ИСП РАН

Технологии поиска ошибок в бинарном коде и защиты от эксплуатации уязвимостей

Курмангалеев Шамиль
к.ф.-м.н., с.н.с.
Отдел компиляторных технологий
ИСП РАН, Москва
2019

Технологии безопасной разработки ПО

- Экспертиза
  - Задействуются опыт и знания людей
- Дедуктивный анализ (верификация модели)
- Статический анализ
  - Поиск возможных дефектов в коде по некоторым шаблонам
- Динамический анализ
  - Тестирование, мониторинг, профилирование, фаззинг, …
- Смешанные техники
- Обфускация (защита от повторного взлома, от обратной инженерии ПО)

Качество кода по Coverity

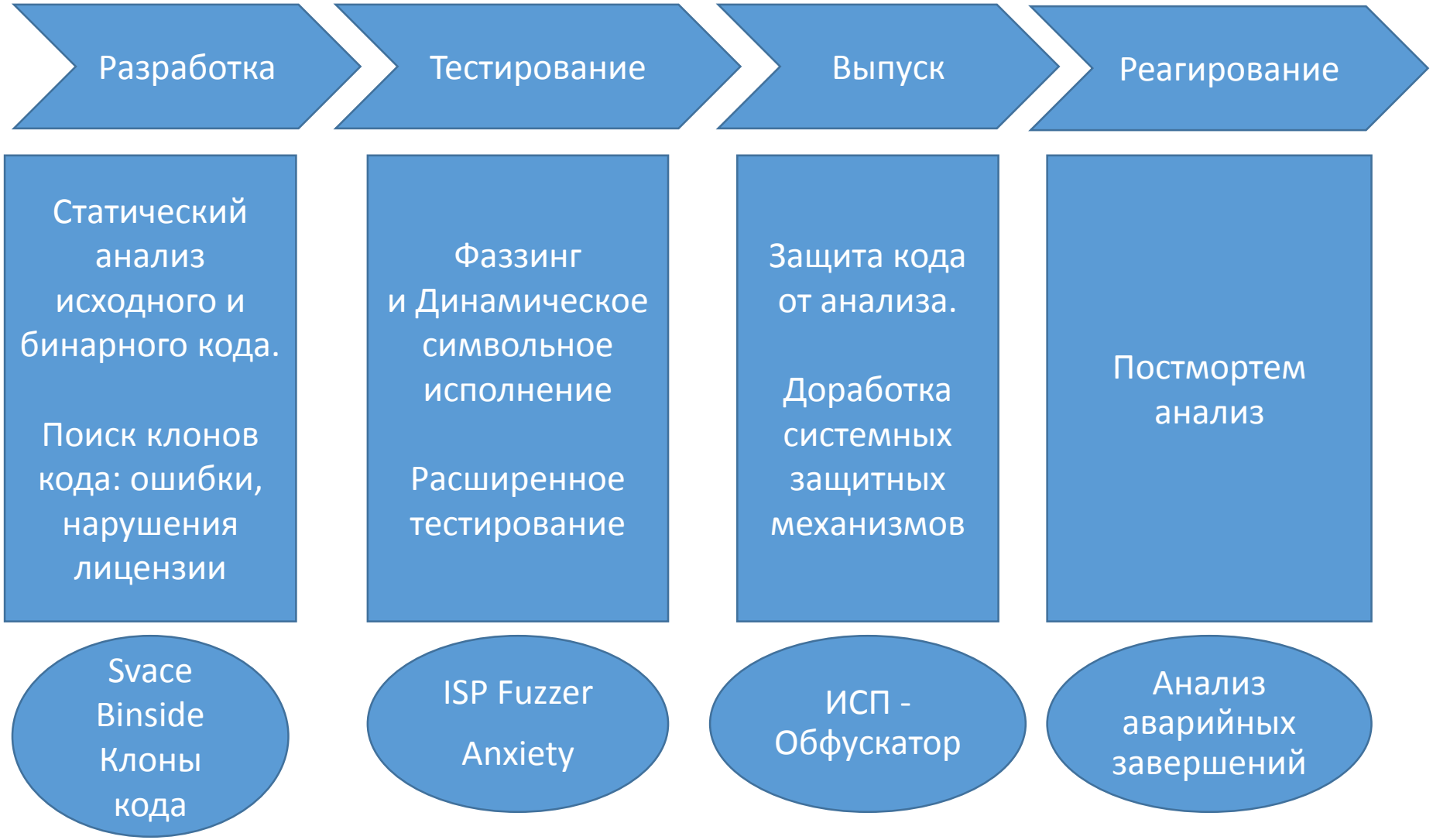
- Программа содержит не более 1 истинного срабатывания инструмента статического анализа на 1000 строк (1 TP/KLOC);
- Менее 0.1 TP/KLOC;
- Менее 0.01 TP/KLOC, при уровне ложных срабатываний не более 20% и ни одного дефекта, помеченного пользователем как важный (“major”).

- Только несколько проектов с открытым кодом достигли уровня 2 или выше (PostgreSQL).
- Ядро Linux анализируемое с 2006 года содержит 0.47 ошибки на 1000 строк – 1 уровень.

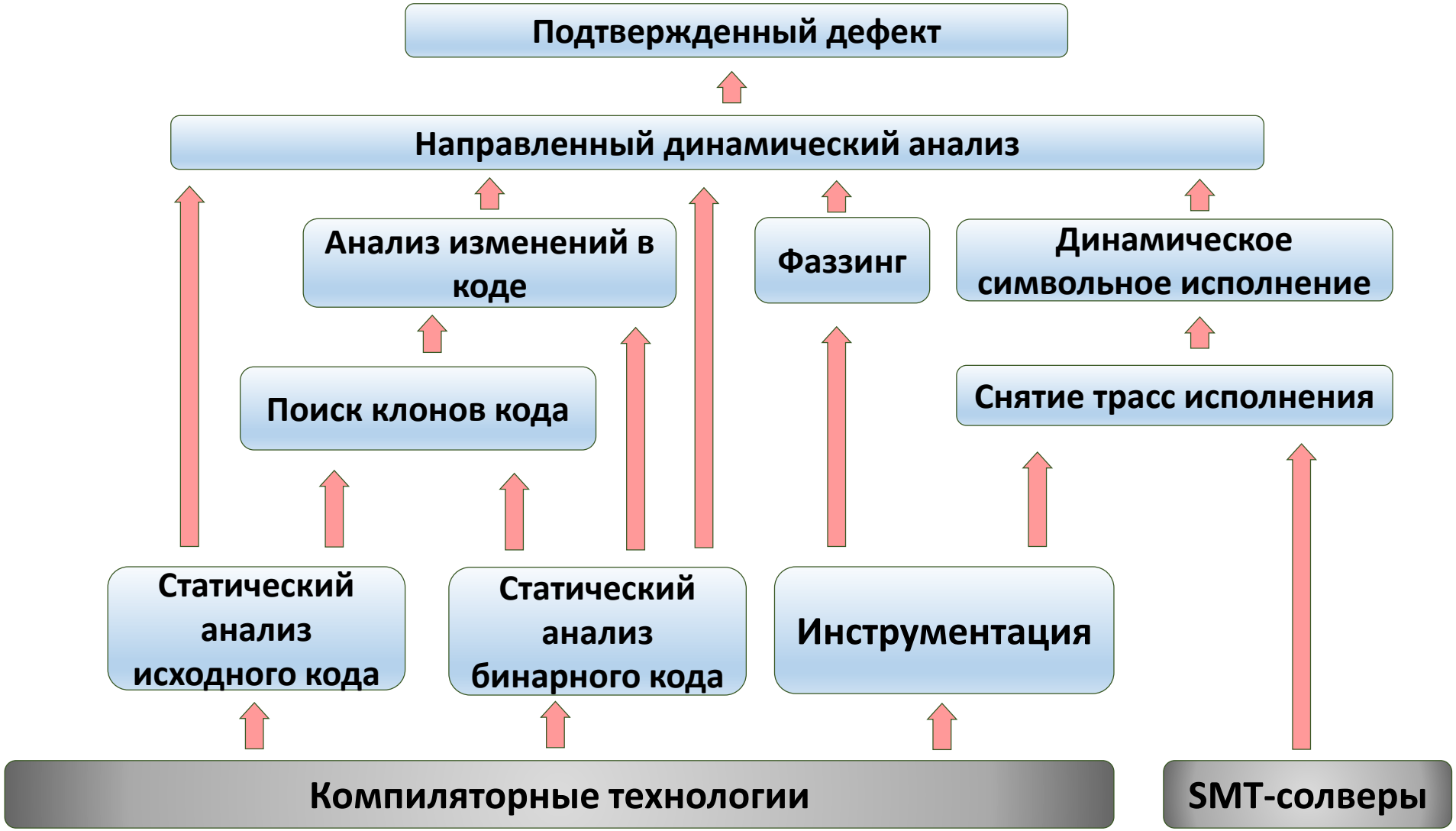
Дефекты

- Heartbleed (OpenSSL) – чтение памяти на сервере или клиенте. Конец 2011- апрель 2014. На момент публикации уязвимости было подвержено около 0.5 млн сайтов (CVE-2014-0160)
- Клиент OpenSSH версии от 5.4 до 7.1, утечка приватного ключа клиента, возможна атака MITM при первом подключении к новой системе (CVE-2016-0777). Трудно сказать ошибка или закладка.
- Ядро Linux от 3.8 до 4.5 - локальный пользователь может получить права суперпользователя (CVE-2016-0728)
- FreeBSD версии 9.3, 10.1 и 10.2 , отказ в обслуживании, повышение привилегий, раскрытие данных (CVE-2016-1881,CVE-2016-1880, CVE-2016-1879, CVE-2016-1882, CVE-2015-5677)
- Apple IOS 6.0-9.2 и OSX 10.0-10.11.2, повышение привилегий, выполнение произвольного кода, отказ в обслуживании (CVE-2016-1722)

Жизненный цикл ПО и инструменты

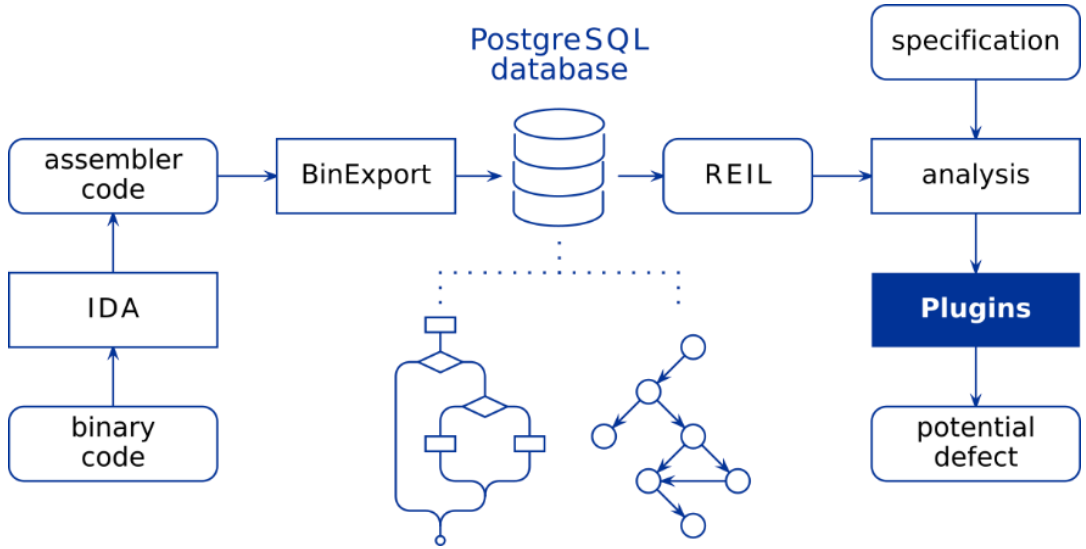


Пирамида технологий анализа



Статический анализ

- Анализ исполняемого кода
- Поиск дефектов, приводящих к аварийному завершению
- Дополняет анализ на уровне исходного кода
- Знание о структуре программы, которое можно использовать в динамическом анализе и фаззинге



Статический анализ

- Внутрипроцедурный анализ
  - Анализ значений
  - Анализ достигающих определений
  - Построение DEF-USE цепочек
  - Трансформация удаления мертвого кода
  - Анализ помеченных данных
  - Анализ работы с памятью
- Межпроцедурный анализ
  - Построение ациклического графа вызовов
  - Генерация аннотаций
  - Использование аннотаций для библиотечных функций

Статический анализ, клоны кода

- Нарушение лицензии OSS
- Использование уязвимых библиотек
- Известные уязвимости
- Восстановление отладочной информации
- Обнаружение copy-paste дефектов
- Обнаружение похожей функциональности в программах

Клоны бинарного кода

Фрагмент кода	Клон типа 1	Клон типа 2	Клон типа 3

public main
main proc near
 	
var_4= dword ptr -4
argc= dword ptr 8
argv= dword ptr 0Ch
envp= dword ptr 10h
 	
push ebp
mov ebp, esp
mov [ebp+var_4], 5
mov eax,[ebp+var_4]
imul eax,[ebp+var_4]
leave
retn
main endp
	public main
	main proc near
	
	var_4= dword ptr -4
	argc= dword ptr 8
	argv= dword ptr 0Ch
	envp= dword ptr 10h
	
	push ebp
	mov ebp, esp
	mov [ebp+var_4], 5
	mov eax,[ebp+var_4]
	imul eax,[ebp+var_4]
	leave
	retn
	main endp
		public main
		main proc near
		
		var_4= dword ptr -4
		argc= dword ptr 8
		argv= dword ptr 0Ch
		envp= dword ptr 10h
		
		push ebp
		mov ebp, esp
		mov [ebp+var_4],**10**
		mov **ecx**,[ebp+var_4]
		imul **ecx**,[ebp+var_4]
		leave
		retn
		main endp
			public main
			main proc near
			
			var_1= dword ptr -4
			argc= dword ptr 8
			argv= dword ptr 0Ch
			envp= dword ptr 10h
			
			push ebp
			mov ebp, esp
			mov [ebp+var_1],**15**
			mov **ecx**,[ebp+var_1]
			
			leave
			retn
			main endp

Развитие методов динамического анализа

- 50-е годы XX века - подача случайных перфокарт или случайным образом перемешанных
- 70-е генерация случайных программ по заданной грамматике
- 1983 The Monkey – генерация случайного ввода от пользователя
- 2006 Sidewinder – эволюционный фаззер, использует покрытие кода как фитнес функцию
- 2007 комбинированное реальное и символьное выполнение на базе инструмента CUTE
- 2012 Mayhem комбинирует реальное и символьное выполнение и строит эксплоит
- 2013 AFL (American Fuzzy Lop)
- 2016 Driller – комбинирует символьное выполнение и фаззинг
- 2017 Vuzzer - эволюционный фаззер использующий статический анализ

Требования и проблемы

- Аварийные завершения должны быть:
  - Воспроизводимыми
  - Уникальными
- Воспроизводимость обеспечивается:
  - Детерминированной работа фаззера и программы
  - Исключительная ситуация в инструментаторе != креш
- Причины не уникальности
  - Требования к скорости работы - дешевый онлайн анализ
  - К ошибке можно прийти разными путями, на разных входных данных
    - Чувствительность к путям
    - Ошибка в вызываемой функции, не зависящая от места вызова
  - Характер разрушения данных

```
1. int a,b,c;
2. ...
3. c = 0;
4. if (a>b)
5.   print ("a>b\n");
6. else
7.   print ("a<=b\n");
8. ...
9. a = b/c;
```

Качественные параметры

Кластер из 100 узлов, тестирование заняло 28 дней машинного времени* (проанализировано 37 тыс. программ из состава Debian Linux)

*от авторов инструмента ***Mayhem***, David Brumley, Thanassis Avgerinos

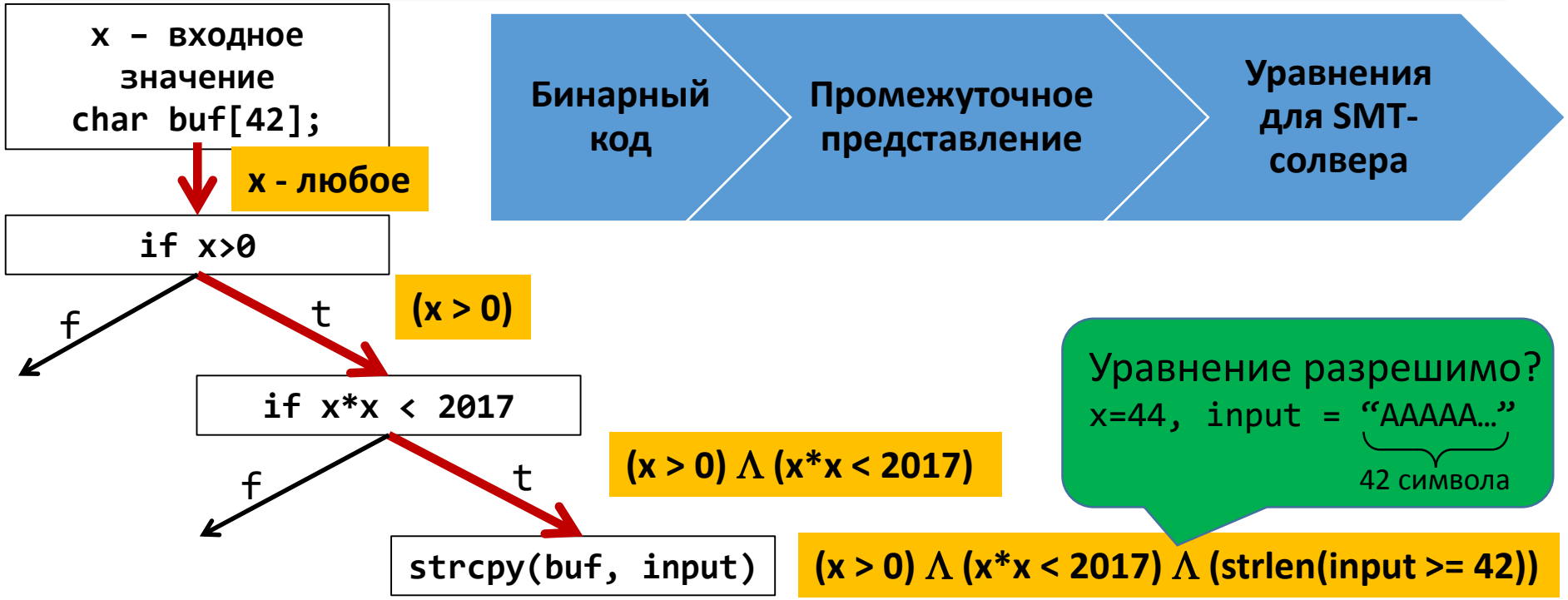
Степень критичности

Требуется анализ контекста процесса в момент аварийного завершения программы:

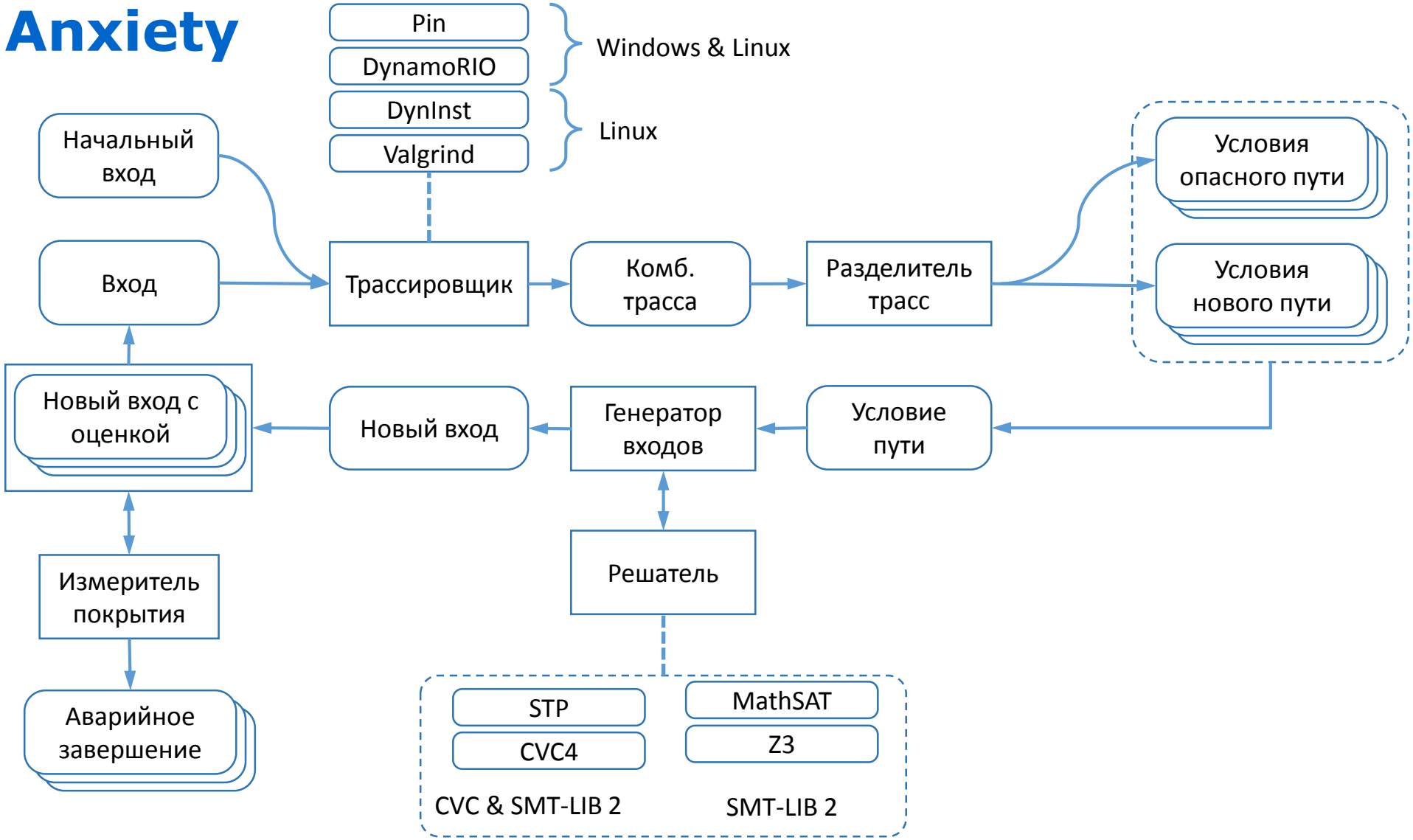
- Microsoft
  - Плагин к WinDbg !exploitable
- Cert
  - Плагин к GDB exploitable
- ИСП РАН
  - Плагин инструментатора DynamoRIO, может работать совместно с поиском уникальных аварийных завершений
  - Онлайн анализ пользовательского ПО в Linux системах (x86/ARM)
  - Символьное выполнение трасс программы для оценки критичности

Символьное выполнение

- Нерешенные проблемы
  - «Экспоненциальный взрыв» числа символьных состояний
  - Предикаты безопасности
- Решаемые проблемы
  - Поддержка множества процессорных архитектур
  - Автоматизация анализа

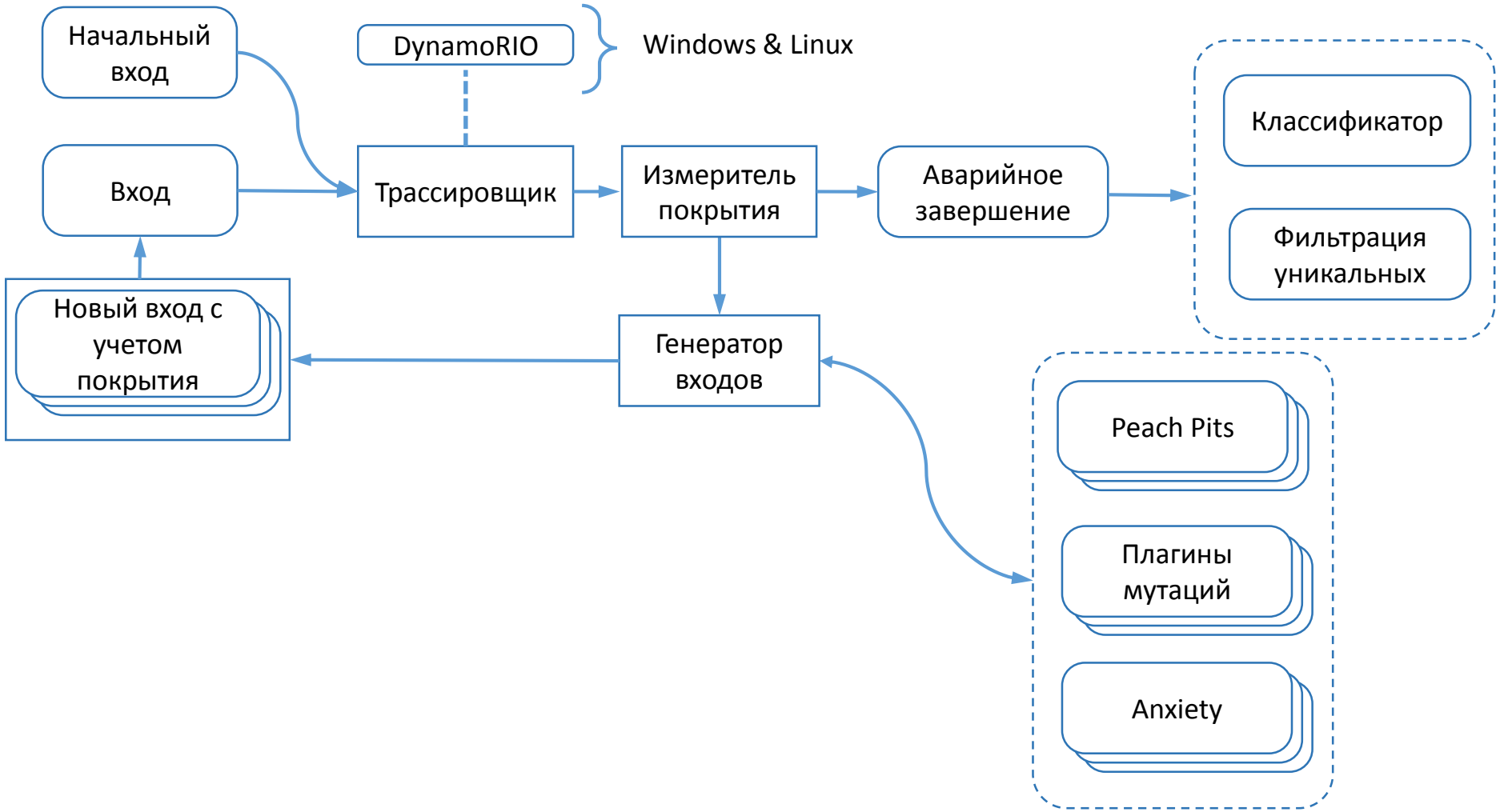


Динамический анализ



Динамический анализ

Фаззер

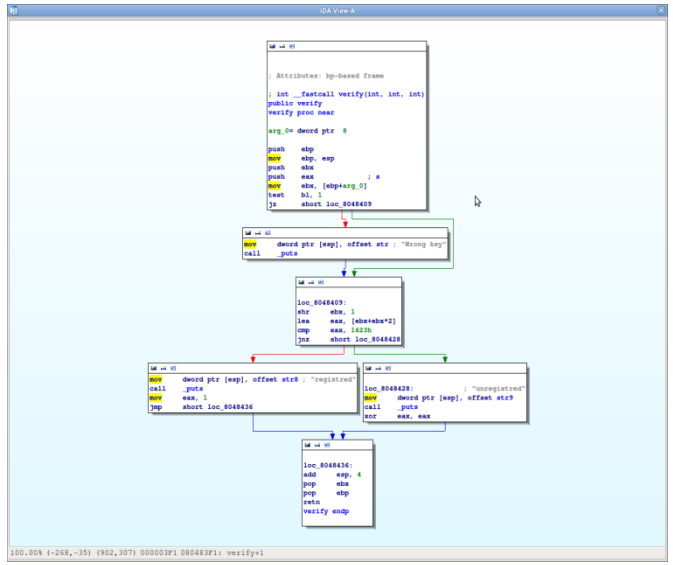


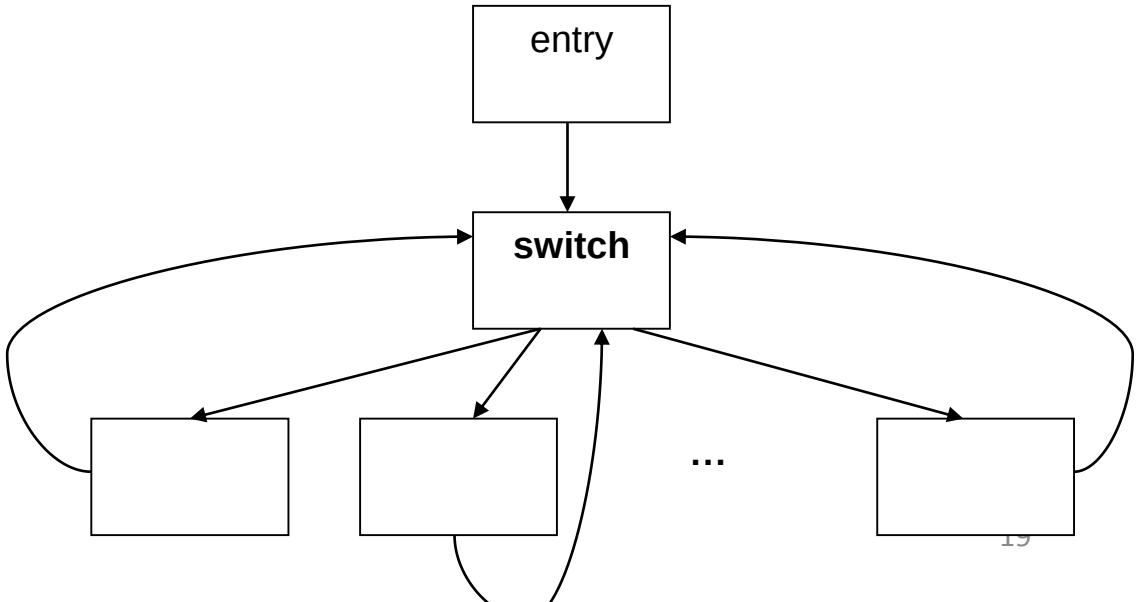
Обфускация, задачи

- Защита от восстановления используемых алгоритмов и структур данных;
- Затруднение генерации эксплоитов на основе анализа патчей;
- Препятствие эксплуатации известной уязвимости в коде программ для разных клиентов;
- Простановка «водяных» знаков на версиях программ для разных клиентов;
- Защита от вмешательства в работу программы;
- Затруднение идентификации используемых компонентов с открытым исходным кодом;
- Усложнение идентификации автора кода.

Пример: Маскирующее преобразование «диспетчер»

- В начало функции вставляем блок «диспетчер» - аналог switch в языке C
- Усредняем размер базовых блоков
- Создаем несколько копий каждого базового блока
- В конец каждого блока дописываем переменную, содержащую номер следующего блока
- Переменная диспетчеризации сцепляется посредством xor с «живыми» переменными в каждом блоке





Пример: Автоматическое шифрование буферов

- Буфер расшифровывается перед каждым обращением.
- Автоматическое шифрование после обращения не всегда возможно (имеются операции с указателями)

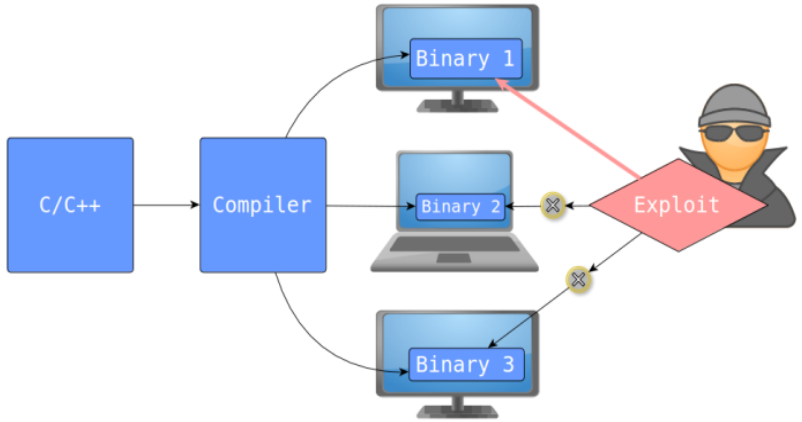
```
sub     rcx, rbp
mov     qword ptr [rsp+100h+n], rcx
call    decryptn
mov     rcx, qword ptr [rsp+100h+n]
mov     edx, 64h
mov     rsi, rbp        ; src
mov     rdi, rbx        ; dest
sub     rdx, rcx        ; n
call    _strncat
mov     esi, 64h
mov     rdi, rbx
xor     eax, eax
call    encryptn
```

Пользователь может создать свои функции шифрования/дешифрования:
char *encrypt(char *s);
char *decrypt(char *s);
char *encryptn(char *s, int len);
char *decryptn(char *s, int len);

Обфускация

Диверсификация

- Диверсификация ПО для предотвращения эксплуатации известных дефектов
- Основано на GCC
- Возможна полная сборка CentOS 6.x, 7.x
- Замедление на 1-2%
- В промышленной эксплуатации



Мелко гранулярная ASLR и переупорядочивание функций в памяти для предотвращения атак на базе ROP

- Комплексная технология (binutils, Glibc, PAX)
- Сборка всей системы CentOS 6.x, 7.x
- Замедление на 1%
- В промышленной эксплуатации

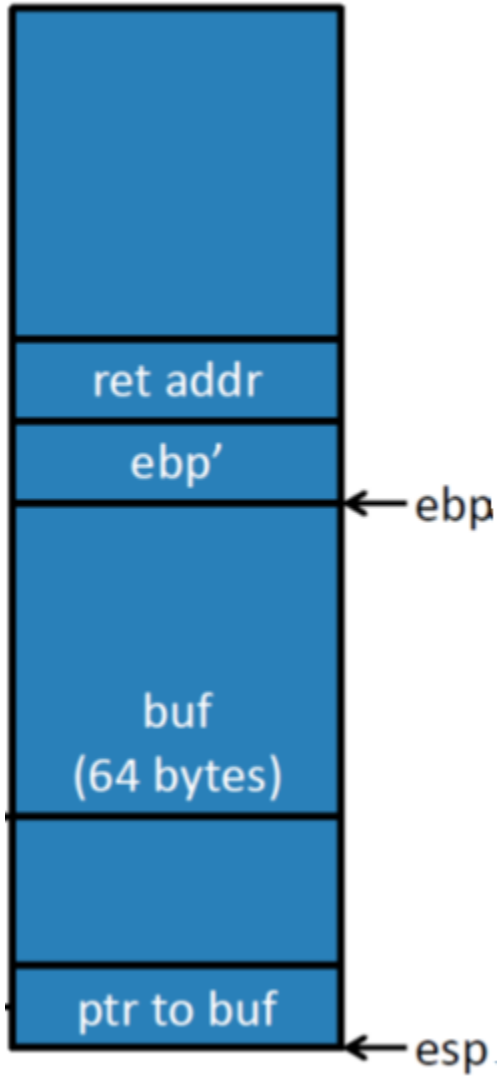
Затруднение обратной инженерии

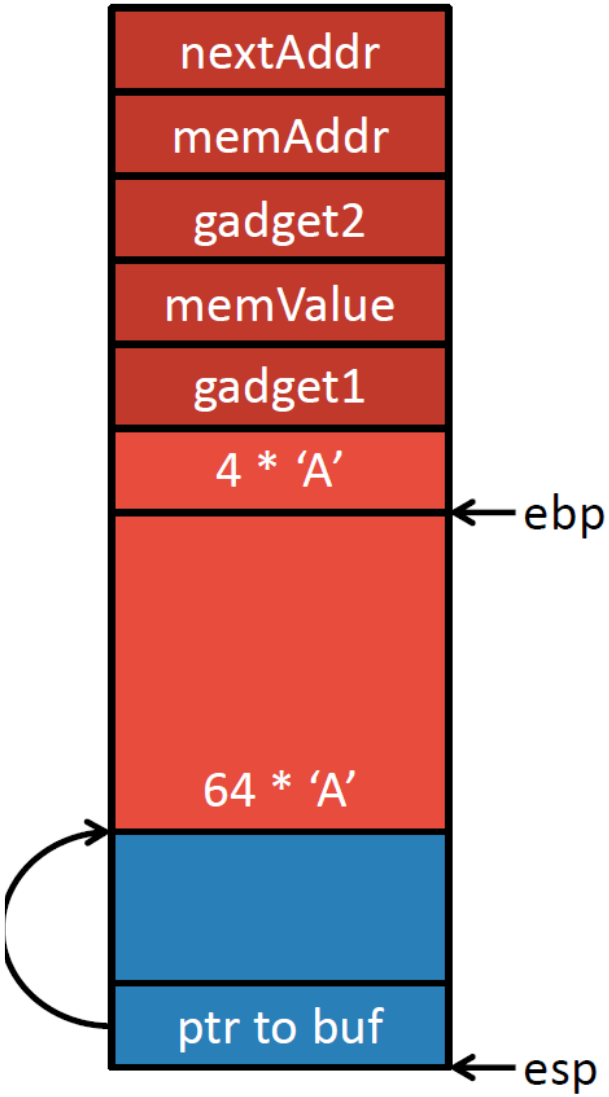
- Основано на LLVM
- Увеличение размера x 1.05-2.5
- замедление x 1.2-5.5

Анализ ROP цепочек

```
gadget1: pop eax
         ret

gadget2: pop ebp
         mov [ebp], eax
         ret
```





Спасибо за внимание!