Многообразие ПО - от UEFI до браузера

ИСП РАН

- Прошивки
- Ядро и драйвера
- Пользовательское ПО
- Различные архитектуры

- Сетевые протоколы
- Парсеры
- Интерпретаторы и JIT компиляторы

- Нейросети

Общие направления исследований по фаззингу

ИСП РАН

- Эмуляция
  - Полносистемная
  - Частичная
  - Использование гипервизоров
- Распределенный, долгосрочный фаззинг
  - Дифференциальный фаззинг
    - Типы инструментов и вариантов ПО
    - Планирование ресурсов

Общие направления исследований по фаззингу

ИСП РАН

- Алгоритмы мутации и отбора данных
  - Мы хотим увеличивать покрытие за меньшее количество запусков ПО
- Формат данных
- Анализ влияния данных на открытие новых путей в программе, «граничные» переходы
- Автоматизация генерации фаззинг тестов
- Сбор и объединение информации от динамики и статики
  - Автоматизированная разметка предупреждений
- Выявление недетерминизма
  - Иногда большие программы ведут себя по разному при запуске на одних данных
    - Причина – напр. Неициализированные переменные, зависимость от таймера, неверное предположение про контейнеры (порядок извлечения элементов)

Программы с состояниями

ИСП РАН

Программа с состояниями моделируется ДКА.

Главный вопрос – покрытие кода по строкам не соответствует покрытию по состояниям.
Программист задумывал последовательность malloc, use, free; но возможны и другие перестановки, с эквивалентным покрытием по строкам кода

- Поддержка состояний программы в фаззере
  - Ручная инструментация кода
  - Автоматическая в компиляторе
  - Автоматическая
    - Статический анализ бинарного кода+инструментатор
  - Учет состояний при планировании и мутациях
- Статический анализ дает представление о коде, динамический анализ о пути в ДКА в ответ на конкретные данные

Статический анализ

ИСП РАН

Svacer | Project: v1.21.4-local | Branch: master | Review | Browse code | Reports | Settings | Ru En | admin

Snapshot: Snapshot 2022-11-29 14:40:44 +0300 | Compare with:

Checkers | Files

- BAD_COPY_PASTE (10)
- DEREF_AFTER_FREE (12)
- DEREF_AFTER_NULL.EX (2)
  - ngx_http_upstream.c : 4521
  - ngx_http_upstream.c : 4521
- DEREF_OF_NULL.COND (4)
- DEREF_OF_NULL.EX (10)
- DEREF_OF_NULL.EX.COND (4)
- DEREF_OF_NULL.RET.LIB.PROC (2)
- DEREF_OF_NULL.RET.STAT (1)
  - ngx_http_core_module.c : 3426
- DEREF_OF_NULL.STRICT (2)
- DOUBLE_FREE.EX (12)
- FORK_PROCESSES_MEET (2)
- HANDLE_LEAK (2)
- INTEGER_OVERFLOW (16)
- LOCAL_VAR.HUGE (2)
- MEMORY_LEAK (2)
- MEMSET_SIZE_MISMATCH (2)
- NEGATIVE_CODE_ERROR.EX (4)
- NO_CAST.INTEGER_OVERFLOW (84)
- NONTERMINATED_STRING (2)
- NONTERMINATED_STRING.STYLE (4)
- NULL_AFTER_DEREF (8)
- PROC_USE.RAND (14)
- PROC_USE.VULNERABLE.GETENV (2)
- SIGN_EXTENSION (4)
- UNCHECKED_FUNC_RES.LIB.STRICT (12)
- UNCHECKED_FUNC_RES.STAT (3)
- UNREACHABLE_CODE (14)
- UNREACHABLE_CODE.RET (12)
- UNREACHABLE_CODE.SWITCH (2)
- UNUSED_VALUE.PARAM_ASSIGN (6)

Filters | Snapshot information | Source code | Show table

/.build/target/clean_build/nginx-1.21.4/src/http/ngx_http_upstream.c

```
6536    ngx_array_t                 headers_in;
6537    ngx_hash_key_t              *hk;
6538    ngx_hash_init_t             hash;
6539    ngx_http_upstream_init_pt   init;
6540    ngx_http_upstream_header_t  *header;
6541    ngx_http_upstream_srv_conf_t  **uscfp;
6542
6543    uscfp = umcf->upstreams.elts;
6544
6545    for (i = 0; i < umcf->upstreams.nelts; i++) {
6546
6547        init = uscfp[i]->peer.init_upstream ? uscfp[i]->peer.init_upstream:
6548                                              ngx_http_upstream_init_round_robin;
6549
6550        if (init(cf, uscfp[i]) != NGX_OK) {
6551            return NGX_CONF_ERROR;
6552        }
6553    }
6554
6555
6556    /* upstream_headers_in_hash */
6557
6558    if (ngx_array_init(&headers_in, cf->temp_pool, 32, sizeof(ngx_hash_key_t))
6559        != NGX_OK)
6560    {
6561        return NGX_CONF_ERROR;
6562    }
6563
6564    for (header = ngx_http_upstream_headers_in; header->name.len; header++) {
        TRACE 3.1  counter-example: ngx_array_push
6565        hk = ngx_array_push(&headers_in);
        TRACE 3.2  Check for NULL
6566        if (hk == NULL) {
6567            return NGX_CONF_ERROR;
6568        }
6569
6570        hk->key = header->name;
6571        hk->key_hash = ngx_hash_key_lc(header->name.data, header->name.len);
6572        hk->value = header;
6573    }
6574
6575    hash.hash = &umcf->headers_in_hash;
6576    hash.key = ngx_hash_key_lc;
6577    hash.max_size = 512;
6578    hash.bucket_size = ngx_align(64, ngx_cacheline_size);
6579    hash.name = "upstream_headers_in_hash";
6580    hash.pool = cf->pool;
6581    hash.temp_pool = NULL;
6582
6583    if (ngx_hash_init(&hash, headers_in.elts, headers_in.nelts) != NGX_OK) {
6584        return NGX_CONF_ERROR;
6585    }
6586
6587    return NGX_CONF_OK;
6588 }
6589
```

Undecided | Unspecified | Undecided

DEREF_OF_NULL.RET.STAT | ngx_http_core_module.c:3426

Trace | Comments | Details

1. Role: dereference
   1. Dereference
      - ngx_http_core_module.c:[3426:19]
   2. ngx_array_push
      - ngx_http_core_module.c:[3424:14]
2. Role: other example
   1. Dereference
      - ngx_http_core_module.c:[3426:19]
   2. ngx_array_push
      - ngx_http_core_module.c:[3424:14]
3. Role: counter-example
   1. counter-example: ngx_array_push
      - ngx_http_upstream.c:[6565:14]
   2. Check for NULL
      - ngx_http_upstream.c:[6566:13]
4. Role: counter-example
   1. counter-example: ngx_array_push
      - ngx_http_upstream.c:[6565:14]
   2. Check for NULL
      - ngx_http_upstream.c:[6566:13]
5. Role: counter-example
   1. counter-example: ngx_array_push
      - ngx_http_upstream.c:[6449:18]
   2. Check for NULL
      - ngx_http_upstream.c:[6450:17]
6. Role: counter-example
   1. counter-example: ngx_array_push
      - ngx_http_upstream.c:[6449:18]
   2. Check for NULL
      - ngx_http_upstream.c:[6450:17]
7. Role: counter-example
   1. counter-example: ngx_array_push
      - ngx_http_upstream.c:[6425:14]
   2. Check for NULL
      - ngx_http_upstream.c:[6426:13]
8. Role: counter-example
   1. counter-example: ngx_array_push
      - ngx_http_upstream.c:[6425:14]
   2. Check for NULL

Интеграция со статическим анализом

ИСП РАН

- DEREF_OF_NULL.RET.STAT (1)
  - ngx_http_core_module.c : 3426
- DEREF_OF_NULL.STRICT (2)

```
6564    for (header = ngx_http_upstream_headers_in; header->
        TRACE 3.1  counter-example: ngx_array_push
6565        hk = ngx_array_push(&headers_in);
        TRACE 3.2  Check for NULL
6566        if (hk == NULL) {
6567            return NGX_CONF_ERROR;
6568        }
```

3. Role: counter-example
   1. counter-example: ngx_array_push
      - ngx_http_upstream.c:[6565:14]
   2. Check for NULL
      - ngx_http_upstream.c:[6566:13]

3670. fuzz step [MATCHED]
- ngx_http_upstream.c:[6565:0]

3671. fuzz step [MATCHED]
- ngx_http_upstream.c:[6566:0]

3672. fuzz step
- ngx_http_upstream.c:[6558:0]

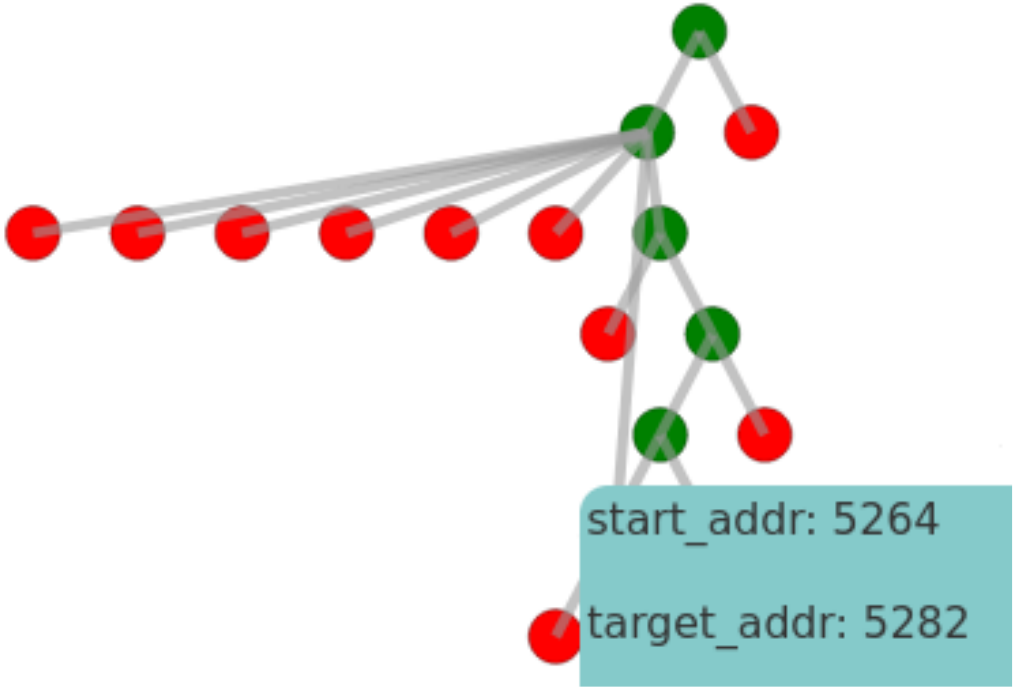
3673. fuzz step

Интеграция со статическим анализом

ИСП РАН

- Возможность улучшить статический анализ, путем непрерывного взаимодействия с фаззером и инструментами динамического анализа

```
27        }
28        if (data[3] == 'h') {
29          i++;
30        } else if (data[3] == 'j') {
31          i++;
32        }
```



EDGE: 5264 -> 5282

```
61 61 61 68 61 61 61 61 61 61 61 61 61 61 61 61 61 61 61 61 61 61 61 61    aaahaaaaaaaaaaaaa
```

Анализ произвольного кода

ИСП РАН

Код

```
......
4011a9: 55                push   rbp
4011aa: 48 89 e5          mov    rbp,rsp
4011ad: bf 60 40 40 00    mov    edi,0x404060 <input>
4011b2: e8 92 ff ff ff    call   401149 <read_data>
4011b7: c6 45 f0 05       mov    byte[rbp-0x10],0x5
4011bb: e8 77 ff ff ff    call   401137 <syscallA>
4011c0: bf 60 40 40 00    mov    edi,0x404060 <input>
4011c5: e8 74 ff ff ff    call   40113e <functionF>
4011ca: 31 c0             xor    eax,eax
4011cc: 31 c9             xor    ecx,ecx
......
```

Данные

```
........
04000000:   FF FF FF FF  EE EE EE EE  |........|
04000008:   4D 45 53 53  41 47 45 00  |MESSAGE.|
04000010:   FF FF 54 45  58 54 00 FF  |..TEXT..|
04000018:   80 00 30 00  40 00 80 00  |..0.@...|
04000020:   AA 47 45 54  20 2F 0A 00  |.GET /..|
04000028:   48 45 4C 4C  4F 21 00 00  |HELLO!..|
04000030:   04 05 00 00  04 05 CA FE  |........|
........
```

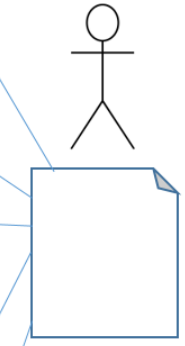
Начальное состояние
def load_dump(): …

Входные данные
[input] := “…”

Символьные данные
mark_symbolic(input)

Неэмулируемые фрагменты
def my_syscallA():
EAX := 1 …

Конечные точки
STOP_ADDR = 0x4011CA



Скрипт эмуляции

Конкретное выполнение

обнаружение ошибок

сбор информации

легковесный анализ

Символьное выполнение

генерация новых данных

символьный анализ

Частичная Эмуляция

Пример автоматического создания Фаззинг-обертки

ИСП РАН

```
int optval = 1;
setsockopt(parentfd, SOL_SOCKET, SO_REUSEADDR, (const void *)&optval , sizeof(int));

bzero((char *) &serveraddr, sizeof(serveraddr));

serveraddr.sin_family = AF_INET;
serveraddr.sin_addr.s_addr = htonl(INADDR_ANY);
serveraddr.sin_port = htons((unsigned short)portno);

bind(parentfd, (struct sockaddr *) &serveraddr, sizeof(serveraddr));
listen(parentfd, 5);

unsigned int clientlen = sizeof(clientaddr);
int childfd = accept(parentfd, (struct sockaddr *) &clientaddr, &clientlen);

bzero(buf, BUFSIZE);
int n = read(childfd, buf, BUFSIZE);

printf("server received %d bytes: %s", n, buf);

close(childfd);
```

Пример автоматического создания Фаззинг-обертки

ИСП РАН

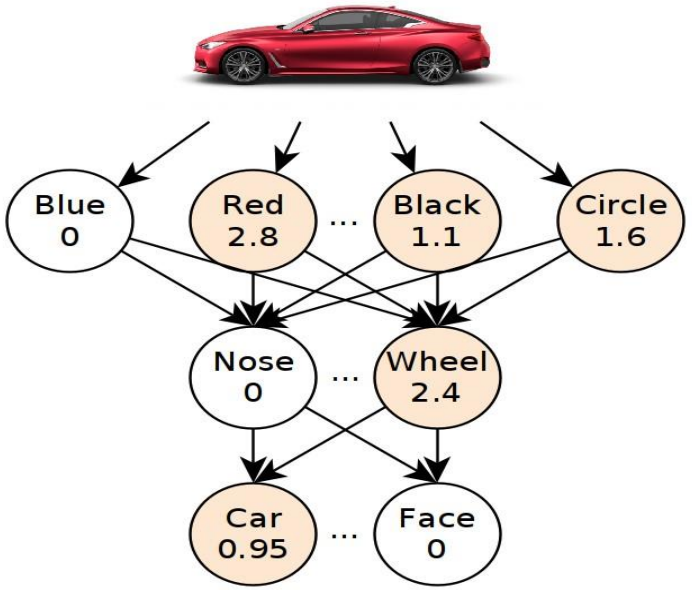
```
if __name__ == "__main__":
    IO_PATH = './io'

    args = read_path_arguments()
    instrumentation = QilingInstrumentation(args.input_path, [ql.os.exit_point], args.lighthouse)

    ql = Qiling(["./bin/main", "8080"], ".", verbose=QL_VERBOSE.OFF)
    io_files = get_io(IO_PATH)

    make_io(ql, IO_PATH, io_files)

    ql.os.set_syscall('accept', accept, QL_INTERCEPT.CALL)
    ql.os.set_syscall('listen', sock_check, QL_INTERCEPT.ENTER)

    ql.os.set_syscall('read', buf_patcher, QL_INTERCEPT.ENTER)
    ql.os.set_syscall('read', exiter, QL_INTERCEPT.EXIT)
    ql.os.set_syscall('write', buf_patcher, QL_INTERCEPT.ENTER)
    ql.os.set_syscall('write', exiter, QL_INTERCEPT.EXIT)

    instrumentation.run(ql)
```

Фаззинг нейронных сетей

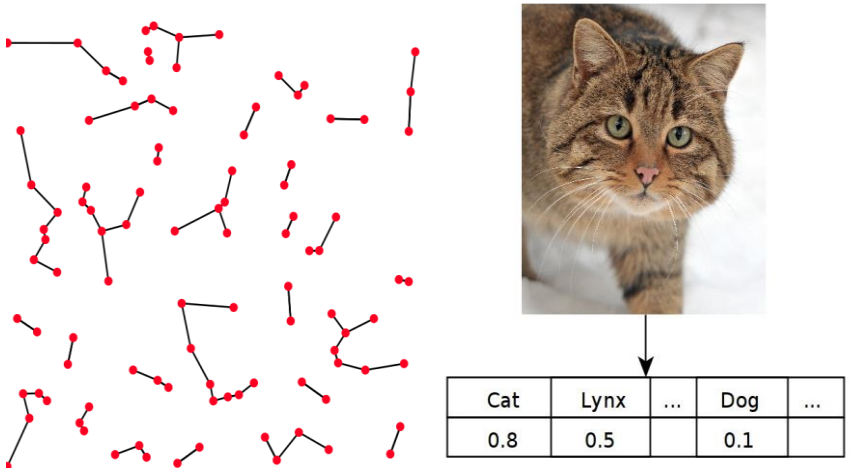
ИСП РАН

Подходы

- Покрытие - значения выходного слоя
- Покрытие - карта активации нейронов



Значения выходного слоя представляются в виде векторов, рассматриваются расстояния между ними.



Фаззинг нейронных сетей

ИСП РАН

- Сети
  - Yolo 8
  - FaceNet
  - Resnet
- Датасет
  - ImageNet
- Мутации
  - Черные квадраты
  - Изменение
    - Температуры, яркости и контраста

- Будущие направление – учет “attention” модели и систем обработки на основе нескольких НС