А. Ю. Тихонов
Московский инженерно-физический институт (государственный университет)

КЛЮЧЕВЫЕ ТЕХНОЛОГИИ, СПОСОБСТВУЮЩИЕ РЕШЕНИЮ ЗАДАЧ ВОССТАНОВЛЕНИЯ И АНАЛИЗА АЛГОРИТМОВ

В работе описаны задачи восстановления алгоритмов при отсутствии исходных текстов реализующих их программ, а также ряд ключевых технологий, способствующих решению таких задач. Предложена методика автоматизированного динамического анализа ПО, основанная на получении и последующем анализе трассы выполнения программы.

1. Задачи восстановления алгоритмов

В выполнении исследований программного обеспечения при отсутствии исходных текстов заинтересованы многие государственные и негосударственные организации. Целью таких исследований является восстановление реализации алгоритмов и их представление в понятном аналитику виде, восстановление протоколов и форматов данных, поиск «недокументированных» возможностей, ошибок, закладок и уязвимостей. Решение таких задач может представлять интерес также с точки зрения сертификационных исследований ПО, поскольку подобные исследования, проводимые только на уровне исходных текстов, не дают полной картины работы программы вследствие известного эффекта неполного соответствия между логикой работы программы, представленной исходными текстами, и логикой работы программы, представленной в виде машинных инструкций [1]. Опубликовано небольшое число методик анализа исполняемых программ, но они основаны на статическом анализе, как, например, в работе [2].

Сформулируем наиболее важные задачи восстановления алгоритмов:

· выделение интересующих аналитика алгоритмов из реализации программы;

· поиск закладок и слабостей;

· восстановление протоколов и форматов данных

· перевод программной реализации исследуемого алгоритма в криптографическую модель для передачи математикам для криптоаналитического исследования.

Главным и зачастую единственным на сегодня эффективным методом решения таких задач является комбинация методов статического анализа (используется дизассемблер, декомпилятор) и «ручного» динамического анализа (используется отладчик и некоторые вспомогательные средства — дамперы, мониторы и т. д.). Среднее время решения типовой задачи восстановления алгоритма при условии, что исполняемый код не был защищен от анализа, может достигать полугода. Некоторые крупные задачи решаются годами. При наличии в программе серьезных защит исполняемого кода, применении обфускации, виртуальных машин сроки исследования растягиваются на неопределенное время. Подобные средства защиты реализуют идеи разработчиков с использованием всей вычислительной мощи современных компьютеров, а успех их исследования зависит от одного-единственного ресурса — мозга аналитика. Его возможностей уже недостаточно для анализа огромных объемов информации, получаемых в процессе исследования. Второй мощнейший ресурс — сама ЭВМ — остается практически незадействованным в анализе.

2. Ключевые технологии, способствующие решению задач восстановления и анализа алгоритмов

Эффективного инструментария, впрочем, как и методики решения подобных задач, до сих пор никто не предлагал. Между тем для успешного решения задач анализа как систем криптографической защиты информации, так и вообще любых современных систем организации документооборота требуются адекватные методы и средства восстановления и анализа алгоритмов. Применение ручных методов с использованием только дизассемблеров и отладчиков уже не позволяет успешно анализировать многие сложные программные продукты.

Можно прогнозировать, что при сохранении существующего подхода к решению задач восстановления и анализа алгоритмов в течение ближайших 5 лет будет достигнута ситуация, когда ни одна новая задача не сможет быть решена.

Методика решения перечисленных задач восстановления алгоритмов может быть получена путем синтеза результатов исследований по следующим направлениям:

· **Виртуализация**. В новых процессорах с архитектурой x64 фирм Intel и AMD реализована аппаратная поддержка виртуализации. Данная технология потенциально имеет отладочные возможности, несравнимые по мощности с возможностями старых процессоров.

· **Обфускация (запутывание алгоритма)**. Эквивалентное преобразование кода программы, формирующее код, анализ которого существенно затруднен по сравнению с первоначальным. На практике применение запутывания приводит к невозможности анализа программы ручными методами. Впервые задача обфускации упомянута в работе [3], подробная классификация запутывающих преобразований дана в [4]. Чрезвычайно бурное развитие технологий обфускации в последнее время связано с заинтересованностью многих крупных фирм в использовании технических средств защиты авторских прав (Digital Right Management).

· **Оптимизация**. Эквивалентное преобразование кода программы, которое может быть использовано для устранения запутывающих преобразований и упрощения понимаемости кода.

· **Слайсинг**. Выделение из исполняемого кода или исходных текстов таких команд или операторов, которые относятся к преобразованию интересующих аналитика входных или выходных данных. Технология слайсинга может быть использована как для исследования, так и для защиты программ от исследования.

· **Декомпиляция**. Представление реализации программы в виде, облегчающем понимание. В случае сложных программных продуктов эффективна только совместно со слайсингом и оптимизацией. Пожалуй, единственной подробной открытой работой по этой теме является [5].

3. Суть предлагаемой методики анализа

Основными элементами предлагаемой методики являются:

— получение трассы как последовательности выполненных процессором инструкций и состояний регистров на каждом шаге;

— слайсинг по трассе как отбор множества шагов трассы, преобразующих указанные аналитиком входные ячейки (регистры, память и т. п.) в выходные;

— получение по слайсу работоспособного алгоритма;

— оптимизация слайса;

— декомпиляция слайса;

— восстановление структур данных по трассе;

— генерация обратного алгоритма и оценка мощности перебора параметров.

По первым трем пунктам нами уже получены практические результаты, позволяющие говорить о перспективности выбранного направления, ведутся работы по остальным пунктам.

Первым шагом методики является получение и сохранение трассы. В результате будет получен огромный объем данных, который практически не поддается ручному анализу. Для оценки размера трассы отметим, что трассировке подвергается работа процессора в течение нескольких секунд, при этом сохраняется информация о состоянии процессора для каждой выполненной им инструкции. Если принять скорость работы процессора 1 млрд операций в секунду (1000 MIPS) и размер одного состояния 100 байт, получим размер трассы около 100 Гбайт. При решении нами практических задач размер трассы редко превышал 10 Гбайт, однако при анализе сложных программных систем, для которых искомый алгоритм заранее не локализован, может потребоваться выполнять анализ в течение десятков секунд.

Следующим шагом методики является выявление в трассе «точек зацепления» — шагов трассы, заведомо принадлежащих искомому алгоритму, либо ячеек памяти, содержащих входные или выходные данные искомого алгоритма или промежуточные результаты его работы. Что именно ищется — шаг

трассы или обрабатываемые ячейки — равнозначно, так как одно определяет другое. Шаги трассы чаще всего ищутся на основе знания стандартного программного интерфейса, предоставляемого конкретной ОС, или интерфейсов взаимодействия с внешним оборудованием. Ячейки ищутся по содержимому.

После обнаружения точки зацепления, к трассе применяется алгоритм слайсинга, результатом работы которого могут быть другие точки зацепления. После нескольких итераций слайсинга аналитик получает восстановленный алгоритм и переходит к этапу создания листинга или контрольного примера, применения оптимизирующих преобразований или выполнению декомпиляции для получения высокоуровневого представления алгоритма.

Рассмотрим элементы данной методики подробнее.

3.1. Задача получения трассы

Трассу как последовательность выполненных процессором инструкций и состояний регистров на каждом шаге можно получить разными методами: путем программной трассировки, трассировки с помощью симулятора оборудования либо аппаратной трассировки.

Первый способ наиболее прост в реализации, но наиболее сильно уязвим для методов защиты от трассировки.

Второй способ наиболее сложен в реализации, но принципиально и при корректной реализации неуязвим для методов защиты (но только при анализе замкнутой системы, когда внутри симулятора исполняются все компоненты анализируемой системы).

Серьезная проблема при выполнении программной трассировки и использовании симуляторов — падение производительности. Скорость работы программы на симуляторе на 3—5 порядков ниже скорости работы на обычном компьютере. Это делает невозможным использование первых двух способов для анализа «незамкнутых» систем, когда одна часть системы недоступна для аналитика и работает на неподконтрольном ему компьютере. Возможный путь решения этой проблемы — аппаратная трассировка.

Высокоскоростная аппаратная трассировка возможна только с помощью специализированных версий процессора и материнской платы. Разработку такой аппаратуры может позволить себе только разработчик процессора.

В последнее время наметилась тенденция переноса технологий виртуализации непосредственно в процессоры. Так, два крупнейших производителя процессоров для персональных компьютеров — Intel и AMD — разработали технологии виртуализации Intel VT и AMD Pacifica. Реализация получения трассы на основе этих технологий может решить проблему программного получения трассы для процессоров IA32/IA64 в случае анализа «замкнутых» систем.

3.2. Задача динамического слайсинга по трассе

Задача слайсинга в том случае, когда аналитик работает с представлением программы в виде инструкций процессора, заключается в отборе множества инструкций, имеющих отношение к работе интересующего аналитика алгоритма. Данная задача сводится к задаче определения достижимости вершины на графе связи инструкций по данным.

Нами разработан подход к выполнению динамического слайсинга по трассе. Данный подход, в частности, позволяет:

— автоматизировать процесс выделения алгоритма в пригодном для генерации контрольного примера виде;

— автоматизировать процесс поиска подозрительных на наличие закладки мест в коде.

3.3. Генерация листинга по трассе

Можно сформулировать несколько задач генерации листинга:

— «простая» генерация листинга;

— генерация «работоспособного» листинга.

«Простая» генерация листинга — упорядочение всех инструкций трассы в порядке возрастания адреса их размещения в памяти и исключение повторов. В результате мы получаем листинг, близкий к виду, формируемому дизассемблером, но без ошибок, присущих статическому анализу.

Сложность — работа с самомодифицируемым кодом.

Генерация «работоспособного» листинга — формирование такого листинга, который может быть откомпилирован в работоспособный код без нарушения логики работы алгоритма. Для этого нужно решить 2 задачи:

— замена константных операндов на метки, если в дальнейшем значения таких операндов будут являться адресами памяти или участвовать в вычислении адресов памяти. Если не произвести таких замен, листинг будет работоспособным только в случае размещения кода и данных по тем же адресам, где они были при снятии трассы;

— формирование ассемблерных директив определения данных и значений данных для всех встреченных в листинге обращений к данным.

3.4. Оптимизация

Задача оптимизации чрезвычайно актуальна в связи с активным развитием в области защиты программ от анализа с помощью запутывающих преобразований (обфускация). Нашей задачей является перенос существующих методов деобфускации на случай динамического анализа по трассе. За критерий оптимизации можно взять минимизацию числа инструкций, хотя в принципе критерием оптимизации должна выступать мера понятности программы, этот вопрос еще нуждается в исследовании.

Под оптимизацией будем понимать эквивалентное преобразование трассы или слайса (эквивалентное в том смысле, что не влияет на результат преобразования входных ячеек в выходные), уменьшающее число инструкций и упрощающее понимание алгоритма.

Оптимизация нацелена в первую очередь на упрощение процесса восприятия алгоритма человеком. В ряде работ, в частности в работах Института системного программирования, исследованы существующие подходы к выполнению обфускации и соответствующие приемы деобфускации, основанные на оптимизирующих преобразованиях [7]. Принятая на данный момент точка зрения — существующие подходы к обфускации являются нестойкими к автоматизированному динамическому анализу, поэтому построение методики анализа ПО на основе динамического анализа позволяет решить задачу деобфускации.

3.5. Декомпиляция

Задачей декомпиляции является представление анализируемого алгоритма в удобном аналитику виде. Обычно эта задача рассматривается вполне традиционно — как перевод реализации алгоритма из машинного представления или представления на языке ассемблера в представление на языке высокого уровня, не обязательно исходного.

Применение декомпиляции ко всей программе целиком во многих случаях нецелесообразно — зачастую аналитика интересует какой-то один аспект поведения программы, описываемый слайсом. Поэтому мы ставим задачу декомпиляции применительно к слайсу интересующего нас алгоритма, причем к слайсу предварительно должны быть применены оптимизирующие преобразования.

С точки зрения анализа криптографических алгоритмов актуальной является постановка задачи декомпиляции как перевод бинарного кода в некую схему алгоритма в терминах криптографических примитивов. Решение этой задачи позволит решить проблему передачи знаний о восстановленном алгоритме в криптоаналитические подразделения.

3.6. Восстановление протоколов и структур данных

Задачу восстановления формата данных определим как восстановление структуры блока данных, являющегося выходом некоторого алгоритма, а также выделение алгоритмов, отвечающих за формирование элементов этой структуры. Аналогичным образом определяется задача восстановления протоколов. В такой формулировке можно решать эту задачу с помощью динамического слайсинга.

Единственная известная нам на данный момент работа по теме восстановления структур данных [6], подтвержденная практическими результатами, основана на статическом анализе исполняемого кода и неприменима для анализа сложных систем или систем, содержащих защищенные от статического анализа алгоритмы. Тем не менее результаты этой работы вполне могут быть адаптированы к предлагаемому нами динамическому методу анализа.

Еще одной задачей восстановления протоколов и форматов данных является задача формального описания и использования знаний о структурах данных и о процессах взаимодействия с помощью сетевых пакетов в соответствии с восстанавливаемым протоколом. Имеющиеся коммерческие разработки сложны в использовании и не адаптированы к нашим требованиям.

3.7. Распознавание алгоритмов в трассе

В настоящее время распространен подход сигнатурного поиска функций при статическом анализе листинга программы. В качестве примера можно привести технологию FLIRT, используемую в дизассемблере IDA. Подобный подход позволяет выявлять реализацию известных алгоритмов, но не позволяет искать разные реализации одного и того же алгоритма.

Задача выявления алгоритмов, «похожих» на заданный образец, имеет много общего с задачей распознавания образов. Имеются различные подходы для решения этой задачи — обработка статистики встречаемости инструкций методами кластерного анализа, генетические алгоритмы распознавания образов и т. п.

Все перечисленные подходы разрабатывались в расчете на статический анализ программы, и в силу присущих самому статическому анализу проблем ни одна известная в настоящее время реализация не предлагает приемлемое решение для этой задачи.

Адаптация существующих методов к динамическому анализу по трассе может позволить решить задачу распознавания с большей эффективностью.

3.8. Пути развития алгоритмов анализа трассы для облегчения криптоаналитических исследований

Одной из видимых на данный момент задач является задача построения обратного алгоритма по динамическому слайсу на тех входных/выходных данных, на которых слайс был получен. Если алгоритм необратим — указать в слайсе необратимые инструкции и оценить требуемый объем перебора значений для обращения. Данная задача представляется очень перспективной.

Другая задача связана с дальнейшим развитием декомпиляции для преобразования слайса криптографического алгоритма в криптографическую схему.

4. Возможности, принципиально реализуемые в рамках предлагаемой методики

В рамках предлагаемой методики принципиально возможно решение следующих задач:

— автоматизация процесса восстановления алгоритма и получения контрольного примера на том множестве входных данных, на котором получена трасса;

— автоматическое устранение запутывающих преобразований;

— автоматизация процесса поиска алгоритмов, подозреваемых на наличие закладок, активных в момент снятия трассы;

— автоматизация процесса восстановления протоколов и форматов данных;

— автоматическая подготовка описаний восстановленного алгоритма.

5. Демонстрация применения методики

Демонстрацию работы системы анализа выполним на примере двух программ, вычисляющих одну и ту же функцию:

```
Function test_func(ByVal n As Integer) As Integer
If n = 0 Then test_func = 0 Else test_func = n + test_func(n - 1)
End Function
```

Первая программа — vb6test — реализована на VisualBasic версии 6.0 и скомпилирована в виде p-кода (кода виртуальной машины). Вторая программа — vbnettest — реализована на VisualBasic .NET и скомпилирована в виде управляемого кода.

Трасса снималась для n=10. Размер трассы, полученной для обоих примеров, составляет около 700 Мб. Результаты обработки трассы представлены в таблице 1, время выполнения анализа — меньше минуты.

Таблица 1.

	Полная трасса процесса			Трасса user-mode			Инстр. в слайсе
	Размер, Мб	Число шагов	Инстр. в листинге	Размер, Мб	Число шагов	Инстр. в листинге	
vb6test	168	2 326 412	76 951	42	575 392	26 620	356
vbnettest	127	1 752 548	115 486	35	484 248	62 726	143

Из таблицы видно, что в случае ручного анализа тех же программ аналитик имел бы дело либо с миллионами шагов при динамическом анализе, либо с десятками тысяч шагов при статическом, использование данной методики за несколько минут сводит задачу анализа к нескольким сотням инструкций в слайсе, т. е. упрощает задачу аналитика на 2—4 порядка.

6. Заключение

В работе предложена методика автоматизированного динамического анализа программного обеспечения, основанная на получении и последующем анализе трассы выполнения анализируемой программной системы. Методика ориентирована на анализ машинного представления программ при отсутствии исходных текстов.

Предлагаемая методика частично реализована в модели системы анализа трассы — TREX (TRace EXplorer).

В разделе 3 сформулированы основные задачи, которые нужно решить для полноценной реализации комплекса анализа ПО. Даже частичное решение поставленных в работе задач и реализация такого решения в виде единого инструментария анализа программ позволят перевести задачу восстановления алгоритмов из разряда «искусства» в разряд инженерной дисциплины, со всеми вытекающими последствиями — упрощение обучения специалистов, прогнозируемость возможности и сроков решения задач, существенное — на порядки — уменьшение времени решения задач восстановления и анализа алгоритмов, возможность решения задач, не решаемых ручными методами.

Следует отметить тот факт, что практически все сформулированные в работе задачи относятся к стыку областей системного программирования и криптографии, причем решение этих задач неразрывно связано.

СПИСОК ЛИТЕРАТУРЫ

1. *Balakrishnan G., Reps T., Melski D., Teitelbaum* T. WYSINWYX: What You See Is Not What You eXecute. To appear in Proc. IFIP Working Conference on Verified Software: Theories, Tools, Experiments. Zurich, Switzerland, Oct. 10—13, 2005.

2. *Balakrishnan G., Reps T., Kidd N., Lal A., Lim J., Melski D., Gruian R., Yong S., Chen C.-H., and Teitelbaum T.* Model checking x86 executables with CodeSurfer/x86 and WPDS++, (tool-demonstration paper) // Proc. Computer-Aided Verification, 2005.

3. *Diffie W., Hellman M. E.* New directions in cryptography. IEEE Transactions in Information Theory, 22, 1976. P. 644—654.

4. *Collberg C., Thomborson C., Low D.* A Taxonomy of Obfuscating Transformations. Department of Computer Science, The University of Auckland. 1997.

5. *Cifuentes C.* Reverse Compilation Techniques. PhD dissertation, Queensland University of Technology, School of Computing Science. July, 1994.

6. *Lim J., Reps T., and Liblit B.* Extracting output formats from executables // Proc. IEEE Working Conference on Reverse Engineering. Benevento, Italy, Oct. 23—27, 2006.

7. *Чернов А. В.* Анализ запутывающих преобразований программ // Труды Института системного программирования РАН. 2002. Т. III.