ПРОБЛЕМЫ ЯДЕРНОЙ, РАДИАЦИОННОЙ И ЭКОЛОГИЧЕСКОЙ БЕЗОПАСНОСТИ

УДК 004.056.53

АНАЛИЗ СУЩЕСТВУЮЩИХ СРЕДСТВ ЗАЩИТЫ ОТ ПЕРЕПОЛНЕНИЯ БУФЕРА НА СТЕКЕ И СПОСОБЫ ИХ ОБХОДА

© 2019 М.А. Паринов

Национальный исследовательский ядерный университет «МИФИ», Москва, Россия

Проблема выявления и предотвращения атак на приложения была и остается одной из актуальных задач информационной безопасности. Изъяны в коде программ приводят к нарушению нормальной работы программного обеспечения. Из-за недочетов разработки могут возникать нарушения целостности, доступности и конфиденциальности данных, прерывание выполнения запущенных процессов или даже системы в целом. В данной работе рассматривается механизм совершения переполнения буфера на стеке, а также существующие современные средства обнаружения или предотвращения переполнения буфера, такие как ASLR, StackGuard и неисполняемый стек. Данные средства защиты выбраны в качестве цели исследования из-за того, что они являются самыми распространёнными и являются встроенными средствами защиты в ОС Linux. Целью работы является анализ проблемы переполнения буфера и неполной эффективности, существующих повсеместно используемых средств предотвращения и обнаружения данного типа атак, а также описание альтернативного способа решения проблемы переполнения буфера. В рамках работы для каждого из широко распространенных средств защиты рассмотрен способ его обхода. Итогом данной работы стало заключение, что существующие средства защиты имеют существенные недостатки и поэтому требуется разработка дополнительного средства защиты, идея которого предложена в конце статьи.

Ключевые слова: переполнение буфера, системные вызовы, инъекции кода, неисполняемый стек, StackGuard, ASRL, информационная безопасность.

Поступила в редакцию: 12.04.2019
После доработки 22.04.2019
Принята к публикации 26.04.2019

ВВЕДЕНИЕ

Проблема выявления и предотвращения атак на приложения была и остается одной из актуальных задач информационной безопасности. Корректная разработка приложения является одним из способов решения данной проблемы, так как небольшая ошибка разработчика в умелых руках злоумышленника может привести к непредсказуемым последствиям.

Изъяны в коде программ приводят к нарушению нормальной работы программного обеспечения. Из-за недочетов разработки могут возникать нарушения целостности, доступности и конфиденциальности данных, прерывание выполнения запущенных процессов или даже системы в целом. Большинство уязвимостей программного обеспечения связано с неправильной обработкой данных, получаемых из сторонних (внешних) источников, или недостаточно строгой их проверкой.

Первый случай использования переполнения стекового буфера произошел более тридцати лет назад вредоносной программой названной Червем Мориса в 1988 году. Он использовал переполнение буфера в демоне UNIX для перемещения от одного компьютера к другому. С тех пор было придумано множество механизмов защиты, но по сей день, проблема остается актуальной.

ПЕРЕПОЛНИЕ БУФЕРА

Для начала рассмотрим механизм переполнения буфера. На рисунке 1 приведен простейший пример уязвимой программы на языке программирования С. Ошибкой в этом примере является отсутствие проверки длины входных данных считываемых в строковую переменную buf [1].

```
#include <stdio.h>
#include <string.h>
int main(int argc, char** argv)
{
char buf[200];
strcpy(buf, argv[1]);
return 0;
}
```

Рисунок 1 – Пример программы на языке С уязвимой к переполнению буфера на стеке [C language program example vulnerable to buffer overflow on the stack]

Если на ввод подать строку длиной менее 200 байт, то программа работает корректно [2-3]. Однако если подать строку размером больше буфера, то получится ошибка сегментации [4-5]. На рисунке 2 изображен момент атаки путем переполнения стекового буфера.

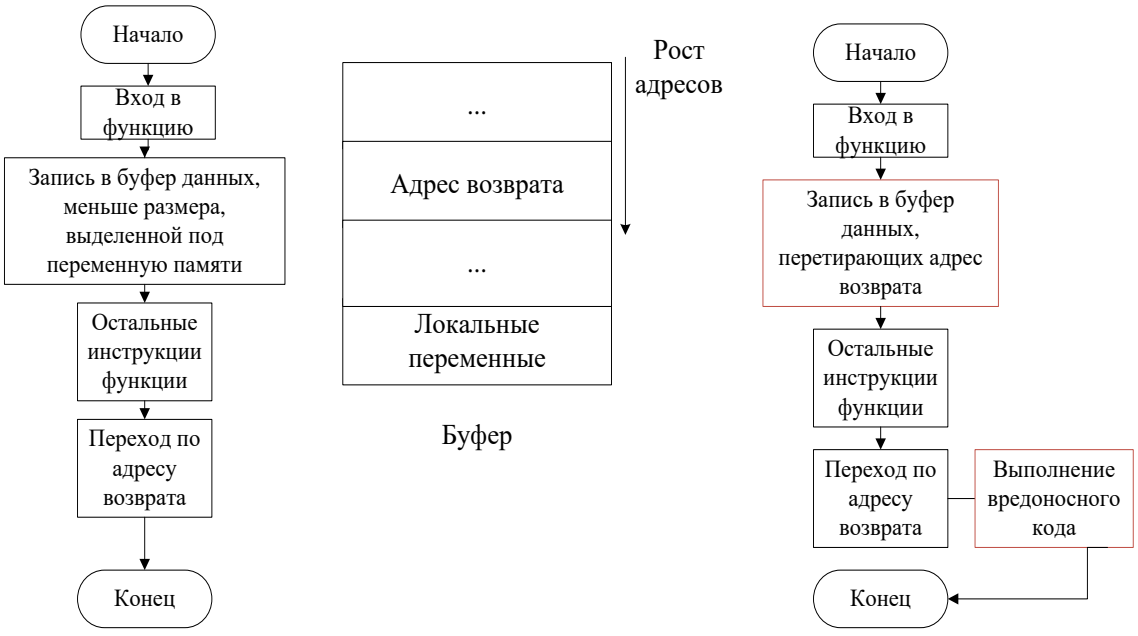

Рисунок 2 – Алгоритмы нормального выполнения функции и выполнения функции в момент атаки путем переполнения стекового буфера [Algorithms of normal function execution and its execution at the moment of an attack by stack buffer overflow]

STACKGUARD

StackGuard был выпущен в свет в 1998 г., как расширение к GCC для защиты программного обеспечения от переполнения буфера на стеке. Целью StackGuard является проверка целостности адреса возврата, который злоумышленник попытается переписать адресом нужного ему исполняемого кода [6].

Для своевременной проверки изменения адреса возврата, она должна происходить перед возвратом функции. Для этого StackGuard помещает контрольную информацию, так называемую канарейку (Canary word), непосредственно перед адресом возврата (рис. 2). Таким образом, при попытке перезаписать адрес возврата, злоумышленник автоматически перепишет канарейку, проверка которой происходит непосредственно перед передачей управления по адресу возврата [7].

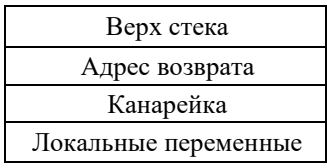

Рисунок 3 – Пример расположения канарейки в стеке [An example of Canary word location on the stack]

Существует три типа канареек.

Канарейка с использованием символов конца строки – подразумевает, что большинство переполнений происходит на строках. Поэтому используя четыре разных символа конца строки можно предотвратить разовое переписывание памяти, так как один или несколько символов прекратят строковую операцию. Однако если атакующий сможет переписать стек несколько раз, то он сможет заменить защищенную контрольную информацию и адрес возврата.

Случайная канарейка – обнаруживает все перезаписи памяти при условии, что атакующий не знает значение канарейки. Значение хранится в глобальной переменной, которая инициализируется в новое значение при каждом запуске программы.

Случайная XOR канарейка – используется при условии, если злоумышленник может переписывать память в случайном месте. В дополнении к случайной канарейке часть или все защищаемые данные шифруются с использованием исключающего «или» с значением канарейки. Уязвимости аналогичны случайной канарейке.

ОБХОД STACKGUARD

Существуют несколько слабостей реализации.

Во-первых, в угоду производительности также существуют незащищенные функции. Например, компилятор gcc защищает функции, имеющие строковые буферы размером более 4 элементов.

Во-вторых, если в системе существует уязвимость утечки памяти, то в независимости от типа канарейки атакующий может переписать указатель предыдущего фрейма, где будет располагаться адрес возврата кода атакующего и правильная канарейка.

Рассмотрим возврат с изменённым указателем предыдущего фрейма более подробно.

По умолчанию StackGuard использует прерывающую канарейку. Ее значение 0x000aff0d. Таким образом, мы не можем переписать адрес возврата. Однако мы можем переписать указатель предыдущего фрейма. Это означает, что мы можем в стеке или куче разместить конструкцию вида (0x000aff0d; Новый адрес возврата) и при возврате к вызвавшей функции указатель фрейма будет смещен на нужный нам адрес и после ее выполнения канарейка останется неизменной, но адрес возврата будет модифицирован адресом известного нам заранее приготовленного вредоносного кода [8].

Также в качестве альтернативных методах обхода стоит упомянуть о таких способах как предугадывание канарейки и использование слабостей реализации.

НЕИСПОЛНЯЕМЫЙ СТЕК

Переполнение буфера обычно производится путем инъекции кода в уязвимое приложение, к которому перейдет управление после того как будет переписан изначальный адрес возврата [9]. Этот код обычно помещается в кучу или стек. Что бы предотвратить выполнение вредоносного кода, все страницы данных процесса помечаются как неисполняемые. Страницы, содержащие код, помечаются как защищенные от записи, поэтому злоумышленник не может модифицировать существующий или вставить свой код. Такая технология так же известна как неисполняемый стек (Data Execution Prevention-DEP).

Одним из достоинств данной технологии является то, что она не влияет на производительность. Так же стоит заметить, что DEP можно применить к уже существующим программам, однако могут возникнуть проблемки с JIT (Just-in-time compilation- компиляции непосредственно во время работы программы) компиляторами (например, Java) которые генерируют исполняемый код в памяти во время исполнения, что требует разрешения и на запись в память и на исполнение [10].

ОБХОД С ВОЗВРАТОМ В БИБЛИОТЕКУ

С тех пор, как появился DEP, проведение атак с внедрением произвольного кода стало почти невозможным, но атакующие нашли другой способ обхода данный защиты. Основная идея заключается в том, что большинство библиотек уже размещены в памяти процесса и содержат большое количество кода, который злоумышленник может использовать по своему желанию. Базовым примером может служить функция system, с помощью которой злоумышленник может запустить командную оболочку linux (sh), непосредственно в которой он может исполнить произвольный код (ret2libc). С тех пор как функция system является частью библиотеки libc, атакующему больше не требуется внедрять свой собственный код, а можно использовать существующие возможности библиотечных функций. Другим вариантом обхода является использование функции mprotect для пометки стека или кучи как исполняемой и уже использовать расположенный в ней атакующим вредоносный код [11].

На основании атаки ret2libc появилась еще одна техника, называемая Return Oriented Programming (Возвратно ориентированное программирование).

Одним из способов защиты от Ret2libc является удаление потенциально опасных функций, однако для ROP это не имеет значения, если быть точнее, то для совершения данной атаки даже не нужно вызывать какие-либо функции. В концепции данной атаки мы будем использовать небольшие последовательности ассемблерных команд, называемых гаджетами, расположенных в исполняемом файле либо в библиотеке, связанной с ним. В общем, вместо возврата к адресу функции в библиотеке мы будем использовать возврат к данным последовательностям [12].

Для того что бы найти данные гаджеты мы должны найти в уязвимом приложении все ret инструкции и посмотреть предыдущие инструкции на наличие нужных нам ассемблерных инструкций.

Данные гаджеты могут использоваться для следующих действий:

– Загрузка константы в регистр – позволяет сохранить верхнее значение в стеке в регистр eax, используя pop инструкцию. Такая последовательность выглядит как Pop eax;ret;. Эта последовательность позволит нам передать управление к следующему гаджету.

– Чтение из памяти – например, mov ecx,[eax];ret – переписывает ecx значением, расположенным по адресу, хранящемуся в eax.

– Запись в память – например, mov [eax],ecx;ret.

– Арифметические операции, такие как сложение, умножение, исключающее или и т.д., например xor eax,ebx;ret.

– Системные вызовы – инструкции, позволяющие нам выполнить прерывание, например, int 0x80;ret или call gs:[0x10];ret

Например, вредоносный код, построенный на основе таких гаджетов и изображенный на рисунке 4 [9] запускает sh с помощью системного вызова execve.

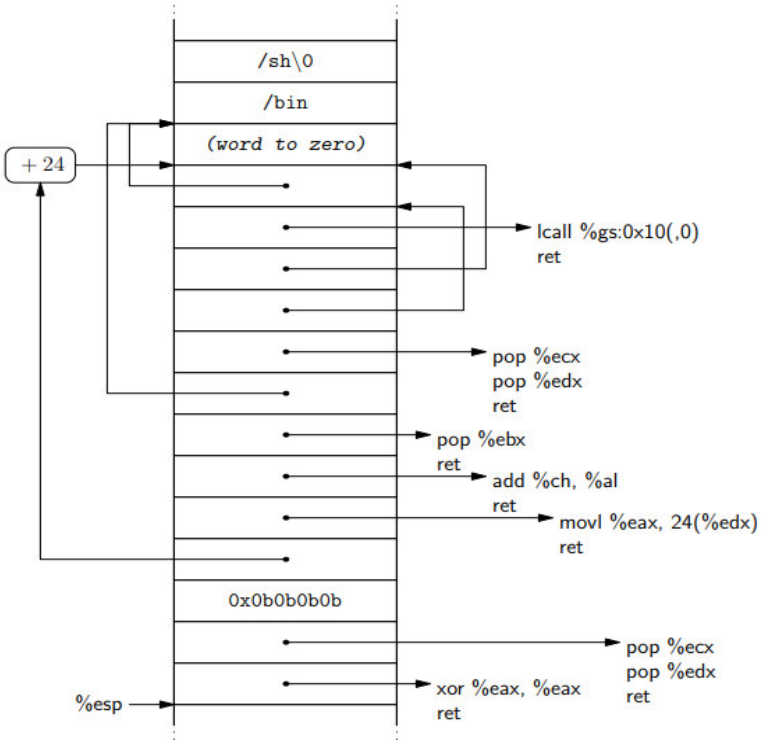

Рисунок 4 – Вредоносный код, запускающий sh, собранный из гаджетов, найденных в уязвимом приложении [Malicious code that runs *sh*, compiled from gadgets found in a vulnerable application]

ASLR

Address Space Layout Randomization – широко распространённая технология, которая защищает систему от определенного типа атак путем рандомизации позиции ключевых компонентов программы в виртуальной памяти. Защите подлежат адреса сегментов данных и кода, стека, кучи, а также используемых библиотек [13]. Для примера, если атакующий смог успешно нарушить поток управления программы, ему будет сложно использовать возвратно ориентированное программирование при наличии ASLR, потому что адреса гаджетов для инъекции в стек будут не известны из-за рандомизации [14]. Использование подбора для поиска адреса нужного гаджета может привести к краху программы или занять большое количество времени.

За время существования ASLR было разработано множество вариантов обхода, среди которых стоит выделить:

– data leak – дополнительная уязвимость утечки данных позволяющие злоумышленнику получить нужные адреса;

– Относительная адресация;

– Слабости реализации;

– Побочные эффекты работы аппаратуры [15].

Примером обхода ASLR может являться техника схожая с ret2libc, которая называется ret2plt. PLT- Procedure Linkage Table состоит из адресов внешних функций, чьи адреса не известны во время компиляции и составляется динамическим линковщиком во время исполнения. Стоить заметить, что нахождение таблицы PLT не меняется при использовании ASLR, что позволяет нам легко вызвать любую из функций используемых уязвимой программой функций с возвратом в PLT вместо стандартного возврата в libc [16] [17].

ЗАКЛЮЧЕНИЕ

В данной работе рассмотрен простейший способ совершения атак с использованием переполнения буфера, а также существующие средства по обнаружению и предотвращению атак на приложения. Ни один из представленных и широко используемых механизмов защиты не может гарантировать предотвращение атак на приложения ввиду слабостей своей реализации. Например, способом обхода StackGuard является техника перетирания указателя текущего фрейма. Для неисполняемого стека предложена технология возврата в библиотеку (ret2libc) основанная на вызове системных функций и ее модификации. Для обхода ASLR используются техника схожая с ret2libc, но с возвратом в Procedure Linkage Table. Данные способы обходов представленных систем защиты далеко не единственные и существует еще множество способов обхода, в том числе способов обхода сразу нескольких систем предотвращения и обнаружения атак на приложения. Стоит заметить, что данные системы сильно уменьшают возможность проведения атак, однако не известно, сколько возможностей остается для совершения атаки, что приводит к выводу, что для обеспечения полной безопасности требуется или усовершенствовать существующие системы, или создавать новую в дополнение к текущим.

В качестве системы по предотвращению атак на приложения предлагается создать программный комплекс по предотвращению атак на приложения путем проверки совершаемых ими системных вызовов. Проверка корректности совершаемого системного вызова предлагается строить на основе эталонной модели защищаемого приложения. Для получения эталонной модели поведения защищаемого конкретного приложения предлагается анализировать его исходные коды и собирать информацию о номерах и порядке совершаемых им системных вызовов. Такой метод защиты поможет предотвратить класс атак переполнения буфера приводящих к инъекции вредоносного кода в ОС Linux.

СПИСОК ЛИТЕРАТУРЫ

1. Aleph One Smashing The Stack For Fun And Profit [Электронный ресурс]. URL: http://www-inst.eecs.berkeley.edu/~cs161/fa08/papers/stack_smashing.pdf (дата обращения: 13.07.2018).
2. *Альфред, В. Ахо* Компиляторы. Принципы, технологии и инструментарий / В. Ахо Альфред, С. Лам Моника, Сети Рави, Д. Ульман Джеффри; пер. с англ. И. Красиков. – Москва : Вильямс, 2008 – 1184 с.
3. Donald E. Porter, Emmett Witchel. Transactional system calls on Linux [Электронный ресурс]. URL : http://www.cs.unc.edu/~porter/pubs/ols10.pdf (дата обращения: 15.09.2018).
4. Michal Sojka. Kernel side of system calls [Электронный ресурс]. URL: http://labe.felk.cvut.cz/~stepan/33OSD/files/osd-e3-kern-syscall.pdf (дата обращения: 12.08.2018).
5. *Стюгин, М.А.* Способ построения программного кода с неразличимой функциональностью. [Электронный ресурс] / М.А. Стюгин // Безопасность информационных технологий. – 2017. – Вып. 24. – № 1. – С. 66-72. ISSN 2074-7136. URL: https://bit.mephi.ru/index.php/bit/article/view/57 (дата обращения: 1.11.2018) doi:http://dx.doi.org/10.26583/bit.2017.1.08.
6. Crispan Cowan, Calton Pu, Dave Maier, Jonathan Walpole, Peat Bakke, Steve Beattie, Aaron Grier, Perry Wagle, and Qian Zhang «StackGuard: Automatic Adaptive Detection and Prevention of Buffer-Overflow Attacks» [Электронный ресурс]. URL: https://www.usenix.org/legacy/publications/library/proceedings/sec98/full_papers/cowan/cowan.pdf (дата обращения: 1.09.2018).
7. Perry Wagle, Crispin Cowan «StackGuard: Simple Stack Smash Protection for GCC» [Электронный ресурс]. URL: ftp://gcc.gnu.org/pub/gcc/summit/2003/Stackguard.pdf (дата обращения: 30.07.2018).
8. Gerardo Richarte «Four different tricks to bypass StackShield and StackGuard protection» [Электронный ресурс]. URL: https://www.cs.purdue.edu/homes/xyzhang/spring07/Papers/defeat-stackguard.pdf (дата обращения: 6.08.2018).

9. Hovav Shacham. The Geometry of Innocent Flesh on the Bone: Return-into-libc without Function Calls (on the x86) [Электронный ресурс]. URL: https://hovav.net/ucsd/dist/geometry.pdf (дата обращения: 21.10.2018).
10. Erik Buchanan, Ryan Roemer, Stefan Savage, Hovav Shacham «Return-oriented Programming: Exploitation without Code Injection» [Электронный ресурс]. URL: https://www.blackhat.com/presentations/bh-usa-8/Shacham/BH_US_08_Shacham_Return_Oriented_Programming.pdf (дата обращения: 22.10.2018).
11. Erik Buchanan, Ryan Roemer, Hovav Shacham, Stefan Savage «When Good Instructions Go Bad: Generalizing Return-Oriented Programming to RISC» [Электронный ресурс]. URL: http://cseweb.ucsd.edu/~savage/papers/CCS08GoodInstructions.pdf (дата обращения: 22.10.2018).
12. Reed Hastings, Bob Joyce «Purify: Fast Detection of Memory Leaks and Access Errors» [Электронный ресурс]. URL: https://web.stanford.edu/class/cs343/resources/purify.pdf (дата обращения: 23.10.2018).
13. Tyler Durden «Bypassing PaX ASLR protection» [Электронный ресурс]. URL: http://phrack.org/issues/59/9.html (дата обращения: 9.11.2018).
14. Hector Marco, Ismael Ripoll «AMD Bulldozer Linux ASLR weakness: Reducing entropy by 87.5%» [Электронный ресурс]. URL: http://hmarco.org/bugs/AMD-Bulldozer-linux-ASLR-weakness-reducing-mmaped-files-by-eight.html (дата обращения: 9.11.2018).
15. Tilo Muller «ASLR Smack & Laugh Reference» [Электронный ресурс]. URL: https://ece.uwaterloo.ca/~vganesh/TEACHING/S2014/ECE458/aslr.pdf (дата обращения: 10.11.2018).
16. Ralf Hund, Carsten Willems, Thorsten Holz «Practical Timing Side Channel Attacks Against Kernel Space ASLR» [Электронный ресурс]. URL: https://www.ieee-security.org/TC/SP2013/papers/4977a191.pdf (дата обращения: 10.11.2018).
17. Dmitry Evtyushkin, Dmitry Ponomarev, Nael Abu-Ghazaleh «Jump Over ASLR: Attacking Branch Predictors to Bypass ASLR» [Электронный ресурс]. URL: http://www.cs.ucr.edu/~nael/pubs/micro16.pdf (дата обращения: 01.12.2018).

REFERENCES

[1] Aleph One. Smashing The Stack For Fun And Profit. URL: http://www-inst.eecs.berkeley.edu/~cs161/fa08/papers/stack_smashing.pdf.
[2] Al`fred V. Axo, S. Lam Monika, Seti Ravi, D. Ul`man Dzheffri. Kompilyatory`. Principy`, texnologii i instrumentarij. Moskva. Vil`yams [Compilers: Principles, Techniques and Tools]. 2013. 1040 p. (in Russian).
[3] Donald E. Porter, Emmett Witchel. Transactional system calls on Linux. URL: http://www.cs.unc.edu/~porter/pubs/ols10.pdf (in Russian).
[4] Michal Sojka. Kernel side of system calls. URL: http://labe.felk.cvut.cz/~stepan/33OSD/files/osd-e3-kern-syscall.pdf.
[5] Styugin M.A. Sposob postroeniya programmnogo koda s nerazlichimoj funkcional`nost`yu. Bezopasnost` informacionny`x texnologij [The Method of Generation Program Code with Indistinguishable Functionality. IT Security]. [S.l.]. V. 24. №. 1. P. 66-72, apr. 2017. ISSN 2074-7136. URL: https://bit.mephi.ru/index.php/bit/article/view/57 (in Russian). doi:http://dx.doi.org/10.26583/bit.2017.1.08.
[6] Crispan Cowan, Calton Pu, Dave Maier, Jonathan Walpole, Peat Bakke, Steve Beattie, Aaron Grier, Perry Wagle, and Qian Zhang. StackGuard: Automatic Adaptive Detection and Prevention of Buffer-Overflow Attacks. URL: https://www.usenix.org/legacy/publications/library/proceedings/sec98/full_papers/cowan/cowan.pdf.
[7] Perry Wagle, Crispin Cowan. StackGuard: Simple Stack Smash Protection for GCC. URL: ftp://gcc.gnu.org/pub/gcc/summit/2003/Stackguard.pdf.
[8] Gerardo Richarte. Four different tricks to bypass StackShield and StackGuard protection. URL: URL: <https://www.cs.purdue.edu/homes/xyzhang/spring07/Papers/defeat-stackguard.pdf (in English).
[9] Hovav Shacham. The Geometry of Innocent Flesh on the Bone: Return-into-libc without Function Calls (on the x86). URL: https://hovav.net/ucsd/dist/geometry.pdf.
[10] Erik Buchanan, Ryan Roemer, Stefan Savage, Hovav Shacham. Return-oriented Programming: Exploitation without Code Injection. URL: https://www.blackhat.com/presentations/bh-usa-08/Shacham/BH_US_08_Shacham_Return_Oriented_Programming.pdf.
[11] Erik Buchanan, Ryan Roemer, Hovav Shacham, Stefan Savage. When Good Instructions Go Bad: Generalizing Return-Oriented Programming to RISC. URL: http://cseweb.ucsd.edu/~savage/papers/CCS08GoodInstructions.pdf.

[12] Reed Hastings, Bob Joyce. Purify: Fast Detection of Memory Leaks and Access Errors. URL: https://web.stanford.edu/class/cs343/resources/purify.pdf.
[13] Tyler Durden. Bypassing PaX ASLR protection. URL: http://phrack.org/issues/59/9.html.
[14] Hector Marco, Ismael Ripoll. AMD Bulldozer Linux ASLR weakness: Reducing entropy by 87.5%. URL: http://hmarco.org/bugs/AMD-Bulldozer-linux-ASLR-weakness-reducing-mmaped-files-by-eight.html.
[15] Tilo Muller. ASLR Smack & Laugh Reference. URL: https://ece.uwaterloo.ca/~vganesh/TEACHING/S2014/ECE458/aslr.pdf.
[16] Ralf Hund, Carsten Willems, Thorsten Holz. Practical Timing Side Channel Attacks Against Kernel Space ASLR. URL: https://www.ieee-security.org/TC/SP2013/papers/4977a191.pdf.
[17] Dmitry Evtyushkin, Dmitry Ponomarev, Nael Abu-Ghazaleh. Jump Over ASLR: Attacking Branch Predictors to Bypass ASLR. URL: http://www.cs.ucr.edu/~nael/pubs/micro16.pdf.

Analysis of Existing Protection Systems from Buffer Overflow and Methods of their Bypass

M.A. Parinov

Institute of Nuclear Physics and Technology (INP&T), National Research Nuclear University «MEPhI», Kashirskoye shosse, 31, Moscow, Russia 115409
[1]ORCID iD: 0000-0002-6947-8753
WoS Researcher ID: G-9341-2019
e-mail: mafimka@gmail.com

Abstract – The issue of detecting and preventing attacks on applications has been and remains one of the urgent tasks of information security. Flaws in the program code lead to disruption of the normal operation of the software. Data integrity, availability and confidentiality of the data, interruption of the execution of running processes or even the system as a whole may occur due to design flaws. This paper discusses the mechanism of buffer overflow on a stack as well as existing modern means of detecting or preventing buffer overflows such as ASLR, StackGuard, and a non-executable stack. These security features are chosen as the research target because they are the most common and are built-in security features in Linux. The objective of the work is to analyze the problem of buffer overflow and the incomplete effectiveness of existing commonly used means of preventing and detecting this type of attack as well as a description of an alternative way to solve the problem of buffer overflow. As part of the work for each of the widespread means of protection considered a way to circumvent it. The result of this work was the conclusion that the existing security tools have significant drawbacks and therefore requires the development of an additional remedy, the idea of which is proposed at the end of the article.

Keywords: buffer overflow, system calls, code injection, Data Execution Prevention, ASLR, StackGuard, information security.