И. О. Леошкевич, А. Н. Велигура

НИЗКОУРОВНЕВАЯ ПОЛИТИКА ОБРАБОТКИ ДАННЫХ

Введение

Защита конфиденциальной информации от несанкционированного доступа на уровне автоматизированных систем осуществляется в рамках политик разграничения доступа. При этом предполагается корректность программ, обрабатывающих конфиденциальную информацию от имени пользователя, в частности, считается, что они не сделают ее общедоступной и своевременно удалят промежуточные данные. Это допущение отсутствует в системах, работающих по модели Белла—Лападула, где выполняется свойство «звезды» — процессы с более высоким уровнем конфиденциальности не могут осуществлять запись в области с более низким уровнем конфиденциальности. Однако многие распространенные на сегодняшний день системы, в том числе из соображений удобства, используют другие модели, например дискреционное управление доступом, в которых ошибки в программном обеспечении могут привести к нарушению конфиденциальности.

Поиск и устранение таких ошибок является актуальной проблемой. Для ее решения возможно применение автоматизированных средств, анализирующих программный код. Эти средства получают на вход формальное описание его желаемых свойств и на выходе выдают заключение об их выполнимости, а в случае отрицательного результата приводят условия, например входные данные, ведущие к их нарушениям. В качестве примеров таких средств можно привести BLAST и Java Pathfinder. В настоящей работе предлагается способ задания и формализованной проверки политики обработки данных на уровне конкретного приложения.

1. Процесс обработки данных

Приложения могут получать входные данные несколькими способами: непосредственно из памяти, с диска, по сети, из графического интерфейса и т. д. Получение данных обычно происходит в три этапа: идентификация «абонента» (получение имени файла или адреса сетевого узла), установление связи с ним (например, открытие файла или сетевого соединения) и непосредственно обмен. При этом зачастую результаты обмена с одним абонентом служат отправной точкой для идентификации другого (получение имени файла из командной строки или графического интерфейса). Процесс обработки данных характеризуется с помощью:

1. Форматов данных, которые, как правило, не относятся к конкретным проверяемым программам. Их описания должны составляться независимо от политик и не должны содержать сведений об абонентах или конфиденциальности. Это позволит использовать одни и те же описания для разных целей.

2. Внешних функций, предоставляемых средой выполнения, с помощью которых осуществляется обмен с абонентами. Как и описания форматов данных, описания внешних функций не должны зависеть от конкретных политик.

3. Меток конфиденциальности, которые связываются с полями форматов. Метки позволяют отличать данные различных уровней конфиденциальности.

4. Абонентов, с которыми связываются несколько потоков, а с потоками — форматы принимаемых или генерируемых ими данных.

5. Имен абонентов, с каждым из которых связывается набор «именующих» полей, возможно, из различных форматов. Более одного поля может потребоваться в случае составного имени, например адреса удаленного узла и порта.

Пусть имеется консольная программа, осуществляющая шифрование, входом для которой является командная строка, содержащая ключи «--in» (имя файла, содержащего открытый текст), «--out» (имя файла, в который записывается шифртекст) и «--key» (имя файла, содержащего

секретный ключ). Данными, формат которых требуется задать, являются содержимое командной строки и файлов. Внешними функциями являются функции получения содержимого командной строки и работы с файлами. Метки конфиденциальности задаются следующим образом: как конфиденциальные помечаются те части форматов, в которых содержатся непосредственно биты ключа, открытого текста и шифртекста, а не служебная информация. Абонентами являются файлы (имеющие единственный поток, соответствующий хранящимся в них данным), кроме того, вводится фиктивный абонент, отвечающий за командную строку. Именами абонентов-файлов являются соответствующие фрагменты командной строки. У фиктивного абонента нет имени, так как оно нужно лишь для установления связи с ним, а командную строку обычно можно получить, вызвав единственную функцию без параметров.

2. Язык описания процесса обработки данных

Язык состоит из пяти примитивов, названных в предыдущем разделе. Форматы описываются с помощью «генерирующих» недетерминированных микропрограмм и «структурирующих» детерминированных микропрограмм. Назначением генерирующих микропрограмм является описание всех возможных входных данных, а структурирующих — определение местонахождения соответствующих им полей в заданном формате. За основу языка и среды выполнения для таких программ взят предложенный в [1] микрокод, состоящий из операций присваивания и условного перехода.

Внешние функции можно разделить на три вида — установление связи, обмен и прекращение связи. Для описания их работы в микрокод добавлены следующие операции:

1. *connect(d, s, n)*, обозначающая, что с дескриптором *d* теперь связан поток *s* абонента с именем *n*. Генерация дескриптора *d* должна осуществляться микрокодом;
2. *send(d, s, buf)*, служащая для передачи данных, и *receive(d, s, buf)*, служащая для приема данных. За симуляцию возврата ошибок и объема реально переданных данных должен отвечать отдельный микрокод, а эти команды лишь отражают сам факт передачи;
3. *seek(d, s, ρos)* и *tell(d, s, ρos)*, служащие для работы с текущим положением;
4. *close(d, s)*, закрывающая поток *s* дескриптора *d*.

С помощью этих операций можно описать большинство взаимодействий со средой. Описание работы с файлами тривиально, нужно лишь преобразовать входные параметры и привнести недетерминированную симуляцию ошибок. Специфика описания сетевых соединений заключается в том, что их нужно представлять в виде двух потоков — принимающего и отправляющего. Для функции получения параметров командной строки нужно использовать фиктивные вызовы *connect* и *close*.

Описания меток конфиденциальности используют структурирующие микропрограммы для идентификации полей форматов. Существует два вида описаний меток: «генерирующие», которые устанавливают метки на изначально конфиденциальные поля, и «проверяющие», которые снимают метки с допустимых конфиденциальных полей. Таким образом, описания меток задаются наборами вида *(field-id, t, m)*, где *field-id* — идентификатор поля, *t* — вид описания метки, *m* — непосредственно метка. Для работы с метками в микрокод добавлены операции *taint(a, ρ, n, m)* и *untaint(a, ρ, n, m)*, служащие для установки или снятия метки *m* с *n* битов по адресу *ρ* в адресном пространстве *a*.

Абоненты и их имена задаются как наборы *(id, (s, gen-format, chk-format)*)* и *(id, field*)* соответственно, где *id* — идентификатор абонента, *gen-format* и *chk-format* — форматы генерируемых и принимаемых потоком *s* данных, а *field* — поле, хранящее часть имени абонента.

3. Установление соответствия программного обеспечения требованиям политики

Проверка политики осуществляется с помощью статического анализа меток. В ходе анализа метки, связанные с одними данными, распространяются на зависящие от них другие данные. В определенных точках происходит проверка на наличие меток в непредусмотренных областях,

которое свидетельствует о возможности нарушения политики. Проверка основана на модификации алгоритма анализа, описанного в [2]. В него добавлены шаги по отслеживанию имен и дескрипторов абонентов с помощью меток, а также по обработке новых операций микрокода.

Целью интерпретации *connect* является формирование структуры всех возможных входных данных с конфиденциальными и именующими полями, помеченными в соответствии с политикой. Первым делом *connect* получает все метки, установленные на переданное ей имя абонента, и с помощью списка имен, заданного в политике, устанавливает идентификатор абонента *id*. Имена помечаются парами *(FIELD, field-id)*, где *FIELD* обозначает, что метка относится к полю, а *field-id* представляет собой идентификатор этого поля. Затем *connect* подставляет на место своего вызова микропрограмму, соответствующую формату генерируемых абонентом с идентификатором *id* данных (*gen-format*). В качестве их расположения используется специальное адресное пространство *$io* и смещение, генерируемое с помощью вызова *_new($io, offset)*, создающего новый регион в адресном пространстве *$io* (подробнее понятие региона описано в [3], где регионы используются для описания фрагментов памяти; в данной работе их применение расширено до описания ввода-вывода). После этого *connect* помечает переданный ей дескриптор меткой *(DESCR, id, s, offset, pos_offset)*. Наконец, *connect* добавляет к месту своего вызова микропрограммы для соответствующих полям *gen-format* генерирующих меток.

Интерпретация операций *send*, *receive*, *seek* и *tell* тривиальна. Заметим лишь, что адреса данных и текущей позиции извлекаются из меток, связанных с переданным им дескриптором. Выбирается та из них, чей идентификатор потока совпадает с переданным.

Целью интерпретации *close* является проверка соответствия переданных абоненту данных политике. Для этого выполняются связанные с соответствующими *chk-format* метками конфиденциальности проверяющие микропрограммы, снимающие с *$io:offset* допустимые метки, и, если *chk-format* = ∅, осуществляется проверка состояния региона *$io:offset* на наличие оставшихся меток: если таковые присутствуют, то возможно нарушение политики. Для определения операторов анализируемой программы, которые сохранили недопустимые данные, достаточно воспользоваться достигающими определениями. Наконец, с помощью операции *_delete* удаляются созданные для отслеживания потока регионы. Аналогичная проверка на наличие меток проводится при завершении программы, однако в этом случае проверяются все существующие регионы.

Заключение

Предложено расширение микрокода, представленного в [1], позволяющее описывать функции ввода-вывода. С помощью шести новых операций можно покрыть взаимодействие с произвольными внешними объектами, такими как файлы, сетевые соединения, графический интерфейс и параметры командной строки. Предложен формальный язык задания политики обработки данных, позволяющий описать допустимые операции над конфиденциальными данными на уровне их форматов и системных функций. Показано, каким образом можно установить соответствие приложения политике с помощью статического анализа его исполняемого кода.

СПИСОК ЛИТЕРАТУРЫ:

1. *Леошкевич И. О.* Получение архитектурно-независимой семантики исполняемого кода // Безопасность информационных технологий. 2009. № 4. С. 120–124.

2. *Леошкевич И. О.* Система для семантического анализа исполняемого кода // VII Межрегиональная конференция студентов и аспирантов. Применение кибернетических методов в решении проблем общества XXI века. Тезисы докладов. Обнинск: ИАТЭ НИЯУ МИФИ, 2010. С. 9–11.

3. *Balakrishnan G., Reps T., Melski D. and Teitelbaum T.* WYSINWYX: What You See is not What You eXecute // Proceedings of the Verified Software: Theories, Tools and Experiments. 2005. P. 202–213.