Д. В. Гуров

ПРИМЕНЕНИЕ АППАРАТНОГО ШИФРОВАНИЯ ПРИ РАБОТЕ СО СТЕКОМ ДЛЯ ЗАЩИТЫ ОТ ЭКСПЛОЙТОВ[1]

Введение

Использование уязвимости типа «Переполнение буфера (buffer overflow)» является одним из основных путей взлома современных приложений. Противодействию этой уязвимости уделяется большое количество времени и сил. Однако в основном все существующие методы эффективны на этапе написания исходного кода (использование безопасных функций или проверки выхода за пределы выделенного буфера) или на этапе компиляции (использование надстроек компилятора, позволяющих контролировать замещение адреса возврата или других заранее заданных областей данных). Среди программ, использующихся для повышения безопасности приложений в части переполнения буфера, можно назвать StackGuard, StackShield, LibSafe и другие [1]. Однако зачастую программа поставляется в виде исполнимого файла, и в этой ситуации проанализировать ее исходный код на предмет наличия уязвимости затруднительно, так же как и перекомпилировать программу, используя безопасный компилятор, для устранения уязвимостей данного типа.

В работе [2] был предложен способ противодействия эксплойтам, основанным на внедрении shell-кода. В основу этого способа была положена рандомизация системы команд для каждого сегмента кода. При загрузке в память очередного процесса для исполнения код каждой команды в нем замещается по специальной таблице замен, уникальной для каждого очередного программного сегмента, загружаемого в память. Для реализации такого способа защиты предложена модификация микропроцессора, которая заключалась в добавлении блока замены кода операции в процесс дешифрирования команды на этапе выборки команды из памяти для исполнения. Это позволило обеспечить полную непредсказуемость представления команд в памяти ЭВМ. В результате попытка внедрить эксплойт в выполняемую программу приводит к записи в атакуемый сегмент программного кода, который на стадии выполнения будет интерпретироваться как набор случайных чисел.

Однако существуют эксплойты, которые не внедряют новых команд ни в процесс, ни в его окружение, но, тем не менее, приводят к выполнению произвольного кода на целевой машине. В качестве примера можно привести эксплойты, построенные по схеме «ret to lib.c». Трудоемкость противодействия такому виду атаки отмечается и другими исследователями [3].

В основе данной атаки лежит загрузка в стек адресов параметров системной функции. При этом значение, замещающее при переполнении стека адрес возврата, равно адресу функции

[1] Работа выполнена в рамках Федеральной целевой программы «Научные и научно-педагогические кадры инновационной России» на 2009–2012 г.

в системной библиотеке Lib.c. Этой функции и передается управление при выполнении команды ret в конце уязвимой функции.

В качестве примера приведем оператор вызова системной функции system:

system («/bin/sh»); *//синтаксис языка C++.*

При его выполнении в память процесса помещается строка-параметр, в стек помещается адрес этой строки, после чего управление передается в системную библиотеку с помощью ассемблерной команды CALL.

Таким образом, по набору инструкций не представляется возможным отличить разрешенный вызов системной функции от фиктивного, организованного с помощью подмены адресов возврата. Бороться с атакой такого типа с помощью упомянутых выше программ (StackGuard, StackShield и др.) невозможно, ведь здесь злоумышленник не использует ни одной внедряемой команды. Внедряемых команд нет, зато внедряются данные.

В настоящей работе предлагается модификация процессора, направленная на защиту данных, помещаемых в стек, путем их шифрования при выполнении команды PUSH и расшифрования при выполнении команды POP.

Теоретическое обоснование предлагаемого метода

В области стека при вызове функции определяются так называемые локальные автоматические переменные. В то же время область стека выполняет и другую функцию — позволяет хранить в себе данные по очереди доступа LIFO. Таким образом, доступ к стеку осуществляется двумя способами: напрямую по адресам памяти (обращение к переменным) и с помощью команд PUSH и POP (по дисциплине LIFO), работающих на основании значения регистра стека ESP.

Примем модель, что при работе со стеком данные, помещаемые в него командой PUSH, извлекаются только командой POP и наоборот: то, что извлекается командой POP, было записано в стек ранее командой PUSH (а никак не непосредственной записью с использованием косвенной адресации через регистры ESP и EBP). Такая модель справедлива для большинства программ, полученных при компиляции с языков высокого уровня.

Использование уязвимости переполнения буфера по схеме «ret to lib.c» возможно вследствие того, что данные, которые помещаются в стек путем записи по адресам массива (в результате переполнения буфера), в дальнейшем извлекаются из стека с помощью команды POP. Эти извлеченные значения записываются в счетчик команд, а также используются в качестве адресов параметров вызванной функции.

Чтобы злоумышленник не смог передать нужные ему значения в счетчик команд и в параметры функции, поступим следующим образом. Те данные, которые записываются или извлекаются с помощью команд PUSH и POP, будем подвергать шифрованию и расшифрованию соответственно. Не зная ключа шифрования, злоумышленник не сможет при переполнении буфера целенаправленно заменить адрес возврата на адрес системной функции, и такие попытки будут иметь тот же результат, что и передача в счетчик команд случайного числа. Для самой же функции такая схема шифрования будет прозрачной, поскольку, как уже говорилось, данные, помещенные командой PUSH, будут извлечены командой POP и, хотя на время пребывания в стеке и будут зашифрованы, но при дальнейшей обработке будут снова получены в открытом виде. Заметим, что данные, адресуемые непосредственно (локальные автоматические переменные), не шифруются и помещаются в память «как есть».

Рассмотрим основные принципы шифрования данных при их помещении в стек. Байты операнда по отдельности подвергаются шифрованию, после чего подаются на запись в стек обычным образом. При считывании данных из стека командой POP данные расшифровываются, в результате чего на выходе получается корректное исходное значение.

Построение системы шифрования предлагается осуществлять на основе так называемых *R*-блоков [4]. Схема процесса преобразования данных показана на рис. 1.

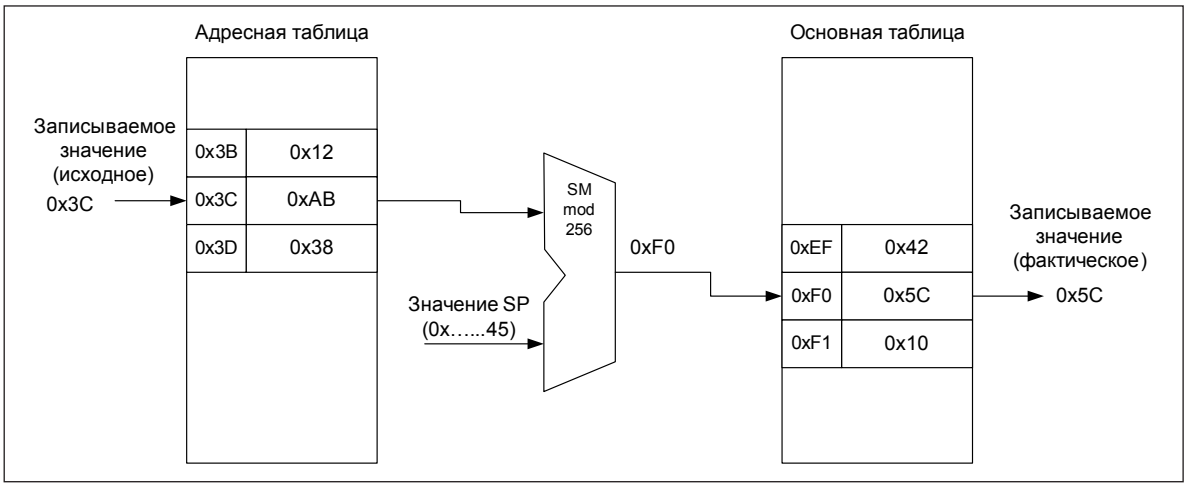

Рис. 1. Схема преобразования данных при помещении их в стек

На рисунке в качестве примера рассматривается запись в стек байта командой PUSH 0x3C.

При этом текущее значение младшего байта регистра указателя стека для примера, изображенного на рис. 1, равно 0x45. На рисунке показаны только младшие байты, поскольку все операции выполняются по модулю 256.

Как видно из рисунка, блок преобразования состоит из двух таблиц и сумматора по модулю 256. Заполнение основной таблицы происходит случайным образом, а адресная, в свою очередь, является «обратной» по отношению к основной, т. е. заполнена адресами, по которым находятся заданные значения в основной таблице (подробнее см. [4]).

Для заполнения основной таблицы предлагается использовать схему, представленную на рис. 2. В основу схемы положены 8 генераторов псевдослучайной последовательности на базе регистров сдвига с линейной обратной связью LFSR [5]. Использование LFSR обусловлено высокой степенью случайности, простой аппаратной реализацией и хорошим быстродействием. С очередным тактом каждый из регистров LFSR генерирует на своем выходе значение 0 или 1, что позволяет получить на выходе всех регистров LFSR 8-разрядное псевдослучайное число, обеспечивающее адресацию таблицы объемом до 256 ячеек. Перед началом работы (при старте системы) регистры LFSR инициализируются начальными уникальными значениями, а в каждую ячейку формируемой таблицы замен заносится ее собственный адрес, от 0 до 255. После инициализации регистров LFSR происходит процесс перемешивания.

Опишем одну итерацию процесса перемешивания. Она состоит из трех тактов. Сгенерированное на первом такте работы блока LFSR-регистров 8-разрядное число записывается в Регистр 1, после чего регистры LFSR переходят в следующее состояние. Полученное в следующем такте очередное число записывается в Регистр 2. Наконец, на третьем такте строки таблицы, адресуемые значениями из Регистра 1 и Регистра 2, меняются местами. На этом одна итерация заканчивается, и далее всё начинается с начала: регистры LFSR сдвигаются, формируется новое значение в Регистре 1 и т. д.

Перед запуском первой программы строки таблицы замен должны быть существенно перемешаны. Исследования показали, что для 90-процентного перемешивания достаточно 40 итераций.

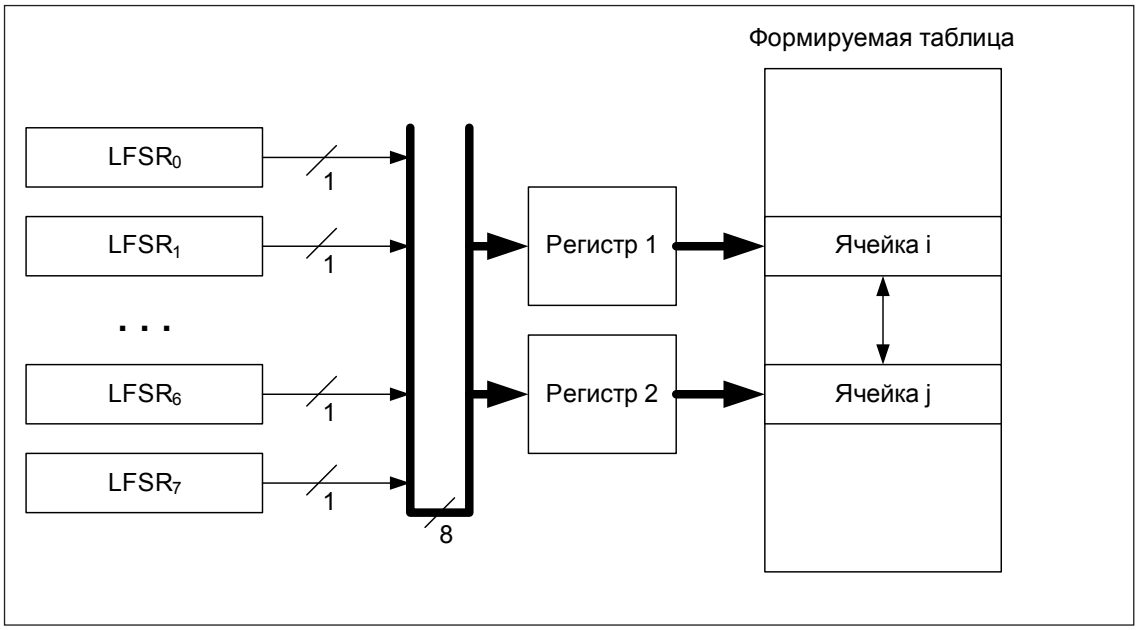

Рис. 2. Схема формирования основной таблицы

После подготовительного этапа, связанного с формированием адресной и основной таблиц, можно осуществлять процесс записи зашифрованных данных в стек.

При выполнении команды PUSH записываемый в стек операнд выставляется не на шину данных для помещения в память, а на вход блока зашифрования при операции со стеком. Также на вход блока подается значение указателя стека (регистра ESP, см. рис. 1). Операнд, который должен быть помещен в стек, обрабатывается побайтно. Выделенный байт операнда подается на вход адресной таблицы и используется в ней в качестве индекса строки. Извлеченное значение подается на сумматор по модулю 256, где складывается с младшим байтом регистра ESP, и полученная сумма используется в качестве адреса в основной таблице замен. Наконец, считанное по данному адресу значение выставляется на шину данных для помещения в стек. Использование двух таблиц (адресной и основной) при стохастическом преобразовании информации дает ускорение работы и независимость времени шифрования от порядка записей в основной таблице [4].

При длине операнда, превышающей 1 байт, для увеличения быстродействия параллельно задействуются несколько одинаковых блоков шифрования.

Аналогично осуществляется извлечение данных (и попутное их расшифрование) при выполнении команды POP. В этом случае преобразования ведутся в обратном порядке, а таблицы используются с иной сортировкой (правая часть каждой таблицы, оказавшись адресной, сортируется по возрастанию, в то время как левая (это значения строк) оказывается перемешанной). Для сокращения потерь времени, обусловленных введением механизма шифрования данных в стеке, блок расшифрования выполняется независимо от блока шифрования.

Дополнительно стоит отметить, что при вызове функций команды PUSH (и команды POP при возврате из функций) в явном виде в программе не присутствуют, но всегда выполняются аппаратными средствами процессора для сохранения текущего состояния процесса (регистра флагов, указателя команд и регистра CS). При таком неявном выполнении команд, работающих со стеком, следует также выполнять шифрование и расшифрование описанным выше способом.

Такая реализация выполнения команд PUSH и POP позволит защитить программу, потенциально уязвимую к переполнению буфера, от атак по схеме «ret to Lib.c» и в целом обеспечит более строгую работу со стеком — не позволит непосредственно (через значение регистра ESP и смещение относительно него) работать с данными в стеке, если они были помещены туда указанными

командами. Кроме того, замещение адреса возврата в результате использования уязвимости форматной строки со спецификатором «%n» также станет невозможным, так как адрес возврата и другие сохраненные в стеке значения регистров зашифрованы и их непосредственное изменение со стороны злоумышленника не имеет смысла.

Выводы

В статье предлагается модификация процессора, предназначенная для защиты программ от воздействия эксплойтов, действие которых основано на изменении содержимого стека. Данный метод позволяет не только обеспечить контроль за данными, помещаемыми в стек командами PUSH и POP, и тем самым фактически разделить механизмы работы со стеком на два класса (прямой доступ к ячейкам памяти и LIFO-архитектура), но и попутно защитить адрес возврата из функции, который является целью злоумышленника при атаке на уязвимость переполнения буфера. Таким образом, можно считать, что уязвимость переполнения буфера закрывается от всех видов эксплойтов, нацеленных на перезапись адреса возврата.

СПИСОК ЛИТЕРАТУРЫ:

1. *Fayolle P., Glaume V.* A buffer overflow study. Attacks and defenses. URL: http://www.shell-storm.org/papers/files/539.pdf.
2. *Гуров Д. В., Иванов М. А.* Динамическая рандомизация системы команд микропроцессора для защиты от эксплойтов // Вестник Рязанского государственного радиотехнического университета. 2011. № 1 (выпуск 35). С. 81–86.
3. *Barrantes E. G., Ackley D. H., Forrest S. and others.* Randomized Instruction Set Emulation to Disrupt Binary Code Injection Attacks. URL: http://portal.acm.org/citation.cfm?id=948147.
4. *Иванов М. А., Чугунков И. В.* Теория, применение и оценка качества генераторов псевдослучайных последовательностей. М.: Кудиц-Образ, 2003. – 240 с.
5. Linear Feedback Shift Registers. URL: http://homepage.mac.com/afj/lfsr.html.