Классификация ROP гаджетов

А.В. Вишняков <vishnya@ispras.ru>
Институт системного программирования РАН,
109004, Россия, г. Москва, ул. А. Солженицына, д. 25

Аннотация. В данной работе предложен метод классификации ROP гаджетов, который позволяет аналитику сделать вывод о применимости техники ROP для эксплуатации уязвимостей в том или ином случае. Метод, реализованный в виде программного инструмента, применим к бинарным файлам программ. Работа инструмента продемонстрирована на 32-х и 64-х битных приложениях из дистрибутива Ubuntu 14.04, а возможность применения ROP на основе результатов классификации подтверждена на нескольких примерах.

Ключевые слова: уязвимость; ROP; классификация.

DOI: 10.15514/ISPRAS-2016-28(6)-2

Для цитирования: Вишняков А.В. Классификация ROP гаджетов. Труды ИСП РАН, том 28, вып. 6, 2016, стр. 27-36. DOI: 10.15514/ISPRAS-2016-28(6)-2

1. Введение

Уязвимость переполнения буфера на стеке имеет широкую распространенность. Эксплуатация данной уязвимости может привести к перехвату потока управления [1]. Обычно перехват потока управления производится при помощи передачи управления на некоторую вредоносную нагрузку. Для формирования такой нагрузки может быть использована техника возвратно-ориентированного программирования (ROP), применимая в условиях работы современных защитных механизмов. Нагрузка формируется из последовательности ROP гаджетов, которые представляют собой набор инструкций, заканчивающийся инструкцией передачи управления.

Важной задачей является исследование программ на предмет применимости ROP. В данной работе предложен метод классификации ROP гаджетов, который позволяет аналитику сделать вывод о том, возможна ли эксплуатация в том или ином случае.

2. Предпосылки к появлению ROP

Получив контроль над значениями, записываемыми в буфер, можно перезаписать адрес возврата из функции указателем на размещенный на стеке вредоносный код. Чтобы избежать негативных последствий от переполнения буфера на стеке, появились различные защитные механизмы.

Предотвращение выполнения данных (DEP) – механизм защиты, запрещающий исполнение кода из областей памяти, помеченных как «данные». Механизм успешно применяется в операционных системах (Windows, Linux и др.) и предотвращает выполнение кода, размещенного на стеке. В ответ на DEP появилась атака возврата в библиотеку, позволяющая обойти DEP. Атака заключается в подмене адреса возврата адресом некоторой библиотечной функции, например, функции system из библиотеки libc.

Механизм рандомизации адресного пространства (ASLR) позволяет разместить важные структуры программы (стек, куча, образ программы, динамические библиотеки) по различным адресам во время каждого запуска программы. Данная защита противодействует проведению атаки возврата в библиотеку, т.к. адрес библиотечной функции неизвестен до выполнения программы. Однако для рандомизации адреса загрузки образа программы или библиотек они должны быть скомпилированы с соответствующими флагами компилятора, что не всегда выполняется. Так в Linux адрес загрузки большинства исполняемых файлов остается нерандомизированным, что оставляет пути для эксплуатации.

Возвратно-ориентированное программирование (ROP) [2] является развитием метода эксплуатации путем возврата в библиотеку. ROP так же, как и атака возврата в библиотеку позволяет перехватывать поток управления для эксплуатации в условиях работы DEP. Но метод представляет большую опасность: он может быть применен в условиях работы ASLR для нерандомизированных исполняемых файлов. Метод использует последовательности инструкций, заканчивающиеся инструкцией передачи управления, из нерандомизированных исполняемых секций программы. Такие последовательности инструкций называются гаджетами. Гаджеты собираются в цепочки, и их адреса размещаются от адреса возврата на стеке так, чтобы первый гаджет передавал управление второму, второй – третьему и т.д. Таким образом, с помощью цепочки гаджетов можно выполнить некоторые вредоносные действия. Например, ROP-цепочка на рис. 1 производит запись произвольного значения по произвольному адресу.

Следует отметить, что применение ROP для эксплуатации уязвимости возможно при условии наличия достаточного для осуществления эксплуатации набора гаджетов из нерандомизированных областей памяти. Выделение гаджетов, пригодных для составления ROP-цепочек, может быть автоматизировано, что достигается путем поиска и последующей классификации гаджетов.

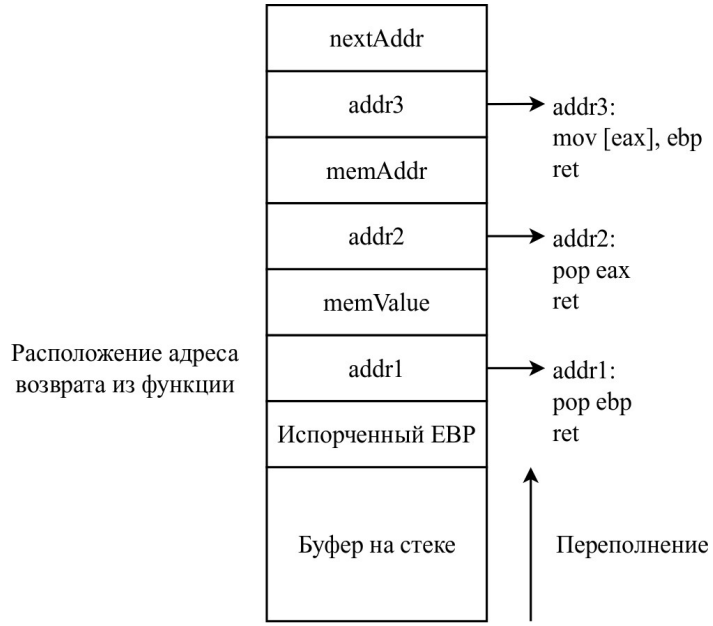

Рис. 1. Состояние стека после размещения ROP-цепочки, которая записывает значение memValue по адресу memAddr и передает управление на nextAddr

Fig. 1. The stack state after setting a ROP-chain which stores memValue to memAddr and transfers control to nextAddr

3. Поиск гаджетов

Для поиска гаджетов может быть использован алгоритм Галилео [2], который осуществляет поиск инструкций передачи управления в исполняемых секциях программы. Для каждой найденной инструкции производится дизассемблирование нескольких байт, предшествующих инструкции. Все корректно дизассемблированные последовательности инструкций добавляются в набор найденных гаджетов.

4. Классификация гаджетов

Классификация гаджетов позволяет повысить уровень представления результатов поиска гаджетов. Классификация производится на основе дополненного набора типов гаджетов, предложенного в Q [3] (табл. 1). Каждый тип гаджета определяется некоторой параметризованной семантикой. Задача классификации гаджетов заключается в сопоставлении каждого гаджета удовлетворяющим его семантике классам и их параметрам. Например, гаджет «mov ecx, edx ; mov edi, esi ; xor eax, eax ; ret» удовлетворяет следующим классам: «MoveRegG : ecx ← edx», «MoveRegG : edi ← esi», «InitConstG : eax ← 0».

Табл. 1. Типы гаджетов. [Addr] означает доступ к памяти по адресу Addr, ∘ – бинарную операцию. a ← b означает, что конечное значение a равно начальному значению b. X ∘← Y – сокращение для X ← X ∘ Y.

Table 1. Gadget types. [Addr] means accessing memory at address Addr, ∘ – binary operation. a ← b means that final value of a equals initial value of b. X ∘← Y is short for X ← X ∘ Y.

Тип	Параметры	Семантика
NoOpG	N/A	Не меняет ничего в памяти и на регистрах
JumpG	AddrReg	IP ← AddrReg
MoveRegG	InReg, OutReg	OutReg ← InReg
LoadConstG	OutReg, Offset	OutReg ← [SP + Offset]
ArithmeticG	InReg1, InReg2, OutReg, ∘	OutReg ← InReg1 ∘ InReg2
LoadMemG	AddrReg, OutReg, Offset	OutReg ← [AddrReg + Offset]
StoreMemG	AddrReg, InReg, Offset	[AddrReg + Offset] ← InReg
ArithmeticLoadG	AddrReg, OutReg, Offset, ∘	OutReg ∘← [AddrReg + Offset]
ArithmeticStoreG	AddrReg, InReg, Offset, ∘	[AddrReg + Offset] ∘← InReg
InitConstG	OutReg, Value	OutReg ← Value
ShiftStackG	Offset, ∘ (+ / –)	SP ∘← Offset
ArithmeticStackG	InReg, ∘ (+ / –)	SP ∘← InReg

Для использования гаджетов в целях эксплуатации необходимо, чтобы они удовлетворяли следующим требованиям:

- Гаджет может изменять указатель стека только на константное значение, исключение – ArithmeticStackG.
- Побочные эффекты выполнения гаджета не должны приводить к неконтролируемому поведению программы. Например, запись значения по произвольному адресу памяти может привести к аварийному завершению программы.

Результатами классификации гаджета являются:

- типы и параметры гаджета;
- список регистров, значения которых не сохраняются в результате выполнения гаджета;
- информация о фрейме гаджета (размер фрейма, смещение ячейки с адресом следующего гаджета относительно указателя стека после передачи управления текущему гаджету). Например, гаджет «pop rax ; ret 16» имеет размер фрейма 32, а адрес следующего гаджета размещается по смещению 8.

5. Метод классификации и детали реализации

Набор гаджетов для классификации получается при помощи инструмента с открытым исходным кодом ROPgadget [4], в котором реализован алгоритм Галилео поиска гаджетов.

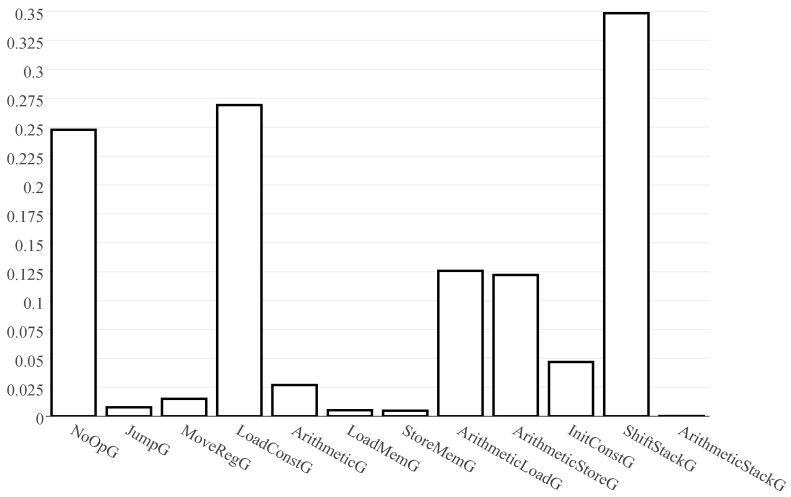

Рис. 2. Отношение числа классифицированных гаджетов данного типа к общему числу найденных гаджетов для архитектуры x86

Fig. 2. The ratio of classified gadgets of a certain type to all found gadgets for x86 architecture

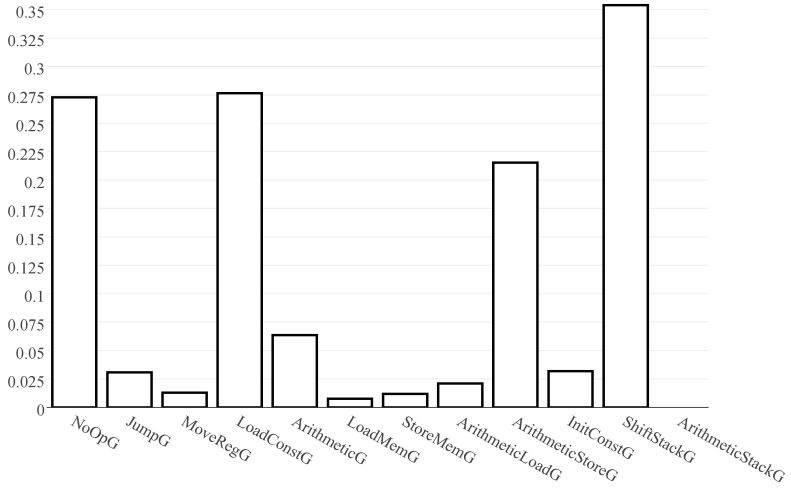

Рис. 3. Отношение числа классифицированных гаджетов данного типа к общему числу найденных гаджетов для архитектуры x86-64

Fig. 3. The ratio of classified gadgets of a certain type to all found gadgets for x86-64 architecture

Классификация гаджета производится на основе анализа эффектов выполнения гаджета на конкретных входных данных. Инструкции гаджета транслируются в промежуточное представление Pivot [5]. Далее запускается процесс интерпретации Pivot-представления. Во время интерпретации отслеживаются обращения к регистрам и памяти. Если происходит первое чтение регистра или области памяти, считанное значение генерируется случайным образом. В результате интерпретации будут получены начальные и конечные значения регистров и памяти. На основе этой информации делается вывод о возможной принадлежности гаджета тому или иному типу. Например, для принадлежности типу MoveRegG должна существовать такая пара регистров, что начальное значение первого регистра равно конечному значению второго. В результате анализа составляется список всех удовлетворяющих гаджету классов и их параметров (список кандидатов). Затем производится еще несколько запусков конкретного выполнения с различными входными данными, в результате которых из списка кандидатов удаляются ошибочно определенные классы.

Предложенный метод основывается на результатах выполнения гаджета на ограниченном количестве наборов конкретных входных данных, что в общем случае не гарантирует соответствие семантике результата выполнения гаджета на произвольных входных данных. Таким образом, возможна неверная классификация гаджета. Однако доля неверно классифицированных гаджетов на практике незначительна. Метод был реализован в виде программного инструмента.

6. Апробация метода

Метод был апробирован на исполняемых файлах из каталога /usr/bin дистрибутива операционной системы Ubuntu 14.04. Классификатор гаджетов показал быструю работу на приложениях архитектур x86 и x86-64. Статистика по распределению типов гаджетов приводится на рис. 2-3.

С использованием классификатора гаджетов была подтверждена возможность эксплуатации следующих примеров в условиях работы современных защитных механизмов.

Эксплуатация DEP и ASLR. На основе результата классификации гаджетов, найденных в 32-х битной программе zsnes с уязвимостью переполнения буфера на стеке (strcpy), была вручную составлена ROP-цепочка, вызывающая интерпретатор командной строки (табл. 2).

Для вызова интерпретатора командной строки используется функция system из библиотеки libc. Так как адрес функции system рандомизирован, используется техника возврата в рандомизированную libc [6]. Механизм ленивого связывания в Linux осуществляет загрузку адреса библиотечной функции в GOT при первом ее вызове. Zsnes не импортирует функцию system, поэтому ее адрес вычисляется во время выполнения программы с помощью ROP, используя адрес какой-либо другой функции, чей адрес уже загружен в

GOT к моменту переполнения. В приведенном примере такой функцией является функция access. Для вычисления адреса используется предположение о том, что смещение двух функций из libc относительно друг друга остается постоянным. Тогда адрес system вычисляется как сумма фактического адреса access, считанного из GOT, и смещения функции system относительно access в библиотеке libc.

Табл. 2. ROP-цепочка для zsnes. Got_addr – адрес GOT, access_got_addr – адрес ячейки GOT с адресом функции access

Table 2. ROP-chain for zsnes. Got_addr – address of GOT, access_got_addr – access GOT entry address

`eax = got_addr`	`LoadConstG : eax ← [esp]`
`ecx = “\xcdbin”`	`LoadConstG : ecx ← [esp]`
`cl += ch  # ecx = “/bin”`	`ArithmeticG : cl ← cl + ch`
`[eax] = ecx`	`StoreMemG : [eax] ← ecx`
`eax = got_addr + 4`	`LoadConstG : eax ← [esp]`
`ecx = 0xf837acd7`	`LoadConstG : ecx ← [esp]`
`[eax] = ecx`	`StoreMemG : [eax] ← ecx`
`[eax] += eax  # [eax] = “/sh\x00”`	`ArithmeticStoreG : [eax] +← eax`
`ecx = access_got_addr`	`LoadConstG : ecx ← [esp]`
`eax = &system - &access`	`LoadConstG : eax ← [esp]`
`eax += [ecx]`	`ArithmeticLoadG : eax +← [ecx]`
`jmp eax`	`JumpG : eip ← eax`

Также необходимо разместить строковый аргумент «/bin/sh» в глобальной памяти программы. В рассматриваемом примере строка размещается в первых двух ячейках GOT. Во время составления цепочки следует учесть ограничения, накладываемые структурой программы на символы ROP-цепочки. В zsnes поток управления зависит от входных данных – последний символ ‘/’ или ‘\’ заменяется нулем, следовательно, ROP-цепочка не должна содержать данных символов. Более того, ROP-цепочка не должна содержать нулевых байтов, т.к. она является частью нуль-терминированной строки.

Для полученной ROP-цепочки удалось продемонстрировать эксплуатацию уязвимости переполнения буфера на стеке в условиях одновременного функционирования защит DEP и ASLR.

Эксплуатация DEP, ASLR и «канарейки». Если в программе присутствует уязвимость переполнения буфера на стеке при условии «write-what-where» [7], то возможна перезапись ячейки таблицы GOT некоторой вызываемой функции адресом гаджета-трамплина (ShiftStackG). После вызова функции гаджет-трамплин передаст управление на оставшуюся часть ROP-цепочки. С помощью классификатора гаджетов в модельном примере был успешно

найден гаджет ShiftStackG для передачи управления на ROP-цепочку, что позволило подтвердить возможность эксплуатации уязвимости в присутствии «канарейки», DEP и ASLR.

7. Заключение

В статье представлен метод классификации ROP гаджетов. Применение предложенного метода позволяет выявить среди обнаруженных в программах уязвимостей переполнения буфера на стеке наиболее приоритетные для исправления, а именно, те уязвимости, для которых возможна эксплуатация. Предложенный метод был реализован в виде программного инструмента.

Возможными направлениями для дальнейших исследований являются дедуктивная верификация семантики гаджета и оценка эффективности современных защит от ROP.

Список литературы

[1]. One A. Smashing the stack for fun and profit. Phrack magazine, v. 7, №. 49, 1996, pp. 14-16.

[2]. Hovav Shacham. The Geometry of Innocent Flash on the Bone: Return-into-libc without Function Calls (on the x86). 2007 ACM Conference on Computer and Communications Security (CCS), Proceedings of CCS 2007, pp. 552-561.

[3]. Edward J. Schwartz, Thanassis Avgerinos, David Brumley. Q: Exploit Hardening Made Easy. 2011 Usenix Security Symposium (SEC), Proceedings of SEC 2011.

[4]. Инструмент ROPgadget. https://github.com/JonathanSalwan/ROPgadget

[5]. V. A. Padaryan, M. A. Solovyev, A. I. Kononov. Simulation of operational semantics of machine instructions. Program. Comput. Software, vol. 37, № 3, 2011, pp. 161-170. DOI: 10.1134/S0361768811030030

[6]. G. F. Roglia, L. Martignoni, R. Paleari, D. Bruschi. Surgically Returning to Randomized lib(c). 2009 Annual Computer Security Applications Conference (ACSAC), Proceedings of ACSAC 2009, pp. 60-69.

[7]. CWE-123: Write-what-where Condition. http://cwe.mitre.org/data/definitions/123.html

Classification of ROP gadgets

A.V. Vishnyakov <vishnya@ispras.ru>
Institute for System Programming of the Russian Academy of Sciences,
25, Alexander Solzhenitsyn st., Moscow, 109004, Russia.

Abstract. Return-oriented programming (ROP) is a dangerous exploitation technique which can be used to bypass modern defense mechanisms. ROP reuses code chunks ending with control transfer instruction from a program binary to form a chain corresponding some payload. These code chunks are called gadgets. Though, a certain set of gadgets should be available to exploit a vulnerability. Determining gadgets that can be used to form a ROP chain can be done by gadgets search and classification. This paper introduces a method for ROP gadgets classification that allows one to evaluate whether or not ROP technique can be used to exploit a program vulnerability. Classification is based on side-effects analysis of gadget execution with concrete inputs. Gadget instructions are translated into IR which is interpreted to track registers and memory usage. Initial registers and memory values are randomly generated. According to initial and final values of registers and memory gadget semantics can be explored. Classification performs several executions to determine gadget semantics. Proposed method is applied to program binaries and its capabilities were demonstrated on 32-bit and 64-bit binaries from Ubuntu 14.04. Using classification results program exploitability was confirmed for several examples. Furthermore, a possible exploitation of stack buffer overflow vulnerability in presence of write-what-where condition was shown on a model example demonstrating a bypass of canary, DEP and ASLR.

Keywords: vulnerability; ROP; classification.

DOI: 10.15514/ISPRAS-2016-28(6)-2

For citation: Vishnyakov A.V. Classification of ROP gadgets. Trudy ISP RAN/Proc. ISP RAS, vol. 28, issue 6, 2016, pp. 27-36 (in Russian). DOI: 10.15514/ISPRAS-2016-28(6)-2

References

[1]. One A. Smashing the stack for fun and profit. Phrack magazine, vol. 7, №. 49, pp. 14-16.
[2]. Hovav Shacham. The Geometry of Innocent Flash on the Bone: Return-into-libc without Function Calls (on the x86). 2007 ACM Conference on Computer and Communications Security (CCS), Proceedings of CCS 2007, pp. 552-561.
[3]. Edward J. Schwartz, Thanassis Avgerinos, David Brumley. Q: Exploit Hardening Made Easy. 2011 Usenix Security Symposium (SEC), Proceedings of SEC 2011.
[4]. ROPgadget. https://github.com/JonathanSalwan/ROPgadget
[5]. V. A. Padaryan, M. A. Solovyev, A. I. Kononov. Simulation of operational semantics of machine instructions. Program. Comput. Software, vol. 37, № 3, 2011, pp. 161-170. DOI: 10.1134/S0361768811030030

[6]. G. F. Roglia, L. Martignoni, R. Paleari, D. Bruschi. Surgically Returning to Randomized lib(c). 2009 Annual Computer Security Applications Conference (ACSAC), Proceedings of ACSAC 2009, pp. 60-69.
[7]. CWE-123: Write-what-where Condition. http://cwe.mitre.org/data/definitions/123.html